**Data**

***All Data used:***

| **Description** | **Used for (topic area)** | **Source (link if available)** | **New data for final DMP (Yes/No)** | **# Observations** | **# Features** |
| --- | --- | --- | --- | --- | --- |
| Table\_5\_1.csv | association | <https://drive.google.com/file/d/1carAG2NKhG6DUqNob321BQUyItHHTU8p/view?usp=sharing> | No | 5 | N/A |
| BayesAssign1\_01.csv | classification | <https://drive.google.com/file/d/1uA1cWrl0YA9l0xprbivkG9zjIV_xqreV/view?usp=sharing> | No | 1000 | 2 |
| DecTreeAssign1.csv | classification | <https://drive.google.com/file/d/1wKIb1oWnsv1w5TzJ66ra4TMed0y-hmbL/view?usp=sharing> | No | 569 | 30 |
| Country-data.csv | clustering | <https://drive.google.com/file/d/1_eGI0AopXC0eLugnieuPSWsD7jK69TOR/view?usp=sharing> | No | 167 | 10 |
| Live.csv | clustering | <https://drive.google.com/file/d/1I3UPZn6AveUw6Oa6Fp-a2GvqVyvbfT8R/view?usp=sharing> | Yes | 7050 | 11 |
| segmented\_customer.csv | clustering | <https://drive.google.com/file/d/1XWvl1JquAS2HrcRnEI7wsgk_nsr1owIa/view?usp=sharing> | Yes | 201 | 5 |
| Randomly Simulated Data For Demonstration | anomaly | N/A | Yes | Can be Adjusted in the code | Can be Adjusted in the code |
| active\_players.csv | data | <https://drive.google.com/file/d/1z-49VhjYDL4kQsX_Pf17VjcQfPcKiO_S/view?usp=sharing> | No | 558 | 8 |

***New Data overview :***

Three new datasets are added to the final portfolio including Live.csv, segmented\_customer.csv, and randomly generated data (for simulation of isolation forest). Let’s go over them one by one.

*Live.csv*

| **Attribute** | **Type** | Description |
| --- | --- | --- |
| status\_id | key / id | Identification for each observation |
| status\_type | categorical | Type of live streaming |
| num\_reactions | numerical | Number of reactions in the live stream |
| num\_comments | numerical | Number of comments in the live stream |
| num\_shares | numerical | Frequency of sharing the link of the live stream |
| num\_likes | numerical | Number of likes (double clicks) in the live stream |
| num\_loves | numerical | Number of loves emoji sent in the live stream |
| num\_wows | numerical | Number of wows emoji in the live stream |
| num\_hahas | numerical | Number of hahas emoji in the live stream |
| num\_sads | numerical | Number of sads emoji in the live stream |
| num\_angrys | numerical | Number of angry emoji in the live stream |

Nowadays, smartphones, tablets, computers have increasingly dominated the lifestyle of modern human beings. This provided the live streamers a better opportunity to attract potential viewers, make money, and make themselves famous. More importantly, as we have to fight the fierce Covid-19 pandemic in the long run, most people chose to stay indoors which made the live streaming industry has now become relatively steady.

The data\_set we are focusing on today is called Live.csv which includes 7050 observations and 11 features. It mainly recorded the features of a live stream. For example, it included the streaming type attribute which can be a photo (sound) or video-based streaming. Most features are numerical, so it is relatively easy to plot using scatter plots and it is possible to normalize the data.

*segmented\_customers.csv*

| **Attribute** | **Type** | Description |
| --- | --- | --- |
| Customer\_ID | key / id | Identification for each observation |
| Gender | categorical | Gender of the customer |
| Age | numerical | Age of the customer |
| Annual Income | numerical | The annual income of the Customer |
| Spending Score | numerical | Spending Score of the Customer |

As more companies are flooding into each industry, the quality of certain items has come to saturation. Thus, market analysts and business analysts increased their focus on the customer side. They commonly observe customers’ activities and expenditures, which provided them with useful information for strategic planning of the company and their production line.

This dataset is a normal customer information data set which shifted its weight to focus on the features related to purchasing power. For example, the annual income and spending score attributes are two attributes closely related to the purchasing power of an individual. The data contains 201 observations and 5 features. Most features are numerical and the number of observations is small which would be a limitation for statistical approach analysis.

*Randomly generated data (for simulation purposes)*

The data for the isolation forest algorithm implemented is a simulation of an MxN 2D NumPy array, in which its size could be adjusted within the code. Data with huge capacity is not used in this scenario since we want to shift our focus more on the algorithm itself. Moreover, we also want to keep the dataset simple to get a better visualization of the trees constructed within the isolation forest.

