Week 6 - Challenge - Company Segmentation

Business Science

2/19/2019

Contents

Challenge Summary

Your organization wants to know which companies are similar to each other to help in identifying potential customers of a SAAS software solution (e.g. Salesforce CRM or equivalent) in various segments of the market. The Sales Department is very interested in this analysis, which will help them more easily penetrate various market segments.

You will be using stock prices in this analysis. You come up with a method to classify companies based on how their stocks trade using their daily stock returns (percentage movement from one day to the next). This analysis will help your organization determine which companies are related to each other (competitors and have similar attributes).

You can analyze the stock prices using what you've learned in the unsupervised learning tools including K-Means and UMAP. You will use a combination of `kmeans()` to find groups and `umap()` to visualize similarity of daily stock returns.

Objectives

Apply your knowledge on K-Means and UMAP along with `dplyr`, `ggplot2`, and `purrr` to create a visualization that identifies subgroups in the S&P 500 Index. You will specifically apply:

- Modeling: `kmeans()` and `umap()`
- Iteration: `purrr`
- Data Manipulation: `dplyr`, `tidyr`, and `tibble`
- Visualization: `ggplot2` (bonus `plotly`)

Libraries

Load the following libraries. If you have never used `plotly` for interactive plotting, you will need to install with `install.packages("plotly")`.

```r
# install.packages("plotly")

library(tidyverse)
library(tidyquant)
library(broom)
library(umap)
library(plotly) # NEW PACKAGE
library(quantmod)
```

```r
interactive <- FALSE
```

Data

We will be using stock prices in this analysis. The `tidyquant` R package contains an API to retreive stock prices. The following code is shown so you can see how I obtained the stock prices for every stock in the S&P 500 index. The files are saved in the `week_6_data` directory.

```r
# # NOT RUN - WILL TAKE SEVERAL MINUTES TO DOWNLOAD ALL THE STOCK PRICES
# # JUST SHOWN FOR FUN SO YOU KNOW HOW I GOT THE DATA
#
# # GET ALL STOCKS IN A STOCK INDEX (E.G. SP500)
# sp_500_index_tbl <- tq_index("SP500")
# sp_500_index_tbl
#
# # PULL IN STOCK PRICES FOR EACH STOCK IN THE INDEX
# sp_500_prices_tbl <- sp_500_index %>%
#     select(symbol) %>%
#     tq_get(get = "stock.prices")
#
# # SAVING THE DATA
# fs::dir_create("week_6_data")
# sp_500_prices_tbl %>% write_rds(path = "week_6_data/sp_500_prices_tbl.rds")
# sp_500_index_tbl %>% write_rds("week_6_data/sp_500_index_tbl.rds")
```

```
#SP500_list
```

We can read in the stock prices. The data is 1.2M observations. The most important columns for our analysis are:

- `symbol`: The stock ticker symbol that corresponds to a company's stock price
- `date`: The timestamp relating the symbol to the share price at that point in time
- `adjusted`: The stock price, adjusted for any splits and divdends (we use this when analyzing stock data over long periods of time)

```
# get_stock_list <- function(stock_index = "SP500") {
#         tq_index(stock_index) %>%
#         select(symbol, company) %>%
#         arrange(symbol) %>%
#         mutate(label = str_c(symbol, company, sep = ', ')) %>%
#         select(label)
# }
# SP500_list <- get_stock_list("SP500")
#
# get_symbol_from_user_input <- function(user_input) {
#     user_input %>% str_split(pattern = ", ") %>%
#         purrr::pluck(1,1)
# }
#
# get_stock <- function(stock_symbol,
#          from = today() - lubridate::days(3696),
#          to = today()){
#     stock_symbol %>%
#         tq_get(get = "stock.prices", from = from, to = to) %>%
#         select(-symbol)
# }
#
#
# SP500_stock_mapped_tbl  <- SP500_list %>%
#     mutate(symbol = label %>% map(get_symbol_from_user_input)) %>%
#     mutate(get = symbol %>% map(get_stock))
#
# SP500_stock_tbl <- SP500_stock_mapped_tbl %>% unnest() %>% select(-symbol1)
#
```

```
# STOCK PRICES
sp_500_prices_tbl <- read_rds("challenge/week_6_data/sp_500_prices_tbl.rds")
sp_500_prices_tbl %>% head(10)
```

```
## # A tibble: 10 x 8
##    symbol date        open  high   low close   volume adjusted
##    <chr>  <date>     <dbl> <dbl> <dbl> <dbl>    <dbl>    <dbl>
##  1 MSFT   2009-01-02  19.5  20.4  19.4  20.3 50084000     15.9
##  2 MSFT   2009-01-05  20.2  20.7  20.1  20.5 61475200     16.0
##  3 MSFT   2009-01-06  20.8  21    20.6  20.8 58083400     16.2
##  4 MSFT   2009-01-07  20.2  20.3  19.5  19.5 72709900     15.2
##  5 MSFT   2009-01-08  19.6  20.2  19.5  20.1 70255400     15.7
```

```
## 6 MSFT   2009-01-09  20.2  20.3  19.4  19.5 49815300     15.2
## 7 MSFT   2009-01-12  19.7  19.8  19.3  19.5 52163500     15.2
## 8 MSFT   2009-01-13  19.5  20.0  19.5  19.8 65843500     15.5
## 9 MSFT   2009-01-14  19.5  19.7  19.0  19.1 80257500     14.9
## 10 MSFT  2009-01-15  19.1  19.3  18.5  19.2 96169800     15.0
```

