This report describes the process of implementing the K-Means algorithm over MongoDB using Stack Exchange data. The assignment tasks use Euclidean distance to calculate distances between points. Points are created using the view count and score of posts. Posts are considered with only these sub-set of tags; "command-line", "boot", "networking", "apt", and "drivers".

Note: The program code is provided individually for each question in the root folder. The program can be executed by following the README.md instructions. The questions from the root directory contain method calls from the other files in the "mongo" subfolder to make the code more manageable and readable. The global settings are provided in "globals.py" inside the root folder.

Question 1: For each document, create a field called 'kmeansNorm' as an array in which the first position is the normalized view count and the second is the normalized score.

First, we need to populate the database. Hence, the first step is to run "insert_data.py" from the root folder, which populates the database with users and valid posts.

Then, I use the populated database (e.g., posts collection) to create a subset of the post-collection (kmposts) based on the given tags above. This code can be executed by running "q1.py" from the root folder. Once the new collection with given tags is created, I fetch the view count and score for each document and update it by inserting the 'kmeansNorm' for each document based on the calculations provided (e.g., Let v be the value of a field and m and M the minimum and maximum values among all posts. The normalized value of v is computed as (v – m) / (M – m)). At the end of this step, each document contains the K-Means norm as an array where the first value is the normalized view count and the second is the normalized score.

Question 2: Write a program that selects k random documents (k and t are inputs to your program). Insert the newly created field from **Q1** into new documents in a centroid collection, assigning the centroid document IDs from 1 to K. Erase any previous records in this collection if they already exist.

This program is executed by running "q2.py" from the root folder. The program takes the inputs K (sample count as a number) and T (a valid existing tag, one of the subset tags as a string). The input parameters (K and T) can be provided in "globals.py" in the root folder.

The program first erases the centroid collection before creating a fresh one with a given tag (T) and the number of documents from that tag based on the given input sample count (K). Each document's primary field is assigned using the sample counter (e.g., 1...K).

Question 3: Implement one step of the k-means algorithm by assigning a new field 'cluster' in each document with a selected tag to assign it to the closest centroid. Then, update the centroids collection with new centroids.

This program is executed by running "q3.py" from the root folder. This method first erases the centroids collection and creates it as fresh (based on the given tag and sample size, similar to **Q2**) to run K-Means from scratch. The input parameters (K and T) can be provided in "globals.py" in the root folder.

In the execution, the method takes the posts (a subset based on the tag) and centroids (freshly created based on the tag and sample size) and calculates the distance between K-Means norms of centroids and posts. The distance clusters are assigned where the distance from posts to the closest centroid (assigning post to that cluster). As I know that the subset size of posts is small and easily manageable in memory, I calculate everything in the memory together instead of reading or writing data in chunks.

Question 4: For each of the tags, initialize the cluster centers and run k-means for up to 100 iterations (or until convergence) starting with k=10 up to k=50 with a step of 5 using only posts with that tag. Plot the sum of squared errors vs the value of k for each of the tags (i.e. one plot per tag). Provide your code and the plots.

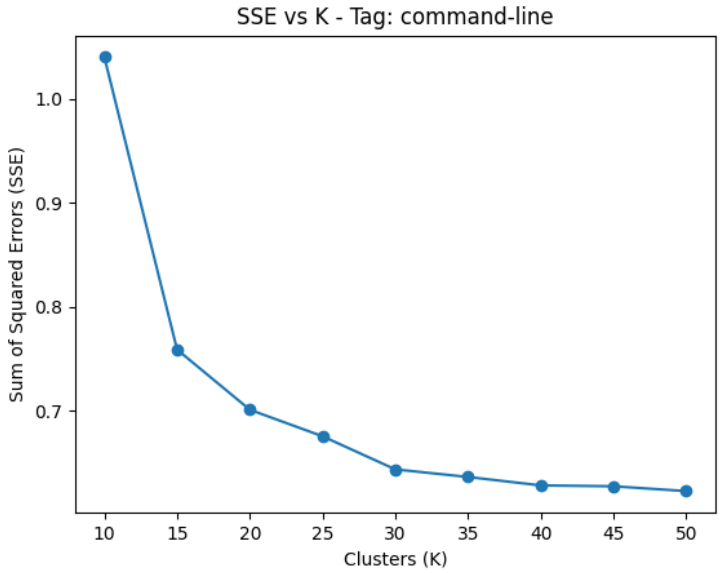
This program is executed by running "q4.py" from the root folder. The code iterates over each subset of tags given above and runs K-Means for each tag. For each tag, the K-Means runs for different values of K (starting from 10, up to 50, with a step size of 5). For each value of K, the convergence value is set for (0.000001). The code executes up to 100 iterations for each value of K unless the convergence condition is met. The execution

to create fresh centroids for each value of K, updating the clusters for posts with selected tags, and updating the centroids for cluster centers following the same strategy as explained in **Q3**.

