Assignment 5

Due at 11:59pm on December 5.

Isabel Shaheen O'Malley

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.

```
library(censusapi)
```

```
Attaching package: 'censusapi'

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

    getFunction
```

```
library(tidyverse)
```

```
-- 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   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
v purrr     1.0.2

-- 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 becor
```

```
  library(magrittr)
```

```
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
```

```
  library(ggmap)
```

```
i Google's Terms of Service: <https://mapsplatform.google.com>
  Stadia Maps' Terms of Service: <https://stadiamaps.com/terms-of-service/>
  OpenStreetMap's Tile Usage Policy: <https://operations.osmfoundation.org/policies/tiles/>
i Please cite ggmap if you use it! Use `citation("ggmap")` for details.

Attaching package: 'ggmap'


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

    inset
```

```
  library(lubridate)
  library(corrplot)
```

```
corrplot 0.92 loaded
```

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

Define access key

```
cs_key <- '410ea52de7d0c298684fa54e92f6118f47a4aec9'
```

Get 6 variables of Illinois data from the ACS

```
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.

```
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. Note 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`.

Create a subregion variable in the ACS dataframe, identical to that in the il_map dataframe

```
subregion <- gsub(' County, Illinois', '', acs_il_c$NAME)
subregion <- tolower(subregion)

acs_il_c <- acs_il_c %>%
  mutate(NAME = subregion)

acs_il_c <- acs_il_c %>% rename(subregion = NAME)

head(acs_il_c)
```

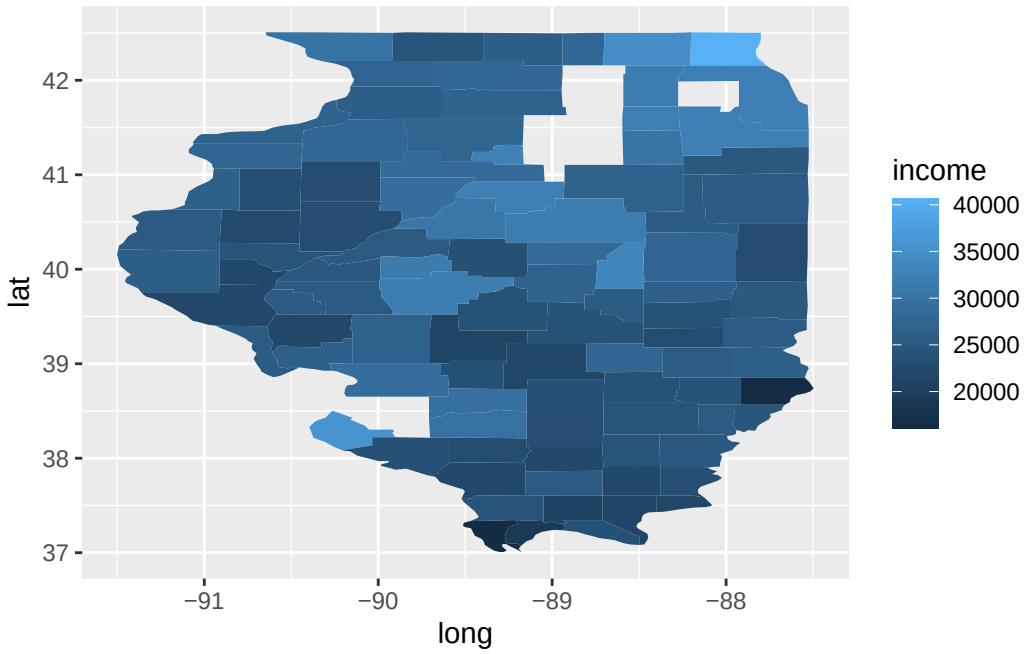
```
  state county subregion    pop hh_income income
1    17    067   hancock  18633     50077  25647
2    17    063    grundy  50338     67162  30232
3    17    091  kankakee 111493     54697  25111
4    17    043    dupage 930514     81521  40547
5    17    003 alexander   7051     29071  16067
6    17    129    menard  12576     60420  31323
```

Join the ACS data with the map data.

```
acs_map <- left_join(acs_il_c, il_map, by = "subregion")
```

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

```
 ggplot(acs_map) +
 geom_polygon(aes(x = long, y = lat, group = group, fill = income))
```



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.