**Code Development**

***Link to code:***

| **Process (make a link to code if appropriate)** | **Used for (topic area)** | **Dataset(s) used** | **New code for final DMP (Yes/No)** | **Additions made** | **Link to code (please open with google collab)** |
| --- | --- | --- | --- | --- | --- |
| Apriori Algorithm | Association Analysis | Table\_5\_1.csv | Yes | Rule Generation code implementation by hand (with only permutation and combination packages) | <https://drive.google.com/file/d/1dn8MvUw8sNLqH81rWSn5W3vuXzjt2RBI/view?usp=sharing> |
| Isolation Forest | anomaly | Randomly Simulated Data For Demonstration | Yes | Isolation Forest algorithm implementation | <https://drive.google.com/file/d/1tXAx2CbZ8PIkAhogPAPQt_gyDZdKhbiU/view?usp=sharing> |
| Bayes | Classification | BayesAssign1\_01.csv | No |  | <https://drive.google.com/file/d/1F8EKNyYW3fbB0V10XMddobqQuR7VUmSQ/view?usp=sharing> |
| Decision Tree Algorithm | Classification | DecTreeAssign1.csv | No |  | <https://drive.google.com/file/d/1lf2DmaPNclRVhoqof391ZtyeTVm88zQV/view?usp=sharing> |
| k-means | clustering | Country-data.csv | No |  | <https://drive.google.com/file/d/14nUWIhVrO4y4dQ8-joDfjZ-NDzJxH_kP/view?usp=sharing> |
| similarity / proximity | data | active\_players.csv | No |  | [0gNMydI6ku5ALdHWbYyLQ7mPgmSBZY](https://drive.google.com/file/d/16-0gNMydI6ku5ALdHWbYyLQ7mPgmSBZY/view?usp=sharing) |

In the Final DMP, I implemented two algorithms by hand, one is the rule generation procedure in the association analysis and the isolation forest algorithm. Let’s go over them one by one.

Apriori Algorithm (Rule Generation without package)

I implemented the rule generation procedure in the association analysis section. Although it is easy to generate the rules with the package, I still decided to do it by hand since I believe that this is an important procedure that is associated with confidence and support. Moreover, it is an important procedure if I encounter jobs involving market basket analysis. The basic logic behind this implementation was relatively simple, however, some unexpected difficulties occurred throughout the process.

***Data Structures and Logic***

Throughout the coding process, I mainly used lists and dictionaries as my data structure for saving observations and candidate\_rules. For simplicity, I implemented several helper functions that are embedded at the end of the code. For example, get\_sub\_lists obtain all the sublists of a list. Another one can be difference\_between two lists, which returns the complement of a sublist that together forms the full list.

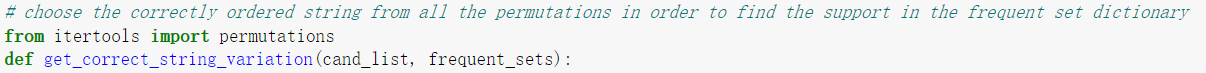




Talking about functions, the two main functions that I implemented was the get\_confidence function and rule\_generation. The get\_confidence function inputs the candidate association rule, the frequent sets for all k, the key\_list which is the keys of the frequent sets dictionary, and “i” which is the length of the candidate association rule. The function finally outputs the confidence of the candidate rule.

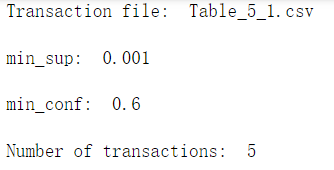
The second function is the main function of the rule\_generation algorithm. Basically, for all i frequent sets, we get all sublists of the frequent set and associate it with its complement towards the full set. Direction matters, so association rules in two ways are created. For example, “bread → milk” and “milk → bread” are both added in the candidate rules if {bread, milk} is a frequent set in i = 2-frequent sets.

Although the logic is simple, problems still arise when I want to locate candidate rules in the frequent set dictionary. For example, when I generate candidate association rules, “milk → bread” and “bread → milk” both occur in the candidate rule, however when I tried to prune the candidate rule, I cannot find the combined support for the latter one. This is because the representation string became “bread, milk” after I merge them together. However, the frequent set in the data is “milk, bread”, which does not match with the string. Therefore, after some quick research on the internet, I decided to implement the permutations algorithm to generate all possible permutations of a data set which allows me to check each variation with the dictionary, and then determine the support of the frequent set.