The second data frame contains information about the stocks the most important of which are:

- `company`: The company name
- `sector`: The sector that the company belongs to

```
# SECTOR INFORMATION
sp_500_index_tbl <- read_rds("challenge/week_6_data/sp_500_index_tbl.rds")
sp_500_index_tbl %>% head(10)
```

```
##    symbol                      company     weight                 sector
## 1    MSFT        Microsoft Corporation 0.03589659 Information Technology
## 2    AAPL                   Apple Inc. 0.03299844 Information Technology
## 3    AMZN              Amazon.com Inc. 0.02834845 Consumer Discretionary
## 4   BRK.B Berkshire Hathaway Inc. Class B 0.01714493             Financials
## 5      FB        Facebook Inc. Class A 0.01676060 Communication Services
## 6     JNJ            Johnson & Johnson 0.01570168            Health Care
## 7     JPM         JPMorgan Chase & Co. 0.01507235             Financials
## 8    GOOG        Alphabet Inc. Class C 0.01470747 Communication Services
## 9   GOOGL        Alphabet Inc. Class A 0.01436854 Communication Services
## 10    XOM      Exxon Mobil Corporation 0.01412361                 Energy
##    shares_held
## 1     84853600
## 2     49533308
## 3      4510051
## 4     21364490
## 5     26385216
## 6     29452358
## 7     36529800
## 8      3378423
## 9      3282939
## 10    46493644
```

Question

Which stock prices behave similarly?

Answering this question helps us **understand which companies are related**, and we can use clustering to help us answer it!

Even if you're not interested in finance, this is still a great analysis because it will tell you which companies are competitors and which are likely in the same space (often called sectors) and can be categorized together. Bottom line - This analysis can help you better understand the dynamics of the market and competition, which is useful for all types of analyses from finance to sales to marketing.

Let's get started.

Step 1 - Convert stock prices to a standardized format (daily returns)

What you first need to do is get the data in a format that can be converted to a "user-item" style matrix. The challenge here is to connect the dots between what we have and what we need to do to format it properly.

We know that in order to compare the data, it needs to be standardized or normalized. Why? Because we cannot compare values (stock prices) that are of completely different magnitudes. In order to standardize, we will convert from adjusted stock price (dollar value) to daily returns (percent change from previous day). Here is the formula.

$$return_{daily} = \frac{price_i - price_{i-1}}{price_{i-1}}$$

First, what do we have? We have stock prices for every stock in the SP 500 Index, which is the daily stock prices for over 500 stocks. The data set is over 1.2M observations.

```
sp_500_prices_tbl %>% glimpse()
```

```
## Rows: 1,225,765
## Columns: 8
## $ symbol   <chr> "MSFT", "MSFT", "MSFT", "MSFT", "MSFT", "MSFT", "MSFT", "MSFT~
## $ date     <date> 2009-01-02, 2009-01-05, 2009-01-06, 2009-01-07, 2009-01-08, ~
## $ open     <dbl> 19.53, 20.20, 20.75, 20.19, 19.63, 20.17, 19.71, 19.52, 19.53~
## $ high     <dbl> 20.40, 20.67, 21.00, 20.29, 20.19, 20.30, 19.79, 19.99, 19.68~
## $ low      <dbl> 19.37, 20.06, 20.61, 19.48, 19.55, 19.41, 19.30, 19.52, 19.01~
## $ close    <dbl> 20.33, 20.52, 20.76, 19.51, 20.12, 19.52, 19.47, 19.82, 19.09~
## $ volume   <dbl> 50084000, 61475200, 58083400, 72709900, 70255400, 49815300, 5~
## $ adjusted <dbl> 15.86624, 16.01451, 16.20183, 15.22628, 15.70234, 15.23408, 1~
```

Your first task is to convert to a tibble named `sp_500_daily_returns_tbl` by performing the following operations:

- Select the `symbol`, `date` and `adjusted` columns
- Filter to dates beginning in the year 2018 and beyond.
- Compute a Lag of 1 day on the adjusted stock price. Be sure to group by symbol first, otherwise we will have lags computed using values from the previous stock in the data frame.
- Remove an `NA` values from the lagging operation
- Compute the difference between adjusted and the lag
- Compute the percentage difference by dividing the difference by that lag. Name this column `pct_return`.
- Return only the `symbol`, `date`, and `pct_return` columns
- Save as a variable named `sp_500_daily_returns_tbl`

```
# Apply your data transformation skills!
sp_500_daily_returns_tbl <- sp_500_prices_tbl %>%
    select(symbol, date, adjusted) %>%
    # Filter to dates beginning in the year 2018 and beyond
    group_by(symbol) %>%
    mutate(year  = year(date)) %>% ungroup() %>%
    filter(year >= 2018) %>%
    select(-year) %>%

    # Compute a Lag of 1 day on the adjusted stock price.
```

```
    group_by(symbol) %>%
    mutate(lag = lag(adjusted, order_by = date, n = 1)) %>%
    filter(!is.na(lag)) %>%
    mutate(diff = adjusted - lag) %>%
    mutate(pct_return = diff/lag)  %>% ungroup() %>%
    select(symbol, date, pct_return)

# Output: sp_500_daily_returns_tbl
sp_500_daily_returns_tbl %>% head(20)
```

```
## # A tibble: 20 x 3
##    symbol date       pct_return
##    <chr>  <date>          <dbl>
##  1 MSFT   2018-01-03   0.00465
##  2 MSFT   2018-01-04   0.00880
##  3 MSFT   2018-01-05   0.0124
##  4 MSFT   2018-01-08   0.00102
##  5 MSFT   2018-01-09  -0.000680
##  6 MSFT   2018-01-10  -0.00453
##  7 MSFT   2018-01-11   0.00296
##  8 MSFT   2018-01-12   0.0173
##  9 MSFT   2018-01-16  -0.0140
## 10 MSFT   2018-01-17   0.0203
## 11 MSFT   2018-01-18  -0.000444
## 12 MSFT   2018-01-19  -0.00111
## 13 MSFT   2018-01-22   0.0179
## 14 MSFT   2018-01-23   0.00317
## 15 MSFT   2018-01-24  -0.000870
## 16 MSFT   2018-01-25   0.00555
## 17 MSFT   2018-01-26   0.0187
## 18 MSFT   2018-01-29  -0.00149
## 19 MSFT   2018-01-30  -0.0126
## 20 MSFT   2018-01-31   0.0245
```

Step 2 - Convert to User-Item Format

The next step is to convert to a user-item format with the `symbol` in the first column and every other column the value of the *daily returns* (`pct_return`) for every stock at each `date`.

We're going to import the correct results first (just in case you were not able to complete the last step).

```
# sp_500_daily_returns_tbl <- read_rds("challenge/week_6_data/sp_500_daily_returns_tbl.rds")
```

Now that we have the daily returns (percentage change from one day to the next), we can convert to a user-item format. The user in this case is the `symbol` (company), and the item in this case is the `pct_return` at each `date`.

- Spread the `date` column to get the values as percentage returns. Make sure to fill an `NA` values with zeros.
- Save the result as `stock_date_matrix_tbl`

```r
# Convert to User-Item Format
stock_date_matrix_tbl <- sp_500_daily_returns_tbl %>%
    spread(date, pct_return, fill = 0)
# Output: stock_date_matrix_tbl
stock_date_matrix_tbl %>% head(20)
```

```
## # A tibble: 20 x 283
##    symbol `2018-01-03` `2018-01-04` `2018-01-05` `2018-01-08` `2018-01-09`
##    <chr>         <dbl>        <dbl>        <dbl>        <dbl>        <dbl>
##  1 A          0.0254       -0.00750     0.0160        0.00215     0.0246
##  2 AAL       -0.0123        0.00630    -0.000380     -0.00988    -0.000959
##  3 AAP        0.00905       0.0369      0.0106       -0.00704    -0.00808
##  4 AAPL      -0.000174      0.00465     0.0114       -0.00371    -0.000115
##  5 ABBV       0.0156       -0.00570     0.0174       -0.0160      0.00754
##  6 ABC        0.00372      -0.00222     0.0121        0.0166      0.00640
##  7 ABMD       0.0173        0.0175      0.0154        0.0271      0.00943
##  8 ABT        0.00221      -0.00170     0.00289      -0.00288     0.00170
##  9 ACN        0.00462       0.0118      0.00825       0.00799     0.00333
## 10 ADBE       0.0188        0.0120      0.0116       -0.00162     0.00897
## 11 ADI        0.0124       -0.00109     0.00405       0.00175    -0.00207
## 12 ADM       -0.00773       0.0168     -0.00667      -0.00224     0.00324
## 13 ADP        0.0109        0.00955    -0.000591     -0.00304     0.00695
## 14 ADS        0.0186        0.0153      0.00269      -0.00551     0.0177
## 15 ADSK       0.0211        0.0246     -0.0110        0.00523     0.00619
## 16 AEE       -0.00514      -0.0114     -0.000697      0.0118     -0.0129
## 17 AEP       -0.00842      -0.0118     -0.00211       0.00876    -0.0118
## 18 AES       -0.000919     -0.00368     0.00369       0           -0.0101
## 19 AFL        0.00296       0.0103      0.00662       0.00256     0.000445
## 20 AGN       -0.00106       0.00846     0.00414      -0.0100      0.0298
## # ... with 277 more variables: 2018-01-10 <dbl>, 2018-01-11 <dbl>,
## #   2018-01-12 <dbl>, 2018-01-16 <dbl>, 2018-01-17 <dbl>, 2018-01-18 <dbl>,
## #   2018-01-19 <dbl>, 2018-01-22 <dbl>, 2018-01-23 <dbl>, 2018-01-24 <dbl>,
## #   2018-01-25 <dbl>, 2018-01-26 <dbl>, 2018-01-29 <dbl>, 2018-01-30 <dbl>,
## #   2018-01-31 <dbl>, 2018-02-01 <dbl>, 2018-02-02 <dbl>, 2018-02-05 <dbl>,
## #   2018-02-06 <dbl>, 2018-02-07 <dbl>, 2018-02-08 <dbl>, 2018-02-09 <dbl>,
## #   2018-02-12 <dbl>, 2018-02-13 <dbl>, 2018-02-14 <dbl>, 2018-02-15 <dbl>, ...
```