- Create a new subset of the data `acs_il_c` and call it `hclust_data, using only the 4 variables needed for this exercise`

```
hclust_data <-
  acs_il_c %>%
  select(subregion, pop, hh_income, income)
```

- Check for missing values

```
any(is.na(hclust_data))
```

```
[1] FALSE
```

- Check object size

```
length(hclust_data)
```

```
[1] 4
```

Next, create the distance matrix of the cleaned data.

```
hclust_d <- dist(hclust_data)
```

```
Warning in dist(hclust_data): NAs introduced by coercion
```

```
# Warning: NAs introduced by coercion
```

Remove NAs from distance matrix

```
hclust_d <- na.omit(hclust_d)
```

View the distance matrix

```
as.matrix(hclust_d)[1:10, 1:10]
```

```
            1          2           3         4          5          6
1        0.00   41922.57  107359.901 1053715.8   29825.52   15313.69
2    41922.57       0.00   72309.842 1016544.7   68559.81   44311.22
3   107359.90   72309.84       0.000  946398.9  124614.70  114635.16
4  1053715.77 1016544.72  946398.926       0.0 1068415.77 1060277.01
5    29825.52   68559.81  124614.698 1068415.8       0.00   40759.96
6    15313.69   44311.22  114635.158 1060277.0   40759.96       0.00
7   107753.94   73482.80    3745.437  945985.8  124446.76  115322.66
8  6014778.02 5978170.51 5907548.826 4961909.5 6028256.24 6021763.60
9   103714.03   70894.67    9329.274  950233.0  119615.83  111768.38
10   18866.96   29283.60   88917.887 1035166.4   40046.16   30368.96
             7       8           9         10
1   107753.937 6014778  103714.031   18866.96
2    73482.795 5978171   70894.672   29283.60
3     3745.437 5907549    9329.274   88917.89
4   945985.806 4961909  950233.022 1035166.38
5   124446.760 6028256  119615.833   40046.16
6   115322.664 6021764  111768.384   30368.96
7       0.000 5907039    6406.074   89189.73
```

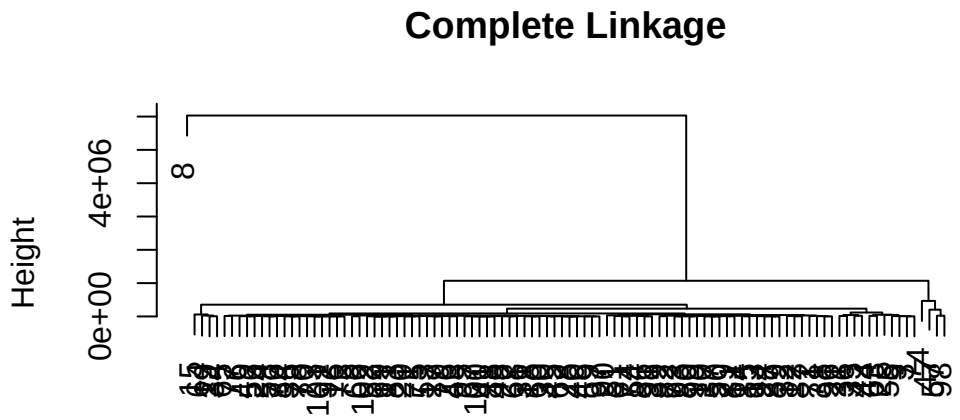
```
8  5907039.332       0 5911122.542 5996113.64
9     6406.074 5911123       0.000   85015.72
10   89189.734 5996114   85015.718       0.00
```