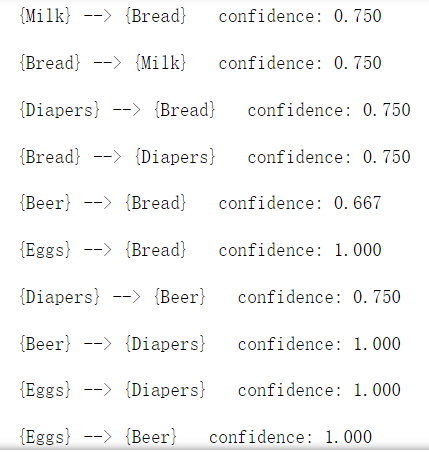


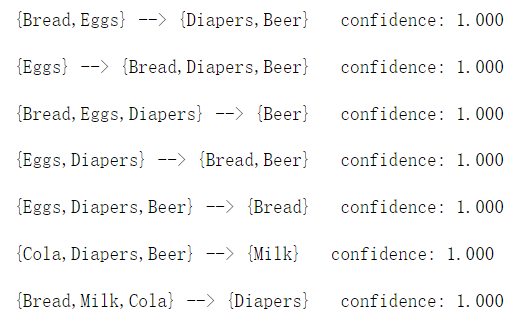
***Analysis and learning***

The output of the code is shown for hyperparameters shown below.



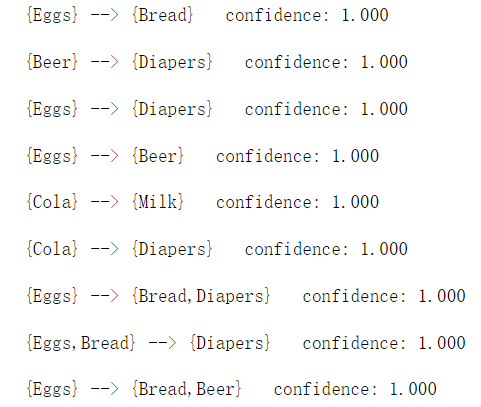
Output (Rule Generation portion)





The figure above shows the partial output of the association rules with minimum support of 0.01 and minimum confidence of 0.6, which is a common combination for market basket analysis. According to the figure, the rules generated all have confidence higher than 0.6. There are about 40 association rules when the MIN\_CONF = 0.6. In contrast, when the MIN\_CONF = 0.8, only 22 rules survived. This observation confirms the pruning logic of the rule generation procedure by which an increase in the MIN\_CONF threshold decreases the number of rules generated.

Part of the output (MIN\_CONF) = 0.8



Isolation Forest (algorithm implemented without package)

I implemented the isolation forest algorithm without any external package in this task. The main focus is to understand the basic logic and behind the isolation forest, especially how each tree is constructed and what is the purpose of implementing trees in that certain way. Moreover, a comparison between the decision tree and the random forest is added in the end.

***Data Structures and Logic***

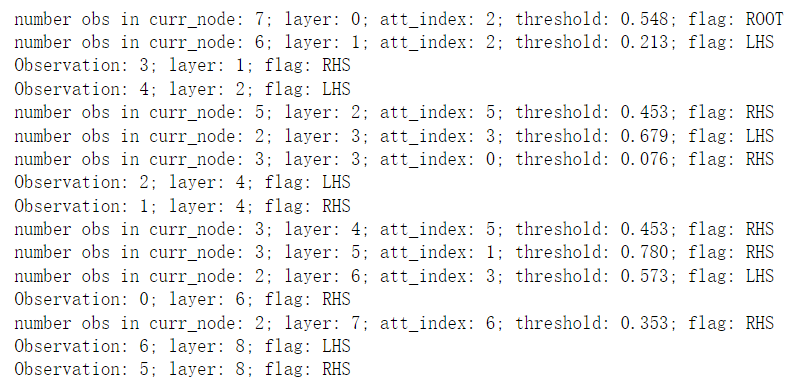




Isolation Forest is basically an algorithm that generates multiple decision trees and focuses on the low layered leaf nodes, which assists researchers to find anomalies (outliers, influential points, etc.). In the algorithm, I basically implemented Two classes: The “Random Tree” class, which indicates the instance of a node within a tree that is randomly generated. The “Isolation\_Forest” class, which indicates the instance of a tree generated by the logic of isolation forest.



The main function within the algorithm is the “\_\_get\_isolation\_forest” function, which allows us to build the tree with the isolation forest’s logic. The logic behind the algorithm is simple but clever. For each iteration, we want to generate a binary tree that split each node on random splitting criteria (random attribute for splitting, random cut point for splitting). We want to do everything randomly because we want to save time when building the decision trees. Instead of exhaustively searching for the best feature cut-point pair, we want to increase the efficiency of the algorithm. Moreover, randomness is essential for us to generate different trees for the data. This would help us in finding anomalies.



This is the output of a single tree. Note that if it is the root node, we display: the number of total observations, the layer that the current node is located, the index of the attribute used for the root node split, the threshold level, the flag variable set to ROOT. If it is an internal node, we display: the number of observations split to the node, the layer that the current node is located, the index of the attribute used for the node split, the threshold level, the flag variable. If it is a leaf node, we display the index of the attribute, the layer that the current node is located, the flag variable. We only want the index of the attribute when we reach a typical leaf since the most important feature of a leaf is its layer or its distance towards the root node. Since for anomalies, their distance towards the root node is relevantly short on average.