Step 3 - Perform K-Means Clustering

Next, we'll perform **K-Means clustering**.

We're going to import the correct results first (just in case you were not able to complete the last step).

```r
# stock_date_matrix_tbl <- read_rds("challenge/week_6_data/stock_date_matrix_tbl.rds")
```

Beginning with the `stock_date_matrix_tbl`, perform the following operations:

- Drop the non-numeric column, `symbol`
- Perform `kmeans()` with `centers = 4` and `nstart = 20`
- Save the result as `kmeans_obj`

```r
# Create kmeans_obj for 4 centers
kmeans_obj <- stock_date_matrix_tbl %>%
    select(-symbol) %>%
    kmeans(centers = 4, nstart = 20)
```

Use `glance()` to get the `tot.withinss`.

```r
# Apply glance() to get the tot.withinss
kmeans_obj %>% broom::glance()
```

```
## # A tibble: 1 x 4
##   totss tot.withinss betweenss  iter
##   <dbl>        <dbl>     <dbl> <int>
## 1  33.6         29.2      4.40     5
```

Step 4 - Find the optimal value of K

Now that we are familiar with the process for calculating `kmeans()`, let's use `purrr` to iterate over many values of "k" using the `centers` argument.

We'll use this **custom function** called `kmeans_mapper()`:

```r
kmeans_mapper <- function(center = 3) {
    stock_date_matrix_tbl %>%
        select(-symbol) %>%
        kmeans(centers = center, nstart = 20)
}
```

Apply the `kmeans_mapper()` and `glance()` functions iteratively using `purrr`.

- Create a tibble containing column called `centers` that go from 1 to 30
- Add a column named `k_means` with the `kmeans_mapper()` output. Use `mutate()` to add the column and `map()` to map centers to the `kmeans_mapper()` function.
- Add a column named `glance` with the `glance()` output. Use `mutate()` and `map()` again to iterate over the column of `k_means`.
- Save the output as `k_means_mapped_tbl`

