SURV727_assignment5

Zihan Tong

GitHub: https://github.com/tzihan/SURV727_assignment5

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.

```r
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   3.5.2     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 becom
```

```r
library(magrittr)
```

```
Attaching package: 'magrittr'

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

    set_names

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

```r
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

```r
cs_key <- readLines("Census_key.txt")
```

```r
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)
```

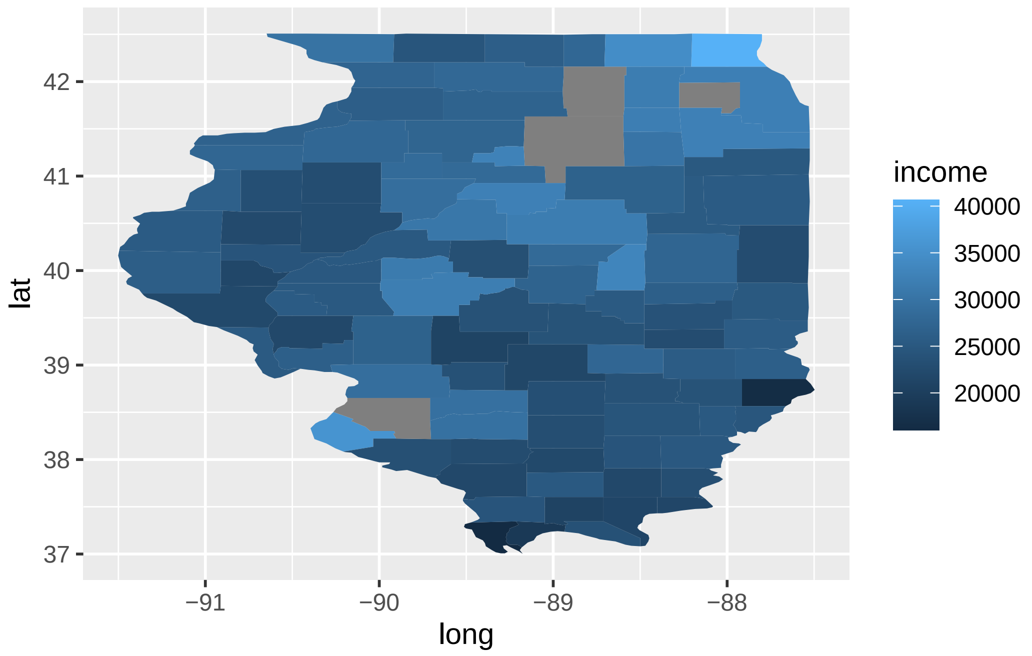
```
       long      lat group order   region subregion
1 -91.49563 40.21018     1     1 illinois     adams
2 -90.91121 40.19299     1     2 illinois     adams
3 -90.91121 40.19299     1     3 illinois     adams
4 -90.91121 40.10704     1     4 illinois     adams
5 -90.91121 39.83775     1     5 illinois     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
acs_il_c <- acs_il_c %>%
  mutate(
    county = NAME %>%
      gsub(" County, Illinois", "", .) %>%
      tolower()
  )

il_map <- il_map %>%
  mutate(subregion = tolower(subregion))

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

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

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



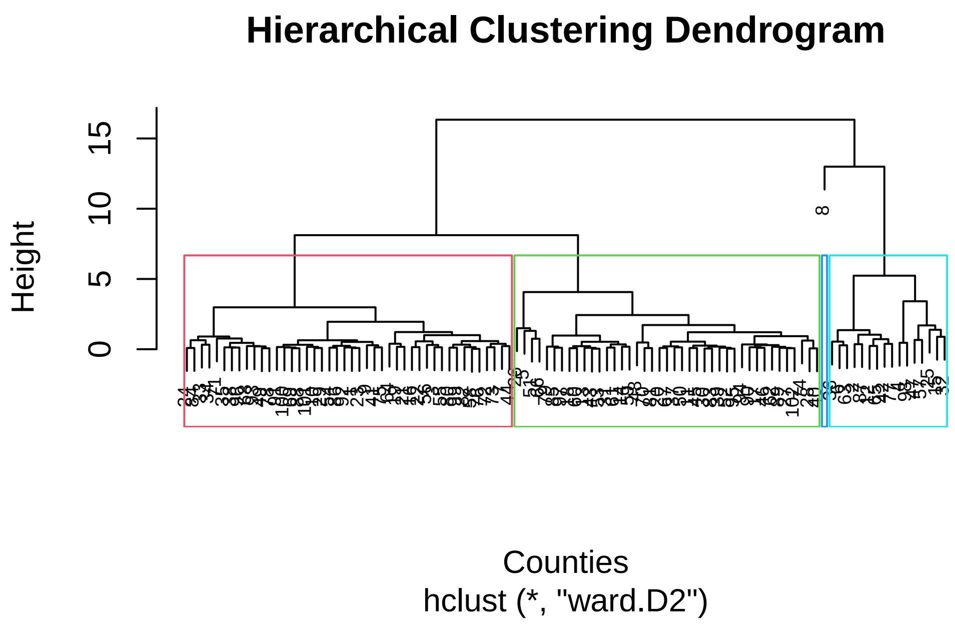
Hierarchical Clustering

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.