Before we go through the n iterations of the n binary tree generation, we set a list named “counter” which has a size of the number of observations. Its main function is to increment on a certain index if a certain observation ends up in the tree node that has layer <= max\_layer (a hyperparameter set at the beginning).



After generating the n binary decision trees according to the isolation forest algorithm, we also updated the counter with the most frequent observation that ends up in leaf nodes with a low distance towards the root. We use the max() function to sort out the observation.



In our case, the observation index is 3, and the number of times it is detected as a potential anomaly is 7. (With max\_layers = 2, tree\_num = 20). To confirm that the algorithm is performing in a correct way, we tuned the hyperparameters. This time, we changed the number of trees to be 30 while keeping everything else the same.



We found that the number of observations remains the same, just with the added frequency of being detected as an anomaly. When we change the number of iterations to 50, the same trend concurred.



When we keep the number of trees the same, and tune the max\_layer threshold, we still found the same result but with higher frequencies of being detected as an anomaly. This corresponds to the logic, in which an increase in the max\_layer threshold increases the standard, thus the observation will be determined as an anomaly in a more frequent manner.



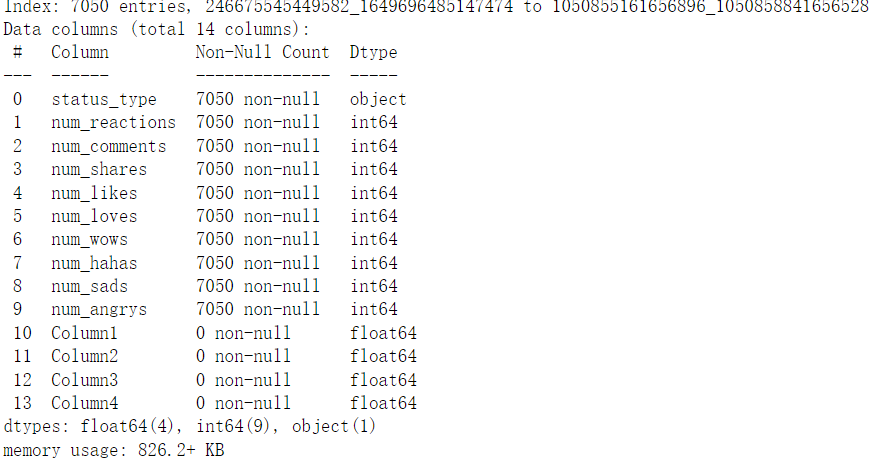
**Package Use**

In the package use section, I added two new materials into my final DMP, which includes the k-means++ algorithm with elbow method graph and hierarchical clustering. The following table indicates the new information added to the package use section.

| **Package Use** |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Process (make a link to code if appropriate)** | **Used for** | **Dataset(s)** | **New Package for final DMP (Yes/No)** | **Source** | **Link to code (please open with google collab)** |
| k-means ++ & elbow method graph | clustering | Live.csv | Yes | sklearn | <https://drive.google.com/file/d/1GJFzsxoaIJTrMxaDB2mckmYWMqXNiZYD/view?usp=sharing> |
| hierarchical clustering | clustering | segmented\_customer.csv | Yes | sklearn | <https://drive.google.com/file/d/1vohc2skggLnP4Hk2ZrFcynhjX3cFFHQv/view?usp=sharing> |
| Decision Tree Visualization | Classification | DecTreeAssign1.csv | No | sklearn | <https://drive.google.com/file/d/10ozAifs1MR-TvaxaSYFzyq9b0AHZUs2O/view?usp=sharing> |

k-means ++ & elbow method graph

K-means++ is a traditional method for initial centroid selection for the k-means clustering algorithm. Instead of randomly selecting the initial centroid, k-means++ select centroids based on the probability of distance towards the nearest center that has already been selected (P(D(X)^2)). This algorithm will be slower than random centroid selection, but will relatively perform better in terms of SSE.