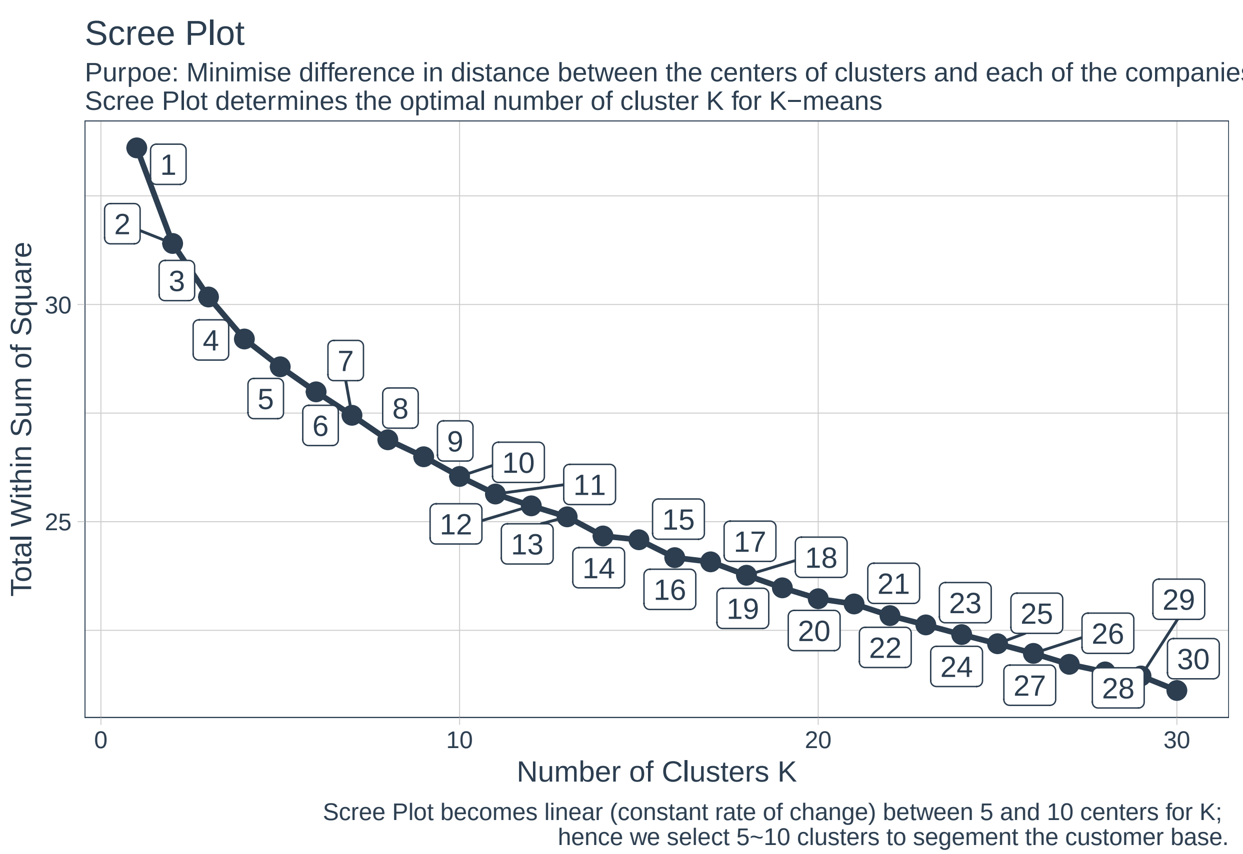
```r
# Use purrr to map
centers_tbl <- tibble(centers = 1:30)

k_means_mapped_tbl <- centers_tbl %>%
    mutate(k_means = centers %>% map(kmeans_mapper),
           glance = k_means %>% map(broom::glance))
# Output: k_means_mapped_tbl
```

Next, let's visualize the "tot.withinss" from the glance output as a ***Scree Plot***.

- Begin with the `k_means_mapped_tbl`
- Unnest the `glance` column
- Plot the `centers` column (x-axis) versus the `tot.withinss` column (y-axis) using `geom_point()` and `geom_line()`

- Add a title "Scree Plot" and feel free to style it with your favorite theme

```r
# Visualize Scree Plot
k_means_mapped_tbl %>%
    unnest(glance) %>%
    ggplot(aes(x = centers, y = tot.withinss), colour = "#2c3e50") +
    geom_point(colour = "#2c3e50", size = 3) +
    geom_line(colour = "#2c3e50", size = 1) +
    ggrepel::geom_label_repel(aes(label = centers), colour = "#2c3e50", size = 4) +
    theme_tq() +
    labs(
        x = "Number of Clusters K",
        y = "Total Within Sum of Square",
        title = "Scree Plot",
        subtitle = "Purpoe: Minimise difference in distance between the centers of clusters and each of
Scree Plot determines the optimal number of cluster K for K-means",
        caption = "Scree Plot becomes linear (constant rate of change) between 5 and 10 centers for K;
        hence we select 5~10 clusters to segement the customer base."
    )
```



We can see that the Scree Plot becomes linear (constant rate of change) between 5 and 10 centers for K.

Step 5 - Apply UMAP

Next, let's plot the `UMAP` 2D visualization to help us investigate cluster assignments.

We're going to import the correct results first (just in case you were not able to complete the last step).

```
# k_means_mapped_tbl <- read_rds("challenge/week_6_data/k_means_mapped_tbl.rds")
```

First, let's apply the `umap()` function to the `stock_date_matrix_tbl`, which contains our user-item matrix in tibble format.

- Start with `stock_date_matrix_tbl`
- De-select the `symbol` column
- Use the `umap()` function storing the output as `umap_results`

```
# Apply UMAP
umap_results <- stock_date_matrix_tbl %>%
    select(-symbol) %>%
    umap()

# Store results as: umap_results
```

Next, we want to combine the `layout` from the `umap_results` with the `symbol` column from the `stock_date_matrix_tbl`.

- Start with `umap_results$layout`
- Convert from a `matrix` data type to a `tibble` with `as_tibble()`
- Bind the columns of the umap tibble with the `symbol` column from the `stock_date_matrix_tbl`.
- Save the results as `umap_results_tbl`.

```
# Convert umap results to tibble with symbols
umap_results_tbl <- umap_results$layout %>%
    as_tibble() %>%
    setNames(c("x", "y")) %>%
    cbind(stock_date_matrix_tbl %>% select(symbol))
# Output: umap_results_tbl

umap_results_tbl %>% head(20)
```