```r
clust_data <- acs_il_c %>%
  select(pop, hh_income, income) %>%
  drop_na()

clust_scaled <- scale(clust_data)

dist_matrix <- dist(clust_scaled)

hc <- hclust(dist_matrix, method = "ward.D2")
```

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

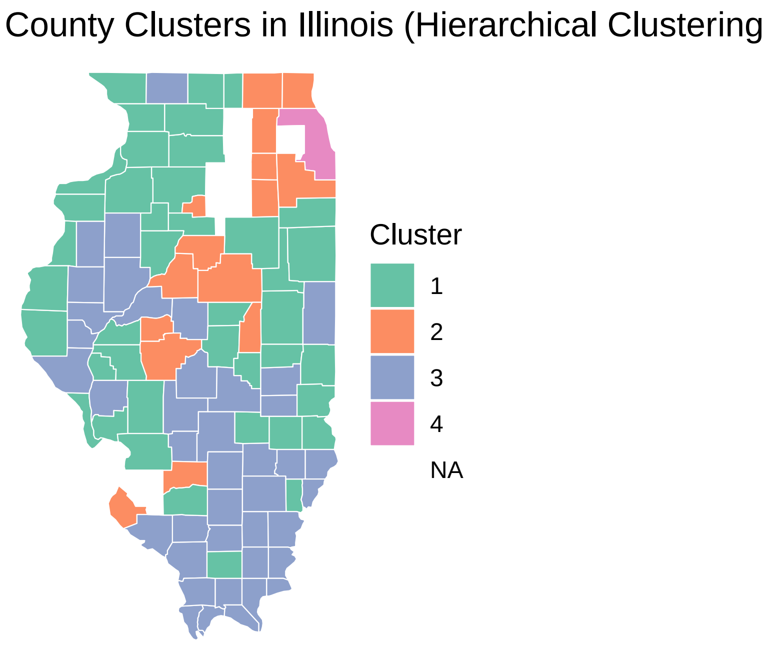
```r
plot(hc, main = "Hierarchical Clustering Dendrogram",
     xlab = "Counties", ylab = "Height", cex = 0.6)
rect.hclust(hc, k = 4, border = 2:5)
```



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`.

```r
clusters <- cutree(hc, k = 4)

acs_il_c$cluster <- clusters

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

ggplot(acs_map) +
  geom_polygon(aes(x = long,
                   y = lat,
                   group = group,
                   fill = factor(cluster)),
               color = "white", size = 0.2) +
  coord_fixed(1.3) +
  scale_fill_brewer(palette = "Set2", name = "Cluster") +
  labs(title = "County Clusters in Illinois (Hierarchical Clustering)") +
  theme_void()
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```



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_all(funs(ifelse(.==-666666666, NA, .))) %>%
            rename(pop = B01003_001E,
                   hh_income = B19013_001E,
                   income = B19301_001E)
```

```
Warning: `funs()` was deprecated in dplyr 0.8.0.
i Please use a list of either functions or lambdas:

# Simple named list: list(mean = mean, median = median)

# Auto named with `tibble::lst()`: tibble::lst(mean, median)

# Using lambdas list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
```

```
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
```

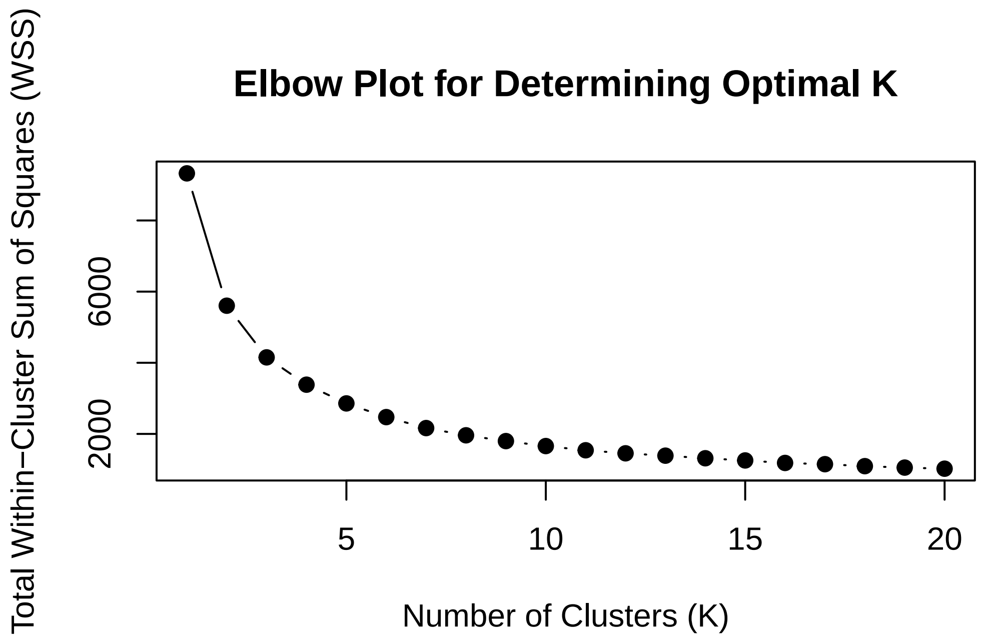
k-Means

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