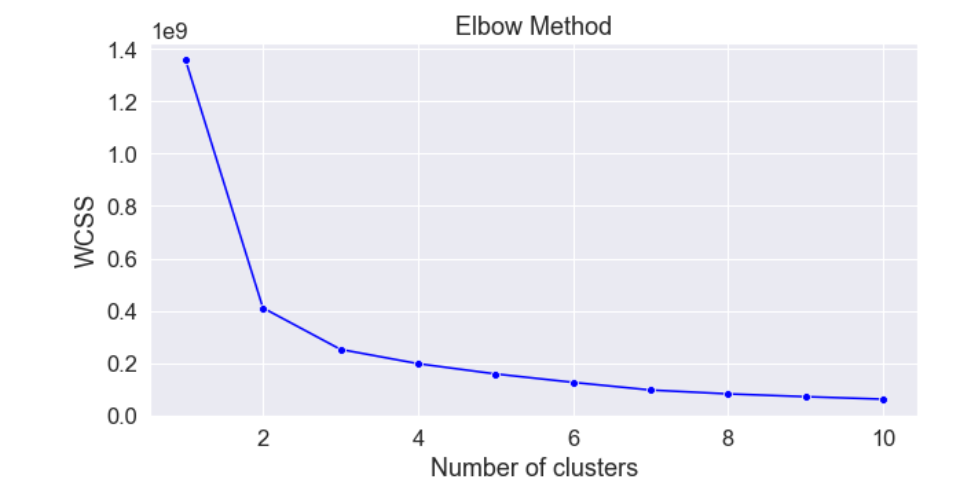


For our data which is Live.csv (described at the beginning of the data section), we use the describe() function to see the summary of the data and then check whether null or duplicate values occurred in the data. If there are occurrences of such data, we drop them in the data cleansing process. Note that in both the elbow graph construction section and the k-means++ cluster plotting section we selected two main features including “num\_comments” and “num\_shares” for graph construction purposes. (Later we will analyze with different pairs of elements)

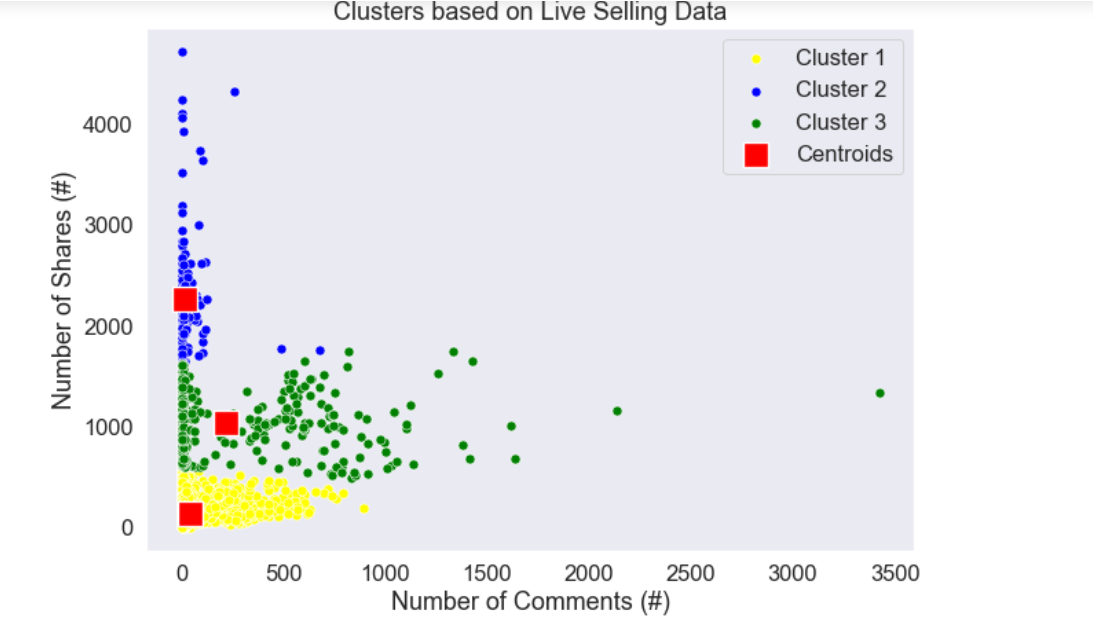


***Implementation and Analysis***

In our scenario, we applied k-means++ from the sklearn package and plotted “WCSS” (Within-Cluster Sum of Square) against “the number of clusters”. The graph output is similar to the SSE against the num\_clusters graph that was mentioned in class.



The increase in the number of clusters leads to a lower WCSS measure as expected. Instead of just randomly selecting the number of clusters, the Elbow Method graph indicates that the acceleration of the drop of WCSS decreases significantly at k = 3, thus we decided to select k = 3 for our k-means++ clustering. Further increase in the number of clusters is desired since it provides better WCSS or SSE, however, it is not proportional to the usage of time and space complexity. Moreover, considering the clarity for interpretation, k = 3 is the best for our scenario.