Create the clusters

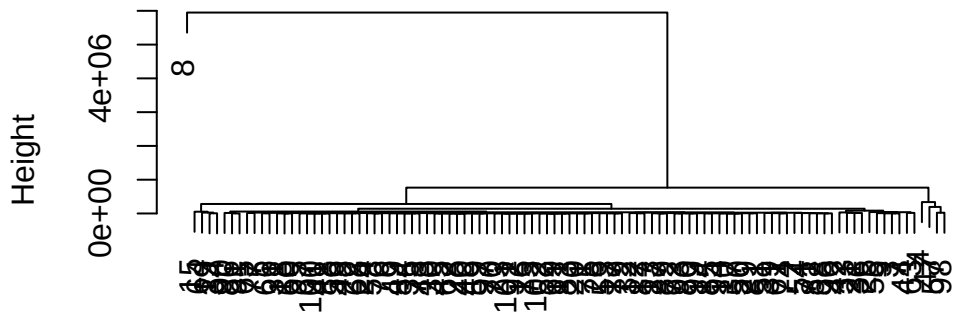
```
hc_complete <- hclust(hclust_d, method = "complete")
hc_average <- hclust(hclust_d, method = "average")
hc_ward <- hclust(hclust_d, method = "ward.D2")
```

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

```
plot(hc_complete, main = "Complete Linkage", xlab = "", sub = "")
```

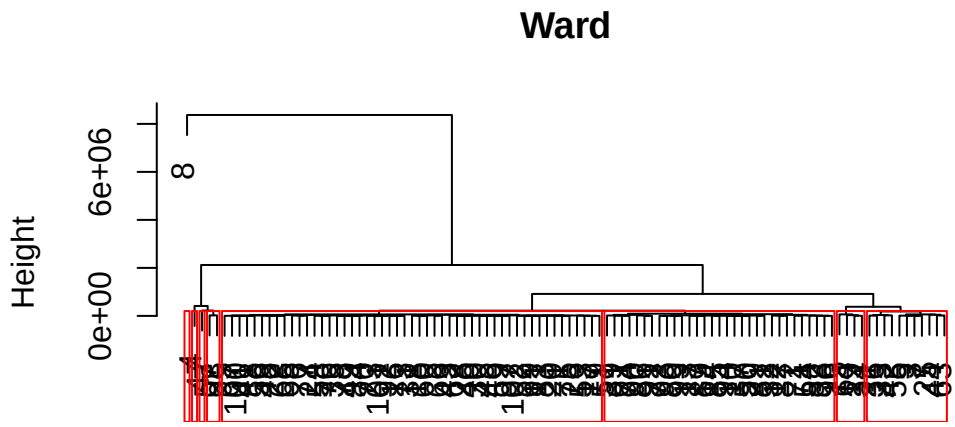



```
plot(hc_average, main = "Average Linkage", xlab = "", sub = "")
```



```
plot(hc_ward, main = "Ward", xlab = "", sub = "")


rect.hclust(hc_ward,
            k = 8,
            border = "red")
```



Create 8 clusters based on the hc_ward object

```
cutree(hc_ward, 8)
```

```
  [1] 1 2 3 4 1 1 3 5 3 2 1 2 2 2 6 1 1 1 6 1 3 1 1 1 3 1 2 1 2 1 1 2 3 6 1 3 2
 [38] 2 2 1 1 3 1 3 1 1 7 1 1 2 2 1 1 2 3 1 8 1 1 1 2 1 3 6 1 1 1 1 1 1 2 2 1 2
 [75] 1 1 1 1 1 2 2 2 1 1 2 2 2 1 2 2 2 1 2 2 1 2 1 8 2 1 1 1
```

Compute the mean of the variables we used to generate the clusters

```
#Select the data
hclust_data <- hclust_data  %>%
  mutate(cluster = cutree(hc_ward, 8)) %>%
  group_by(cluster) %>%
  summarise(mean(pop), mean(hh_income), mean(income), subregion =  names(table(subregion))
```

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`.

```
# acs_map <- acs_map %>%
  # mutate(cluster = cutree(hc_ward, 8))
```