```
tract_clust_data <- acs_il_t %>%
  select(pop, hh_income, income) %>%
  drop_na()

tract_scaled <- scale(tract_clust_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. Plot within cluster sums of squares for a range of K (e.g. up to 20).

```r
wss_func <- function(k) {
  kmeans(tract_scaled,
         centers = k,
         nstart = 20,
         iter.max = 200)$tot.withinss
}

k_values <- 1:20
wss_values <- sapply(k_values, wss_func)

# Elbow plot
plot(k_values, wss_values,
     type = "b", pch = 19,
     xlab = "Number of Clusters (K)",
     ylab = "Total Within-Cluster Sum of Squares (WSS)",
     main = "Elbow Plot for Determining Optimal K")
```



- From the elbow plot, WSS decreases quickly up to K = 3 or 4, and the curve flattens afterward. Both values are reasonable, but I chose K = 4 because it keeps WSS low while providing a more detailed clustering structure.

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

```
set.seed(123)
kmeans_result <- kmeans(
  tract_scaled,
  centers = 4,
  nstart = 20,
  iter.max = 200
)
```

```
acs_il_t$non_missing <- complete.cases(
  acs_il_t[, c("pop", "hh_income", "income")])

acs_il_t$cluster_k4 <- NA
acs_il_t$cluster_k4[acs_il_t$non_missing] <- kmeans_result$cluster
```

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$county <- sub(".*, (.*) County,.*", "\\1", acs_il_t$NAME)

cluster_summary <- acs_il_t %>%
  filter(!is.na(cluster_k4)) %>%
  group_by(cluster_k4) %>%
  summarize(
    mean_pop = mean(pop, na.rm = TRUE),
    mean_hh_income = mean(hh_income, na.rm = TRUE),
    mean_percap_income = mean(income, na.rm = TRUE),
    .groups = "drop"
  )
cluster_summary
```

```
# A tibble: 4 x 4
  cluster_k4 mean_pop mean_hh_income mean_percap_income
       <int>    <dbl>          <dbl>              <dbl>
1          1    3764.         70344.             34486.
2          2    3137.         37770.             19719.
3          3    4134.        120509.             65518.
4          4    7011.         65178.             29126.
```

- Cluster 3 has the highest household income and per-capita income and represents the most affluent areas.
- Cluster 2 has the lowest values across all indicators and represents economically disadvantaged areas.
- Clusters 1 and 4 fall in the middle in terms of income.
- Cluster 4 has a much larger average population, likely corresponding to more urban or densely populated regions.

```r
county_mode <- acs_il_t %>%
  filter(!is.na(cluster_k4)) %>%
  group_by(cluster_k4) %>%
  summarize(
    most_frequent_county = names(which.max(table(county)))
  )
county_mode
```

```
# A tibble: 4 x 2
  cluster_k4 most_frequent_county
       <int> <chr>
1          1 Cook
2          2 Cook
3          3 Cook
4          4 Cook
```

- The most frequent county in all four clusters is Cook County, likely because Cook has the largest population and the largest number of census tracts, so it dominates the clusters.

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) {
  result <- kmeans(
    x = tract_scaled,
    centers = K,
    nstart = 20,
    iter.max = 200
  )
  return(result$cluster)
}
```

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
for (k in 2:10) {
  clusters <- run_kmeans(k)
  col_name <- paste0("cluster_k", k)
  acs_il_t[[col_name]] <- NA
  acs_il_t[[col_name]][acs_il_t$non_missing] <- clusters
}
```

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

```r
head(acs_il_t)
```

```
  state county  tract                                       NAME  pop
1    17   Cook 806002 Census Tract 8060.02, Cook County, Illinois 7304
2    17   Cook 806003 Census Tract 8060.03, Cook County, Illinois 7577
3    17   Cook 806400    Census Tract 8064, Cook County, Illinois 2684
4    17   Cook 806501 Census Tract 8065.01, Cook County, Illinois 2590
5    17   Cook 750600    Census Tract 7506, Cook County, Illinois 3594
6    17   Cook 310200    Census Tract 3102, Cook County, Illinois 1521
  hh_income income non_missing cluster_k4 cluster_k2 cluster_k3 cluster_k5
1     56975  23750        TRUE          1          2          2          1
2     53769  25016        TRUE          1          2          2          1
3     62750  30154        TRUE          2          2          3          2
4     53583  20282        TRUE          4          2          3          3
5     40125  18347        TRUE          4          2          3          3
6     63250  31403        TRUE          2          2          3          2
  cluster_k6 cluster_k7 cluster_k8 cluster_k9 cluster_k10
1          4          5          2          9           8
2          4          5          2          9           8
3          5          7          6          2           9
4          5          2          6          2           9
5          2          2          1          4           6
6          5          7          6          2           9
```