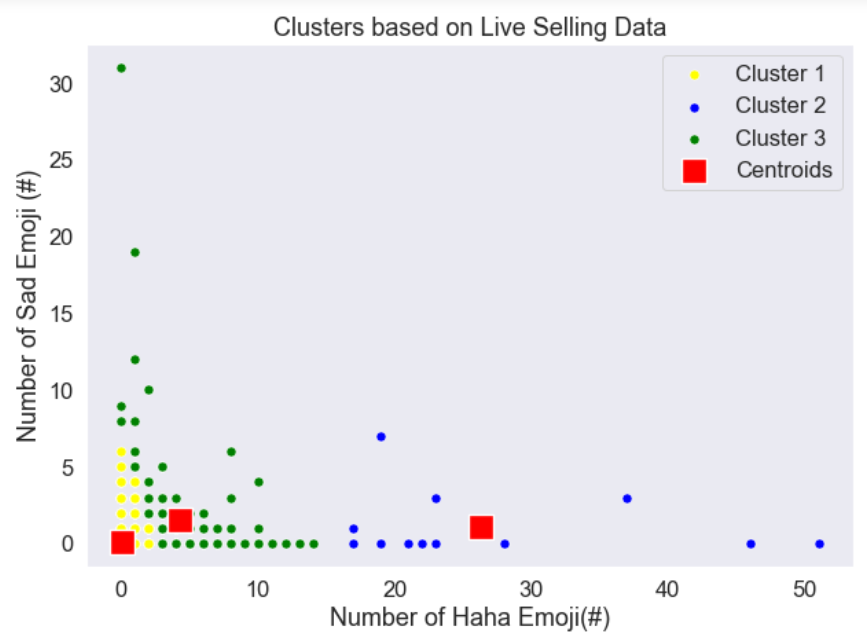


Next, we used k-means++ to fit the two attributes. From the results above, we may clearly see that the observations are being clustered based on the number of shares and the Number of Comments. Three clusters are formed after the k-means++ clustering procedure. As we may see, the number of shares became the dominant feature in the clustering procedure by which the clusters are mainly split based on the number of shares. The yellow cluster are observations that have a low number of shares in contrast with the green and blue clusters that have a relatively higher number of shares. This yellow sample of observations represents the live streams that have a relatively low number of shares and a medium number of comments. To predict, this cluster of data are the live streams that have low-quality streaming and poor content. The reason behind the higher number of comments compared to the blue cluster is that there are a great number of **keyboard warriors** who may have an unhappy day and would like to post disliking comments to vent their frustrations on the live streamer.

The second cluster (in green) is the live streams that have a medium number of shares and a relatively high and flexible number of comments. The major reason behind the high variance of this cluster is due to the mixing of the good and bad live streamers. The good live streamers are the ones that are in great shape and continuing to increase fan base at a steady pace. The bad ones are streamers with poor content, however, rely on the high number of comments created by **water armies** that increase their popularity and number of shares. However, due to the low quality of streaming content, the number of shares is still low relative to the blue cluster. The outlier that has a number of comments = 3500 indicates the existence of water armies that create fake comments just for popularity as no other observations reach such a high number of comments.

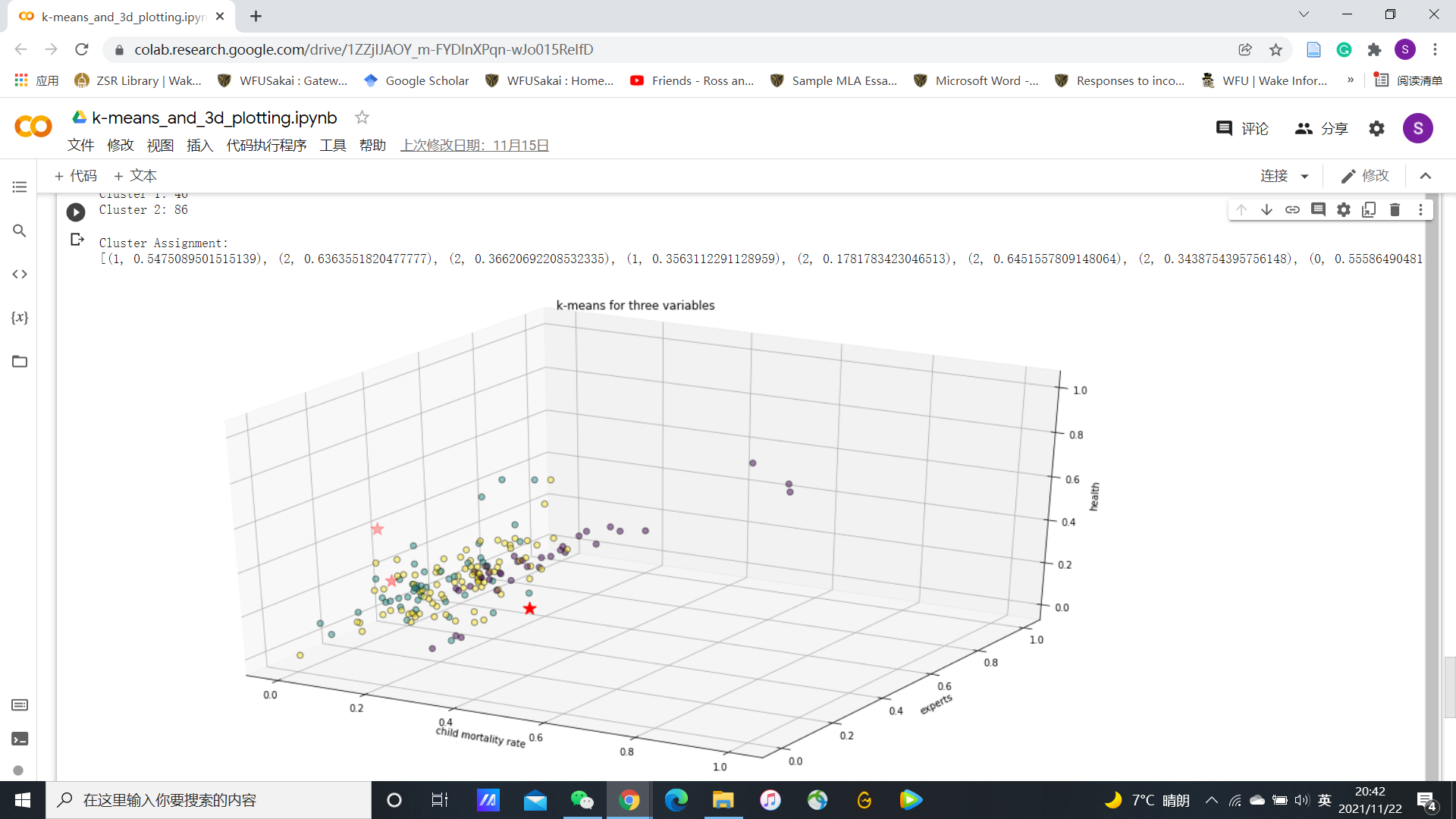
The blue cluster, contrary to the other two, has a relatively high number of shares while a low number of comments. These observations are the elite proportion among the live streams. They represent popular live streamers and streams that are broadcasting breaking news.

