Prep5

Sebastian Montesinos

Due by midnight, Monday March 21

Reminder: Prep assignments are to be completed individually. Upload a final copy of the .Rmd and renamed .pdf to your private repo, and submit the renamed pdf to Gradescope.

Reading

The associated reading for the week is Chapter 19 and Section 12.1. The topics are text as data and clustering. Each of these topics are potential topics that could be incorporated into your final projects. Clustering is explored further in Stat 240 (Multivariate Data Analysis), and we hope to have a text analytics elective in the future.

1 - Scraping Poems

For our first prep problem, we want to return to Lab 5b, where we played around with scraping the poems of Emily Dickinson from Wikipedia pages. In the Lab folder, open scrape-poems.R, the provided R script for the Lab solution. We want to look at the script before we tackle the data in our next Lab, combining our scraping and text analysis skills. The goal here is to work to understand the script and how it is doing the scraping, as this may be useful for your projects.

Note: If you decide to run the script in its entirety yourself, the process will take a decent length of time (15-30 minutes). However, you can run individual lines (learn about these keyboard shortcuts!) before the loops start to answer the questions that follow.

> part a - In the `collect list` section, two datasets are generated - poem_table and url_table, which are then joined. What is the difference between these two datasets (before the join) and how is that visible in the scraping code?

Solution: The poem_table dataset contains a list of the titles of all of the Emily Dickinson poem's from the webpage. The code grabs the table on the wikipedia page and records all the names. On the other hand, url_table is a combination of a table containing the names of all these poems, and of a table containing the links to all these poem's scraped from the wikipedia page. Therefore, url_table contains two columns, one for the name of the poem and one for its link, while poem_table just contains once column with the names of the poems.

> part b - In the `scrape all poems` section, what does `n_links <- nrow(poem_table)` do to help set up the loop to go through all the poems? Why might that be problematic?

Solution: n_links records the number of links that the table contains and therefore the amount of iterations the code will need to go through. However, the number of rows in the two tables do not match and so there will be a problem when it loops past that number. However, a trycatch is used in case this occurs.

> part c - When scraping the text of one particular poem, the text is found on its webpage in a different way than the other poems. Which poem is problematic in this way?

Solution: 'I never lost as much but twice' is problematic because the normal link does not send you to the page with the poem text.

> part d - What is reported as the poem text if an error occurs while the function is working through the loop?

Solution: "Missing" is reported as the poem text if an error occurs.

> part e - Finally, why is it important to end the file with a write_csv command? I.E. how is this useful?

Solution:The write_csv allows us to store the scraped file so we do not have to redo the scraping every single time we reload the script. This is useful because scraping and writing files can take a while for very large amounts of data.

2 - Text as Data

```
# set up text data provided
data(Macbeth_raw)

# str_split returns a list: we only want the first element
macbeth <- Macbeth_raw %>%
  str_split("\r\n") %>%
  pluck(1)
```

In Section 19.1.1, the `str_subset()`, `str_detect()`, and `str_which()` functions are introduced for detecting a pattern in a character vector (like finding a needle in a haystack). This section also introduces *regular expressions*.

> part a - Explain what the length values and 6 returned records tell us about what each function does below:

```
length(str_subset(macbeth, "  MACBETH"))
```

```
## [1] 147
```

```
str_subset(macbeth, "  MACBETH") %>% head()
```

```
## [1] "  MACBETH, Thane of Glamis and Cawdor, a general in the King's"
## [2] "  MACBETH. So foul and fair a day I have not seen."
## [3] "  MACBETH. Speak, if you can. What are you?"
## [4] "  MACBETH. Stay, you imperfect speakers, tell me more."
## [5] "  MACBETH. Into the air, and what seem'd corporal melted"
## [6] "  MACBETH. Your children shall be kings."
```

```
length(str_which(macbeth, "  MACBETH"))
```

```
## [1] 147
```

```
str_which(macbeth, "  MACBETH") %>% head()
```

```
## [1] 228 433 443 466 478 483
```

```
length(str_detect(macbeth, "  MACBETH"))
```

```
## [1] 3194
```

```
str_detect(macbeth, "  MACBETH") %>% head()
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE
```

Solution: The first code using str_subset finds and returns every line where " MACBETH" is found as a string in the character vector. The code using str_which will tell us us the indicdes where " MACBETH" is found within the character vector. The code using str_detect will also look for every instance of " MACBETH" but will return a vector that is the length of the original vector, with information about whether " MACBETH" was found at each line.