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, .))) %>%
                      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
```

Pull map data for il with tract into a dataframe

```
  # il_map <- map_data("tract", region = "illinois")
  # head(il_map)
```

Join acs map with il map by census tract

```
  # acs_map <- left_join(acs_il_t, il_map, by = "tract")
```

k-Means

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

```
c_data <-
  acs_il_t %>%
  select(tract, pop, hh_income, income)
```

Remove NA values

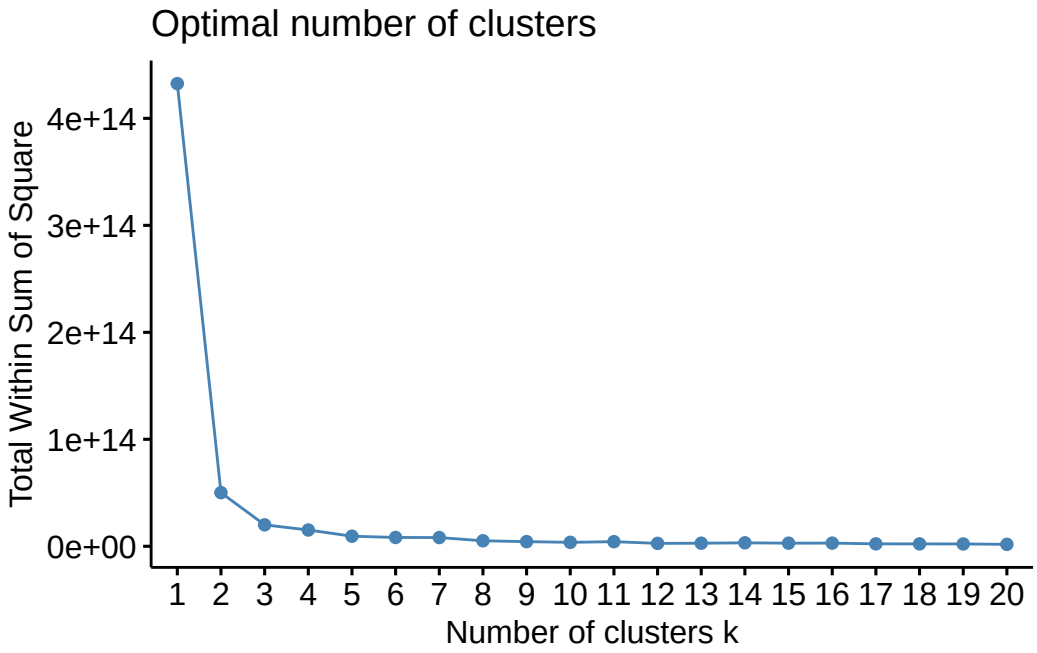
```
c_data <- na.omit(c_data)
```

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. We can run 'kmeans()' to employ K-Means Clustering. Note that 'nstart' can be used to try out several starting points for the clusters. I will use 4 clusters.

```
#Specify the data we are using
# hclust_data
km_1 <- kmeans(c_data, 20, nstart = 20)
# 4 = 4 clusters, #nstart = how many sets of random iterations we want to run
```

Plot within cluster sums of squares for a range of K (e.g. up to 20).

```
fviz_nbclust(c_data, #data set we want to use
             kmeans, #cluster method
             method = "wss", #method used for estimating the optimal number of clusters
             k.max = 20)
```




```
# How similar are the individual data points within each cluster?
# k.max = max number of clusters
# calculates the total within sum of squares for each number of clusters
```

- Based on this plot I think the optimal number of clusters could be 3, 4, or 5. The within sum of squares appear to level out (stop decreasing) at 5.

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

```
km_2 <- kmeans(c_data, 5, nstart = 20)
```

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

```
#Select the data:
acs_il_t %>%
 select(tract, pop, hh_income, income) %>%

#Add clusters - ?

#Compute mean of the variables by cluster
  group_by(cluster) %>%
  summarise(mean(pop), mean(hh_income), mean(income))
```

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.

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.

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