For another example of k-means++ clustering, we chose the 7th and 8th attributes in the data set: num\_hahas (number of hahas emoji that occurred in the live stream) and num\_sads (number of sads emoji that occurred in the live stream). These are two contradicting emojis, from the clustering graph created, we can have great confidence to predict the quality of the live stream content.



In this scenario, we still chose k = 3 for a clearer interpretation. The yellow cluster is the cluster that contains live streamers with a small fan base (started just now), thus having a low number of both Sad and Haha emojis. The green cluster contains the live streamers that have a medium number of Haha emojis and a relatively variable number of sad emojis. The anomaly that occurred at the top left side of the graph indicates an extremely high sad emoji observation. This observation might have resulted from breaking news that transmits extremely negatively feeling content, such as disasters or pandemics (such as covid 19 break out). The blue cluster has a relatively high number of haha emoji while a low number of sad emojis. These observations are mainly providing high-quality content that is possibly dramatic.

In conclusion, k-means++ are providing interpretable and meaningful graphs. Most of the clustering graphs produced provided useful content for analysis. In comparison, k-means clustering done in the clustering DMP relatively have low accuracy and a low degree of separation.



K-means clustering visualization (dataset = ‘Country.csv’)

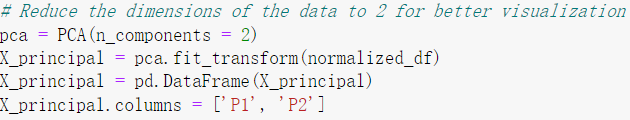
As we may see in the 3D cluster plotting regarding the data set ‘Country.csv’ and variables: child mortality rate, experts, health, the data points are mixed together for different clusters, which signals a bad performance of clustering.

Link to code: <https://drive.google.com/file/d/1ZZjIJAOY_m-FYDInXPqn-wJo015ReIfD/view?usp=sharing>

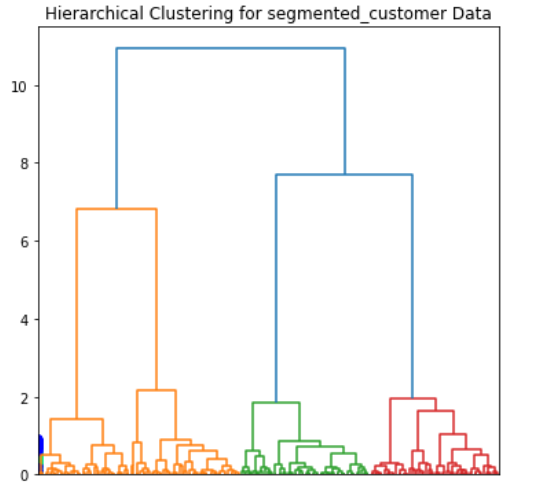
Hierarchical Clustering

For the second package use, I chose to use the hierarchical clustering package from sklearn. The data I used for this part is the ‘segmented\_ customers.csv’ as described in the data sections. We first drop the first column which is the ‘id’ of the customers since it is irrelevant to the analysis of the dataset. Then, I cleaned the data by filling the ‘None’ values, normalized, and scaled the data.