part b - Why do the two lines below differ in their results?

```
str_subset(macbeth, "MACBETH\\.") %>% head()
```

```
## [1] "  MACBETH. So foul and fair a day I have not seen."
## [2] "  MACBETH. Speak, if you can. What are you?"
## [3] "  MACBETH. Stay, you imperfect speakers, tell me more."
## [4] "  MACBETH. Into the air, and what seem'd corporal melted"
## [5] "  MACBETH. Your children shall be kings."
## [6] "  MACBETH. And Thane of Cawdor too. Went it not so?"
```

```
str_subset(macbeth, "MACBETH.") %>% head()
```

```
## [1] "  MACBETH, Thane of Glamis and Cawdor, a general in the King's"
## [2] "  LADY MACBETH, his wife"
## [3] "  MACBETH. So foul and fair a day I have not seen."
## [4] "  MACBETH. Speak, if you can. What are you?"
## [5] "  MACBETH. Stay, you imperfect speakers, tell me more."
## [6] "  MACBETH. Into the air, and what seem'd corporal melted"
```

Solution: '.' is a metacharacter that will match any character, so if you want to look for a literal period you need to put in the two backslashes. Therefore, the first line will return every place where "MACBETH." is found, while the second will just look for MACBETH followed by any character.

part c - The three commands below look similar, but return different results. In words, explain what overall pattern is being searched for in each of the three cases (i.e., what do the patterns `MAC[B-Z]`, `MAC[B|Z]`, and `^MAC[B-Z]` indicate?)?

```
str_subset(macbeth, "MAC[B-Z]") %>% head()
```

```
## [1] "MACHINE READABLE COPIES MAY BE DISTRIBUTED SO LONG AS SUCH COPIES"
## [2] "MACHINE READABLE COPIES OF THIS ETEXT, SO LONG AS SUCH COPIES"
## [3] "WITH PERMISSION.  ELECTRONIC AND MACHINE READABLE COPIES MAY BE"
## [4] "THE TRAGEDY OF MACBETH"
## [5] "  MACBETH, Thane of Glamis and Cawdor, a general in the King's"
## [6] "  LADY MACBETH, his wife"
```

```
str_subset(macbeth,"MAC[B|Z]") %>% head()
```

```
## [1] "THE TRAGEDY OF MACBETH"
## [2] "  MACBETH, Thane of Glamis and Cawdor, a general in the King's"
## [3] "  LADY MACBETH, his wife"
## [4] "  MACBETH. So foul and fair a day I have not seen."
## [5] "  MACBETH. Speak, if you can. What are you?"
## [6] "  MACBETH. Stay, you imperfect speakers, tell me more."
```

```
str_subset(macbeth, "^MAC[B-Z]") %>% head()
```

```
## [1] "MACHINE READABLE COPIES MAY BE DISTRIBUTED SO LONG AS SUCH COPIES"
## [2] "MACHINE READABLE COPIES OF THIS ETEXT, SO LONG AS SUCH COPIES"
```

Solution: MAC[B-Z] looks for any line containing MAC followed by any character other than A. MAC[B|Z] looks for any line containing MAC followed by either a B or a Z. ^MAC[B-Z] looks for any line starting with MAC followed by a b or z at the beginning of the line (the ^ symbol anchors it to the beginning of the line).

Optional - Explore these other patterns to figure out what they do.

```
str_subset(macbeth, ".*MAC[B-Z]") %>% head()
```

```
## [1] "MACHINE READABLE COPIES MAY BE DISTRIBUTED SO LONG AS SUCH COPIES"
## [2] "MACHINE READABLE COPIES OF THIS ETEXT, SO LONG AS SUCH COPIES"
## [3] "WITH PERMISSION.  ELECTRONIC AND MACHINE READABLE COPIES MAY BE"
## [4] "THE TRAGEDY OF MACBETH"
## [5] "  MACBETH, Thane of Glamis and Cawdor, a general in the King's"
## [6] "  LADY MACBETH, his wife"
```

```
str_subset(macbeth, ".MAC[B-Z]") %>% head()
```

```
## [1] "WITH PERMISSION.  ELECTRONIC AND MACHINE READABLE COPIES MAY BE"
## [2] "THE TRAGEDY OF MACBETH"
## [3] "  MACBETH, Thane of Glamis and Cawdor, a general in the King's"
## [4] "  LADY MACBETH, his wife"
## [5] "  MACDUFF, Thane of Fife, a nobleman of Scotland"
## [6] "  LADY MACDUFF, his wife"
```

```
str_subset(macbeth, "more$") %>% head()
```

```
## [1] "    Who, almost dead for breath, had scarcely more"
## [2] "    Hath left you unattended. [Knocking within.] Hark, more"
## [3] "more"
```

```
# Code below should return character(0) (i.e., nothing)
str_subset(macbeth, "^MAC[B|Z]") %>% head()
```

```
## character(0)
```

