

Assignment 5

Nicole Blackburn

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.

[Click here for Github Link](#)

```
library(censusapi)
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.4      v readr      2.1.5
v forcats    1.0.0      v stringr    1.5.2
v ggplot2    4.0.0      v tibble     3.3.0
v lubridate  1.9.4      v tidyr      1.3.1
v purrr      1.1.0
-- 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 become
```

```
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)
```

Warning: package 'factoextra' was built under R version 4.5.2

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

```
census_key <- read_file("C:/Users/npbvb/OneDrive - University of Maryland/R_Projects/727/APIs/census_key.txt")

acs_il_c <- getCensus(name = "acs/acs5",
                     vintage = 2016,
                     vars = c("NAME",
                              "B01003_001E",
                              "B19013_001E",
                              "B19301_001E"),
                     region = "county:*",
                     regionin = "state:17",
                     key = census_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. 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`.

```
acs_trans <- acs_il_c |>
  mutate(subregion = NAME |>
    str_remove(" County, Illinois.*") |>
    tolower())
acs_trans |>
  filter(!subregion %in% il_map$subregion) |>
  distinct(subregion)
```

	subregion
1	dupage
2	lasalle
3	st. clair
4	dekalb

```
il_map |>
  filter(!subregion %in% acs_trans$subregion) |>
  distinct(subregion)
```

	subregion
1	de kalb
2	du page
3	la salle
4	st clair

```

acs_trans <- acs_trans |>
  mutate(subregion = subregion |>
    str_replace_all("dupage","du page") |>
    str_replace_all("lasalle", "la salle") |>
    str_replace_all("st. clair","st clair") |>
    str_replace_all("dekalb","de kalb"))

acs_map <- il_map |>
  left_join(acs_trans, by = "subregion")

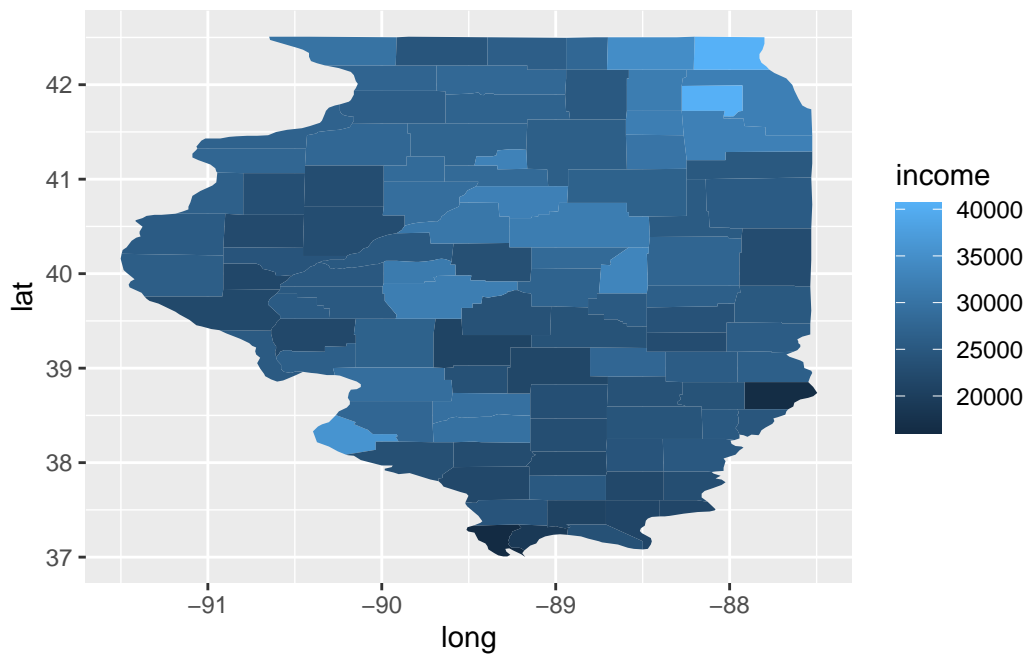
```

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. Next, create the distance matrix of the cleaned data. This distance matrix can be used to cluster counties, e.g. using the ward method.