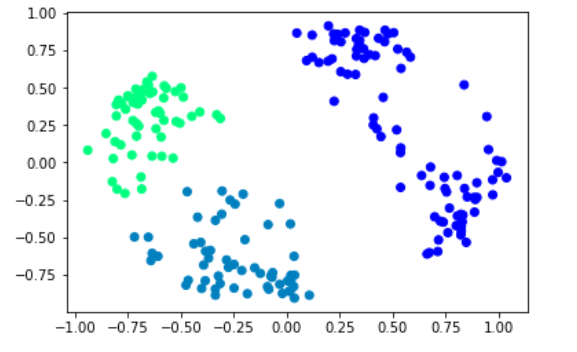
***Implementation and Analysis***



After that, I used the first two columns for the clustering procedure, which are Age against Final Income. We do not have to worry about the difference in scale between age and final income since we normalized the data in the data cleaning procedure. We then fit the observations into the dendrogram for hierarchical clustering visualization.

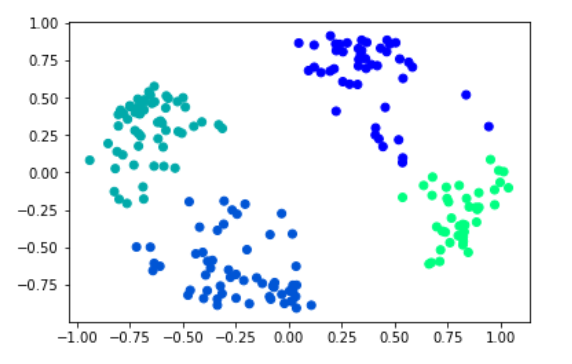


From the figure above, we may see that multiple layered dendrograms are formed. However, we may only focus on the colors of the diagram. As we may see, the dendrogram divided the observations into 3 clusters: orange, green, and red. The orange cluster indicates the low aged group with median income. The green cluster is the medium-aged group with relatively low income. The red cluster is the high-aged group with relatively low income. The green and red clusters are then clustered together to form the meta-cluster and then finally join with the orange cluster to form the whole data set. The hierarchical clustering procedure generates clusters bottom up. It first separates the data set into k clusters, then reduces to k-1, and finally 1 cluster. This approach is perfect for viewing clustering with a different number of centroids at the same time. However, the data separation is not that obvious and we cannot really receive important info regarding the attributes side.



The above graph is the hierarchical clustering procedure in the form of a scatter plot. From the plot, we can see that the data set is divided into three clusters, colored in dark blue, grey-blue, and green. The green data indicates a low-aged group with medium income. This is the white-collar group in the customer database with relatively medium purchasing power. The grey-blue cluster is middle-aged customers with relatively decreasing annual income. This trend corresponds to the job losses in the middle-aged group. The third cluster in dark blue is the gold-collar managers with high annual income. Together with the retired elderly who have a minimum income. From this observation, we market analysts can easily make suggestions to companies to target specific products to a certain customer base.

One observation here is that the third cluster in dark blue contains both high and low-income elderly groups. Would it be possible to split this cluster into two to make a better interpretation if k = 4? We may find out by tuning the parameters.



As expected, after the hyperparameter k is changed to 4, the two income groups are separated for the elderly, which makes the interpretation make more sense.

**Exercises**

Please view the Final\_Exercises.pdf in the portfolio

[Final\_Exercises.pdf](https://drive.google.com/file/d/1KDYHcvbhTrglFVaGiG5tDnx1PxqvhPZu/view?usp=sharing)

**Major learning points**

In this class, I mainly learned about how to train and interpret different types of data sets using data mining procedures such as decision trees, association analysis, clustering-based procedures. I also learned how to correctly and more efficiently detect anomalies using methods such as isolation forest. This material will be essential for me throughout my career even if I do not focus on the data science track. Beyond the application of such techniques, I also learned the lower level logic of the algorithms such as the Apriori algorithm and k-means clustering. Moreover, I also completed detailed exercises of theory, which solidified my foundations.

**Self-assessment**

I think I learned a great deal of information in this class and did a decent job in the coding sections. However, since I only completed the basics in some of my DMPs, some extra material involving packages is not completed. In the final DMP, I added additional material regarding previously undone selective materials such as hierarchical clustering and kmeans++. I am now comfortable with implementing algorithms and applying packages for most of the data mining procedures as well as analyzing the models and graphs. Thus I choose to give myself a B+ in the overall assessment.