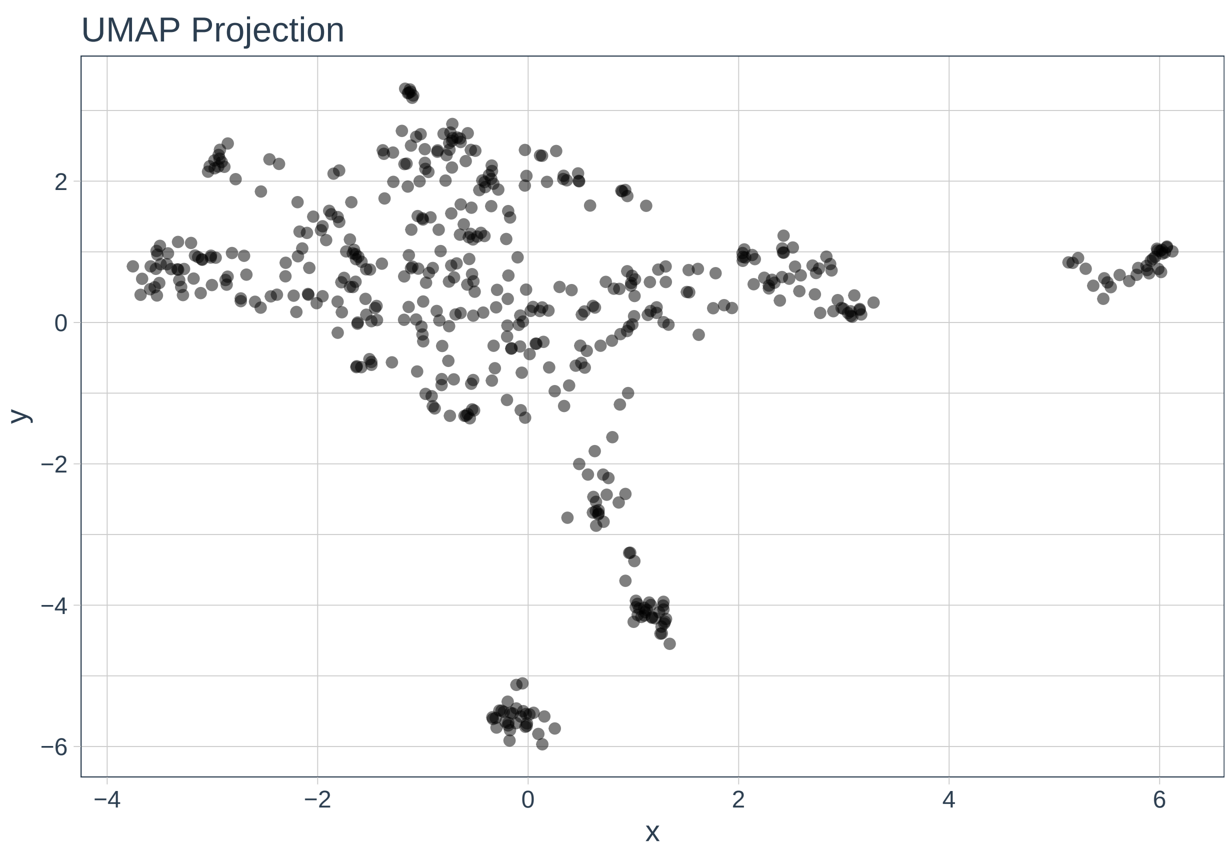
```
##             x          y symbol
## 1  -1.9684862  1.3058786      A
## 2   0.8943466  1.8559051    AAL
## 3  -0.5407814 -0.8665694    AAP
## 4  -3.0159053  0.9196175   AAPL
## 5   0.1107305  0.1630438   ABBV
## 6   0.5571554 -0.4015356    ABC
## 7  -3.3255021  0.7480565   ABMD
## 8  -1.7284814  1.0057644    ABT
## 9  -1.7746957  0.5694575    ACN
## 10 -3.5050678  0.5591511   ADBE
## 11 -2.8862855  2.2029960    ADI
## 12  0.2526680 -0.9715817    ADM
## 13 -2.2020776  0.1489927    ADP
## 14 -0.5331798  0.6840772    ADS
## 15 -3.5488257  0.4987073   ADSK
## 16  1.0228947 -4.0284488    AEE
```

```
## 17  1.2449314 -4.0998656    AEP
## 18  1.1762783 -4.1674484    AES
## 19  0.9789272  0.5597251    AFL
## 20  1.5079183  0.4318009    AGN
```

Finally, let's make a quick visualization of the `umap_results_tbl`.

- Pipe the `umap_results_tbl` into `ggplot()` mapping the `V1` and `V2` columns to x-axis and y-axis
- Add a `geom_point()` geometry with an `alpha = 0.5`
- Apply `theme_tq()` and add a title "UMAP Projection"

```r
# Visualize UMAP results
umap_results_tbl %>%
    ggplot(aes(x, y)) +
    geom_point(alpha = 0.5) +
    theme_tq() +
    labs(title = "UMAP Projection")
```



We can now see that we have some clusters. However, we still need to combine the K-Means clusters and the UMAP 2D representation.