```
acs_map$pop <- as.numeric(acs_map$pop)
acs_map$hh_income <- as.numeric(acs_map$hh_income)
acs_map$income <- as.numeric(acs_map$income)

cluster <- acs_map |>
  select(pop, hh_income, income) |>
  mutate_all(scale)

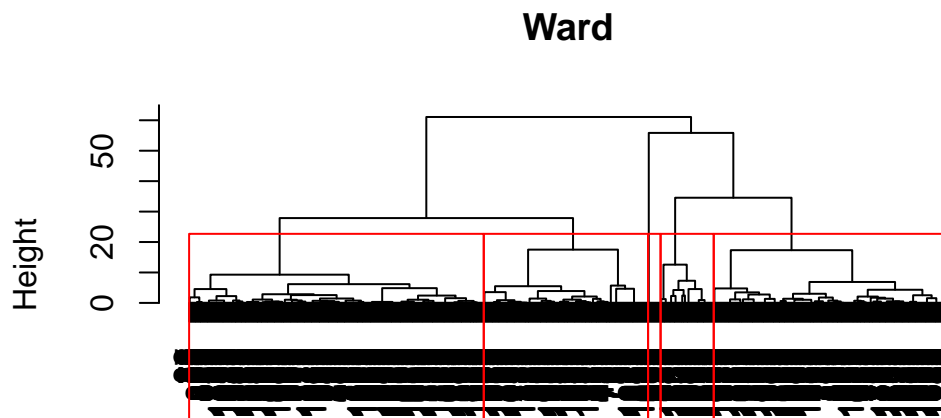
clust_d <- dist(cluster)

clust_ward <- hclust(clust_d, method = "ward.D2")
```

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

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

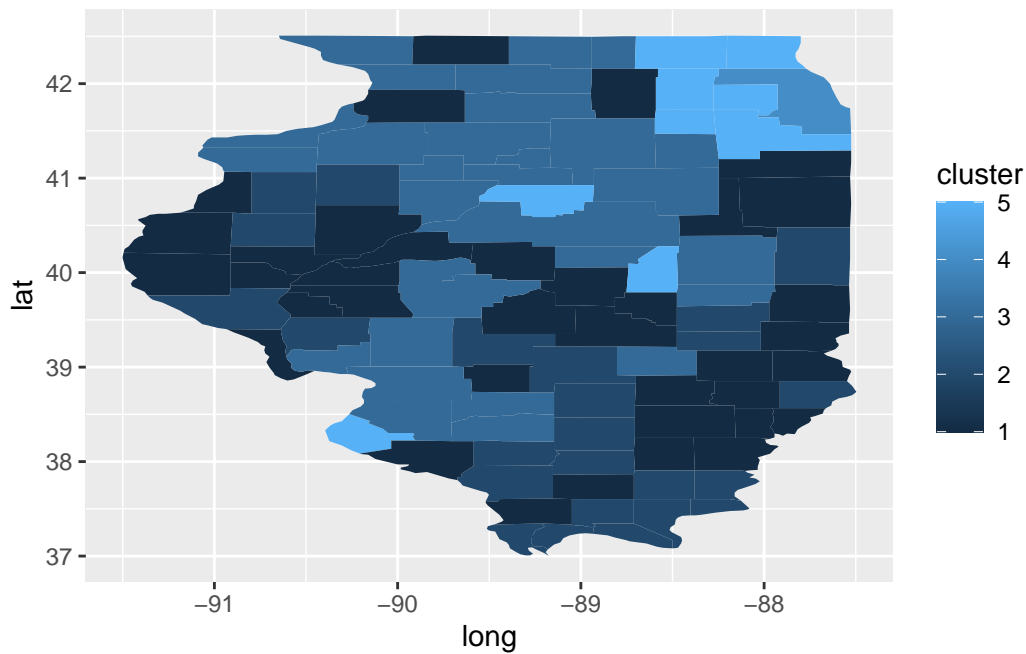
rect.hclust(clust_ward,
  k = 5,
  border = "red")
```



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_cl <- acs_map |>
  mutate(cluster = cutree(clust_ward, 5))

ggplot(acs_map_cl) +
  geom_polygon(aes(x = long,
                  y = lat,
                  group = group,
                  fill = 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 = census_key) %>%
mutate_all(list(~ifelse(.==66666666, 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