Notes:

3 - K-means clustering

Section 12.1.2 walks through an example of how k-means clustering can identify genuine patterns in data—in this case, clustering cities into continental groups merely based on city location (longitude and latitude coordinates). The textbook code is modified below to parse out some of the steps for using the `kmeans()` function of the **mclust** package and to add the centroid locations to a reproduction of Figure 12.4. The data for this example comes from the **mdsr** package.

> part a - Run the code below to implement the k-means algorithm. Take a look at the resulting output for each object in the code chunk in some way (print in the console, `glimpse()`, or `head()`).

```
data(world_cities)

# Identify the 4,000 biggest cities in the world
big_cities <- world_cities %>%
  arrange(desc(population)) %>%
  head(4000) %>%
  select(longitude, latitude)

# Make sure to set seed for reproducibility!
set.seed(15)
city_kmeans_results <- big_cities %>%
  kmeans(centers = 6, nstart = 30)
```

Solution: glimpse(big_cities) glimpse(city_kmeans_results)

> part b - The code below reports on the class of the `city_kmeans_results` and the named elements we can refer to and select from that class. What type of object is `city_kmeans_results`? What named elements does the object contain and what information does each element give us?

Hint: the **Value** section of the help documentation will help you understand what each named element is.

```
# Check object class or type
class(city_kmeans_results)
```

```
## [1] "kmeans"
```

```
# Check named elements of object
names(city_kmeans_results)
```

```
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Solution: city_kmeans_results is of the 'kmeans' class. The named elements we can refer to from that class are cluster (A vector of integers that indicates which cluster each point is allocated to), centers (A matrix of the cluster centres), totss (The sum of squares), withinss (the within-cluster sum of squares for each cluster), tot.withinss (The total within cluster sum of squares), betweenss (the between cluster sum of squares), size (the number of points in each cluster), iter (the number of iterations), and ifault (an integer indicating possible algorithm problems).

part c - Where are the estimated centroids of the 6 clusters? How many cities were assigned to the first cluster? Which cluster contains the most cities?

```
# Centroids
city_kmeans_results$centers
```

```
##   longitude  latitude
## 1  75.14407  27.43226
## 2 -94.47442  31.07927
## 3 120.38618  23.72534
## 4 -57.10048 -15.21344
## 5  18.91076  -1.12440
## 6  18.77544  45.37853
```

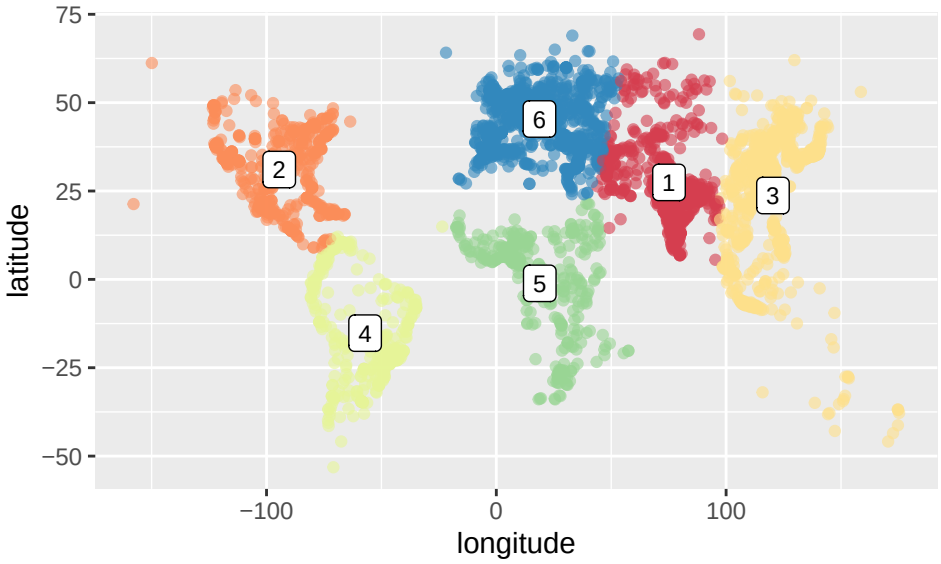
```
# Cluster sizes
city_kmeans_results$size
```

```
## [1] 726 554 984 392 356 988
```

Solution: The estimated centroids are at ~75 long 27 lat, ~-94 long 31 lat, ~120 long 32 lat, ~-57 long ~15 lat, ~17 long -1 lat, and ~18 long 45 lat. That final location at ~18 long and 45 lat contains the most points at 988 cities.

part d - Run the code below to create a plot of the assigned clusters. What do you think Cluster 2 represents?