Step 6 - Combine K-Means and UMAP

Next, we combine the K-Means clusters and the UMAP 2D representation

We're going to import the correct results first (just in case you were not able to complete the last step).

```r
# k_means_mapped_tbl <- read_rds("challenge/week_6_data/k_means_mapped_tbl.rds")
# umap_results_tbl   <- read_rds("challenge/week_6_data/umap_results_tbl.rds")
```

First, pull out the K-Means for 10 Centers. Use this since beyond this value the Scree Plot flattens.

- Begin with the `k_means_mapped_tbl`
- Filter to `centers == 10`
- Pull the `k_means` column
- Pluck the first element
- Store this as `k_means_obj`

```r
# Get the k_means_obj from the 10th center
k_means_obj <- k_means_mapped_tbl %>%
    filter(centers == 10) %>%
    pull(k_means) %>%  pluck(1)
# Store as k_means_obj
```

Next, we'll combine the clusters from the `k_means_obj` with the `umap_results_tbl`.

- Begin with the `k_means_obj`
- Augment the `k_means_obj` with the `stock_date_matrix_tbl` to get the clusters added to the end of the tibble
- Select just the `symbol` and `.cluster` columns
- Left join the result with the `umap_results_tbl` by the `symbol` column
- Left join the result with the result of `sp_500_index_tbl %>% select(symbol, company, sector)` by the `symbol` column.
- Store the output as `umap_kmeans_results_tbl`

```r
# Use your dplyr & broom skills to combine the k_means_obj with the umap_results_tbl
umap_kmeans_results_tbl <- k_means_obj %>% broom::augment(stock_date_matrix_tbl) %>%
    select(symbol, .cluster) %>%
    left_join(umap_results_tbl, by = "symbol") %>%
    left_join(sp_500_index_tbl %>% select(symbol, company, sector), by = "symbol")

# Output: umap_kmeans_results_tbl
umap_kmeans_results_tbl %>% head(20)
```

```
## # A tibble: 20 x 6
##    symbol .cluster      x      y company                              sector
##    <chr>  <fct>     <dbl>  <dbl> <chr>                                <chr>
##  1 A      9        -1.97   1.31  Agilent Technologies Inc.            Health Ca~
##  2 AAL    5         0.894  1.86  American Airlines Group Inc.         Industria~
##  3 AAP    9        -0.541 -0.867 Advance Auto Parts Inc.              Consumer ~
##  4 AAPL   8        -3.02   0.920 Apple Inc.                           Informati~
##  5 ABBV   9         0.111  0.163 AbbVie Inc.                          Health Ca~
##  6 ABC    6         0.557 -0.402 AmerisourceBergen Corporation        Health Ca~
##  7 ABMD   8        -3.33   0.748 ABIOMED Inc.                         Health Ca~
##  8 ABT    9        -1.73   1.01  Abbott Laboratories                  Health Ca~
##  9 ACN    9        -1.77   0.569 Accenture Plc Class A                Informati~
## 10 ADBE   8        -3.51   0.559 Adobe Inc.                           Informati~
## 11 ADI    1        -2.89   2.20  Analog Devices Inc.                  Informati~
```