k-Means

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

```

acs_il_t$pop <- as.numeric(acs_il_t$pop)
acs_il_t$hh_income <- as.numeric(acs_il_t$hh_income)
acs_il_t$income <- as.numeric(acs_il_t$income)

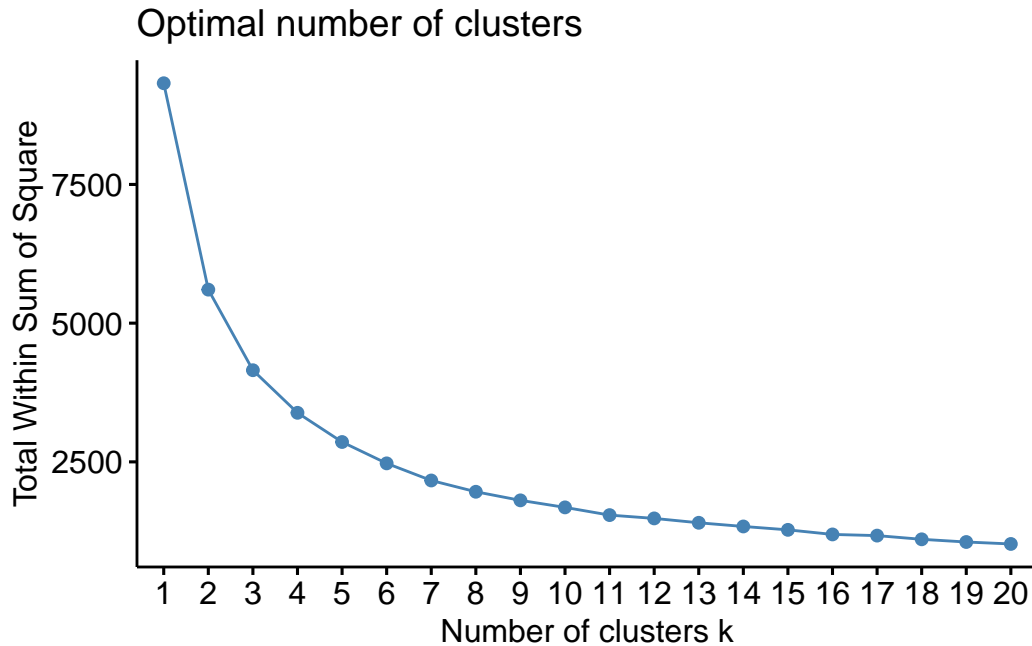
acs_il_t <- na.omit(acs_il_t)

acs2 <- acs_il_t |>
  select(pop, hh_income, income) |>
  mutate_all(scale)

```

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).

```
fviz_nbclust(acs2,  
             kmeans,  
             method = "wss",  
             k.max = 20)
```



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

```
km_1 <- kmeans(acs2, 7, nstart = 20)
```

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.

```
acs_il_t$cluster <- km_1$cluster  
  
acs_il_t <- acs_il_t |>  
  mutate(county = sapply(strsplit(NAME, ", "), `[, 2])  
  
acs_il_t |>  
  group_by(cluster) |>
```



```

summarize(
  pop_mean = mean(pop),
  hh_mean = mean(hh_income),
  inc_mean = mean(income))

# A tibble: 7 x 4
  cluster pop_mean hh_mean inc_mean
  <int>     <dbl>   <dbl>   <dbl>
1       1    5548.  46825.  21881.
2       2    4107. 144266.  80535.
3       3    2569.  32693.  17696.
4       4    6575.  82391.  36858.
5       5    3706.  97293.  50536.
6       6    3370.  60331.  30277.
7       7   14574.  88722.  39686.

acs_il_t |>
  count(cluster, county, sort = T) |>
  group_by(cluster) |>
  slice(1)

```

```

# A tibble: 7 x 3
# Groups:   cluster [7]
  cluster county      n
  <int>   <chr>   <int>
1       1 Cook County  305
2       2 Cook County   70
3       3 Cook County  365
4       4 Cook County  121
5       5 Cook County  176
6       6 Cook County  272
7       7 Kane County    6

```

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.

```

km_f <- function(K) {
  km <- kmeans(acs2, K, nstart = 20)
  return(km)
}

```

We want to utilize this function to iterate over multiple Ks (e.g., $K = 2, \dots, 10$) 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.

```
for (K in 2:10) {
  km <- km_f(K)
  cluster_col <- paste0("cluster_", K)
  acs_il_t[[cluster_col]] <- km$cluster
}
```

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

```
head(acs_il_t)
```

	state	county	tract	NAME	pop			
1	17	Cook County	806002	Census Tract 8060.02, Cook County, Illinois	7304			
2	17	Cook County	806003	Census Tract 8060.03, Cook County, Illinois	7577			
3	17	Cook County	806400	Census Tract 8064, Cook County, Illinois	2684			
4	17	Cook County	806501	Census Tract 8065.01, Cook County, Illinois	2590			
5	17	Cook County	750600	Census Tract 7506, Cook County, Illinois	3594			
6	17	Cook County	310200	Census Tract 3102, Cook County, Illinois	1521			
	hh_income	income	cluster	cluster_2	cluster_3	cluster_4	cluster_5	cluster_6
1	56975	23750	1	1	1	4	3	5
2	53769	25016	1	1	1	4	3	5
3	62750	30154	6	1	3	3	5	3
4	53583	20282	3	1	3	1	4	3
5	40125	18347	3	1	3	1	4	2
6	63250	31403	6	1	3	3	5	3
	cluster_7	cluster_8	cluster_9	cluster_10				
1	5	6	6	1				
2	5	6	6	1				
3	1	8	8	6				
4	3	8	8	6				
5	3	7	9	4				
6	1	8	8	6				