```
# Join cluster assignments with original dataset
big_cities <- big_cities %>%
  add_column(cluster = factor(city_kmeans_results$cluster))

# Make dataframe out of estimated cluster centroids
city_cluster_centers <- city_kmeans_results$centers %>%
  data.frame() %>%
  add_column(cluster_number = 1:6)

# Plot cluster assignments and centroids
ggplot(big_cities, aes(x = longitude, y = latitude)) +
  geom_point(aes(color = cluster), alpha = 0.6)  +
  scale_color_brewer(palette = "Spectral") +
  geom_label(data = city_cluster_centers,
             aes(label = cluster_number),
             size = 3) +
  theme(legend.position = "none")
```



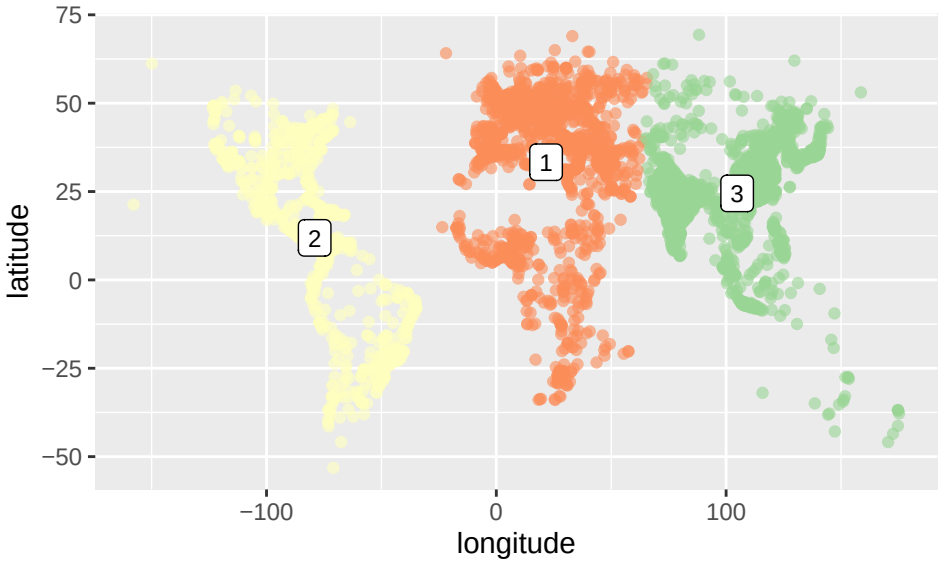
Solution: Cluster 2 represents the cities that cluster around the general longitude/latitude of the North America.

part e - In k-means clustering, the analyst specifies the number of clusters to create. Update the `center` argument within the `kmeans()` function to identify 3 clusters instead of 6. Create a plot like the one above, but coloring the points by these new cluster assignments. How many cities are in Cluster 1 now? What does Cluster 2 represent now?

Solution: Cluster 1 now contains 1466 cities. Cluster 2 now represents the cities that cluster around the general longitude/latitude of both North and South America.

```
# Be sure to set a seed for reproducibility
big_cities2 <- world_cities %>%
  arrange(desc(population)) %>%
  head(4000) %>%
  select(longitude, latitude)

# Make sure to set seed for reproducibility!
set.seed(15)
city_kmeans_results2 <- big_cities2 %>%
  kmeans(centers = 3, nstart = 30)

big_cities2 <- big_cities2 %>%
  add_column(cluster = factor(city_kmeans_results2$cluster))

# Make dataframe out of estimated cluster centroids
city_cluster_centers2 <- city_kmeans_results2$centers %>%
  data.frame() %>%
  add_column(cluster_number = 1:3)

# Plot cluster assignments and centroids
ggplot(big_cities2, aes(x = longitude, y = latitude)) +
  geom_point(aes(color = cluster), alpha = 0.6)  +
  scale_color_brewer(palette = "Spectral") +
  geom_label(data = city_cluster_centers2,
```

```
            aes(label = cluster_number),
            size = 3) +
theme(legend.position = "none")
```



“‘