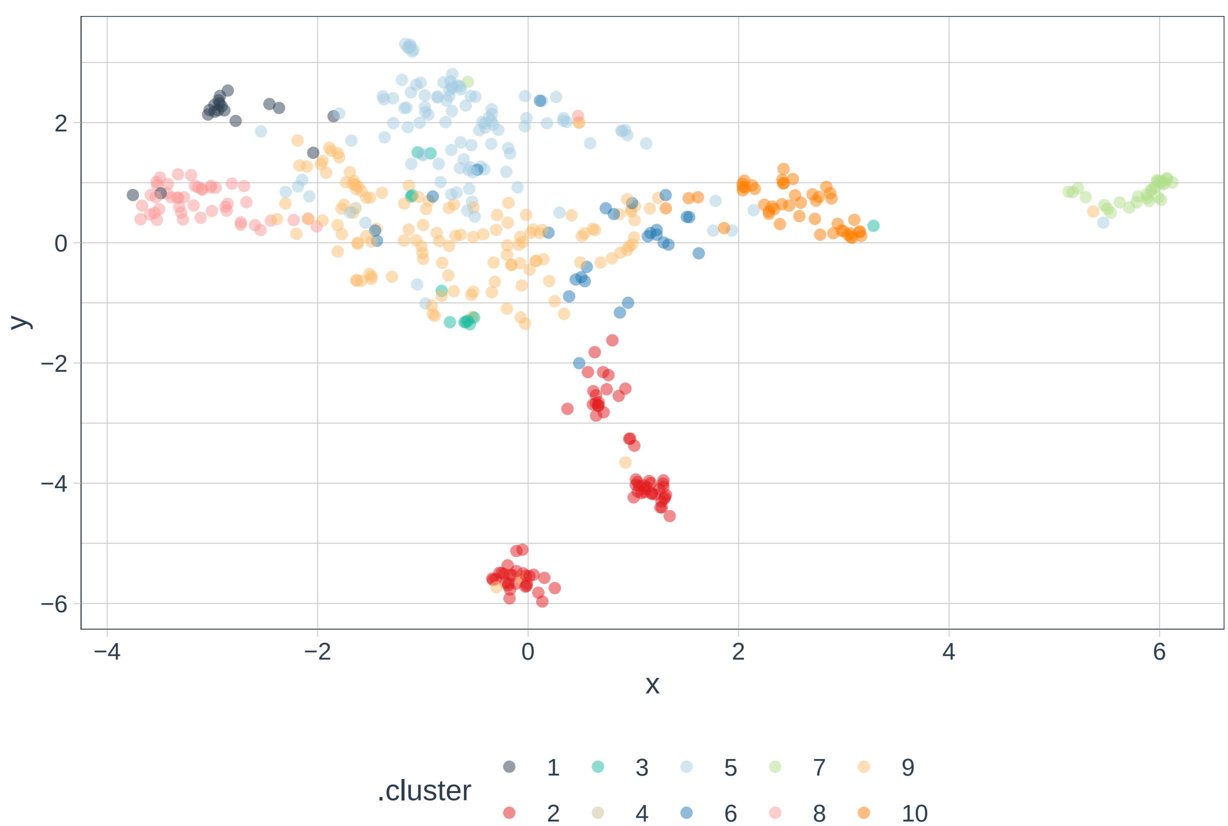
```
## 12 ADM    9          0.253 -0.972 Archer-Daniels-Midland Company       Consumer ~
## 13 ADP    9         -2.20   0.149 Automatic Data Processing Inc.       Informati~
## 14 ADS    5         -0.533  0.684 Alliance Data Systems Corporation    Informati~
## 15 ADSK   8         -3.55   0.499 Autodesk Inc.                        Informati~
## 16 AEE    2          1.02  -4.03  Ameren Corporation                   Utilities
## 17 AEP    2          1.24  -4.10  American Electric Power Company Inc. Utilities
## 18 AES    2          1.18  -4.17  AES Corporation                      Utilities
## 19 AFL    9          0.979  0.560 Aflac Incorporated                   Financials
## 20 AGN    6          1.51   0.432 Allergan plc                         Health Ca~
```

Plot the K-Means and UMAP results.

- Begin with the `umap_kmeans_results_tbl`
- Use `ggplot()` mapping `V1`, `V2` and `color = .cluster`
- Add the `geom_point()` geometry with `alpha = 0.5`
- Apply `theme_tq()` and `scale_color_tq()`

Note - If you've used centers greater than 12, you will need to use a hack to enable `scale_color_tq()` to work. Just replace with: `scale_color_manual(values = palette_light() %>% rep(3))`

```r
# Visualize the combined K-Means and UMAP results
umap_kmeans_results_tbl %>%
    ggplot(aes(x, y, colour = .cluster)) +
    geom_point(alpha = 0.5) +
    theme_tq() +
    scale_colour_tq()
```



If you've made it this far, you're doing GREAT!!!

BONUS - Interactively Exploring Clusters

This is an interactive demo that is an extension of what we've learned so far. You are not required to produce any code in this section. However, it presents an interesting case to see how we can explore the clusters using the `plotly` library with the `ggplotly()` function.

These two functions combine to produce the interactive plot:

- `get_kmeans()`: Returns a data frame of UMAP and K-Means result for a value of `k`
- `plot_cluster`: Returns an interactive `plotly` plot enabling exploration of the cluster and UMAP results. The only additional code you have not seen so far is the `ggplotly()` function. This is a topic for **Week 7: Communication**.

```
get_kmeans <- function(k = 3) {

    k_means_obj <- k_means_mapped_tbl %>%
        filter(centers == k) %>%
        pull(k_means) %>%
        pluck(1)

    umap_kmeans_results_tbl <- k_means_obj %>%
        augment(stock_date_matrix_tbl) %>%
        select(symbol, .cluster) %>%
        left_join(umap_results_tbl, by = "symbol") %>%
        left_join(sp_500_index_tbl %>% select(symbol, company, sector),
                  by = "symbol")

    return(umap_kmeans_results_tbl)
}

plot_cluster <- function(k = 3, interactive = TRUE) {

    g <- get_kmeans(k) %>%

        mutate(label_text = str_glue("Stock: {symbol}
                                      Company: {company}
                                      Sector: {sector}")) %>%

        ggplot(aes(x, y, color = .cluster, text = label_text)) +
        geom_point(alpha = 0.5) +
        theme_tq() +
        scale_color_tq()

        if (interactive) {
        return(ggplotly(g, tooltip = "text"))
    } else {
        return(g)
    }

}
```

We can plot the clusters interactively.

```r
plot_cluster(k = 10, interactive = interactive)
```