The execution can be followed on the console, showing the running of each iteration for each value of K for each tag showing the error and convergence condition. Once the condition is met (convergence or maximum iterations). The code creates the plots for the sum of squared errors vs the value of k for each of the tags (i.e. one plot per tag). The plots are saved in the root folder and named with tags (e.g., "sse_plot_boot.PNG" for the boot tag.

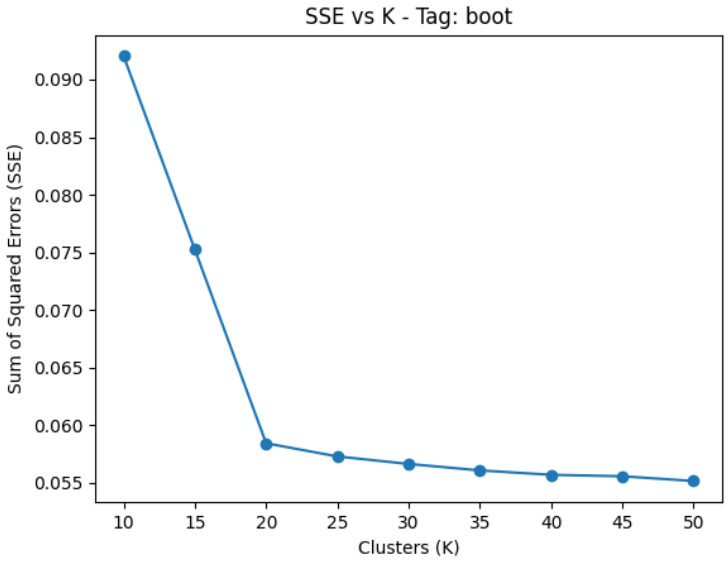
Tag COMMAND-LINE. BEST K = 20

The best value of K is around 20. There is only a small decrease in error after that.



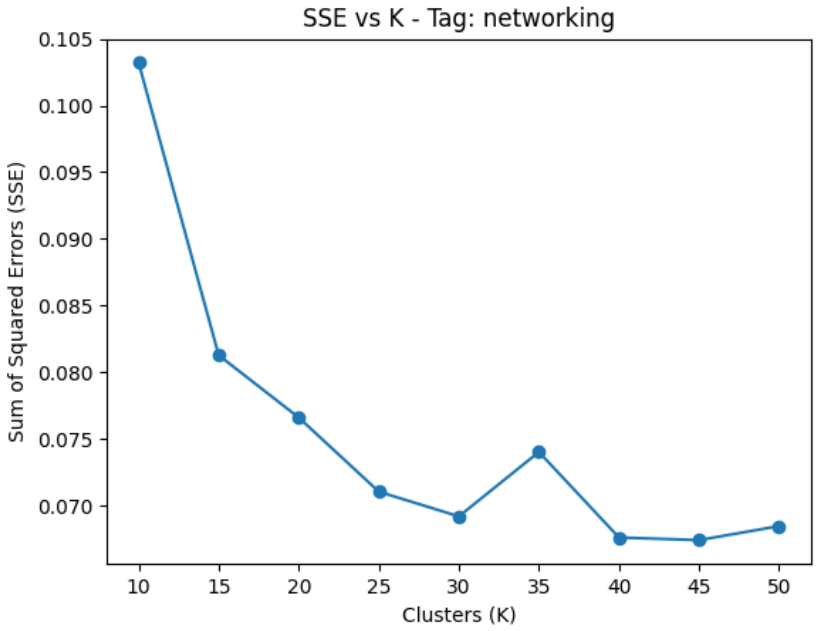
Tag BOOT. BEST K = 20

The best value of K is around 20. There is only a small decrease in error after that.



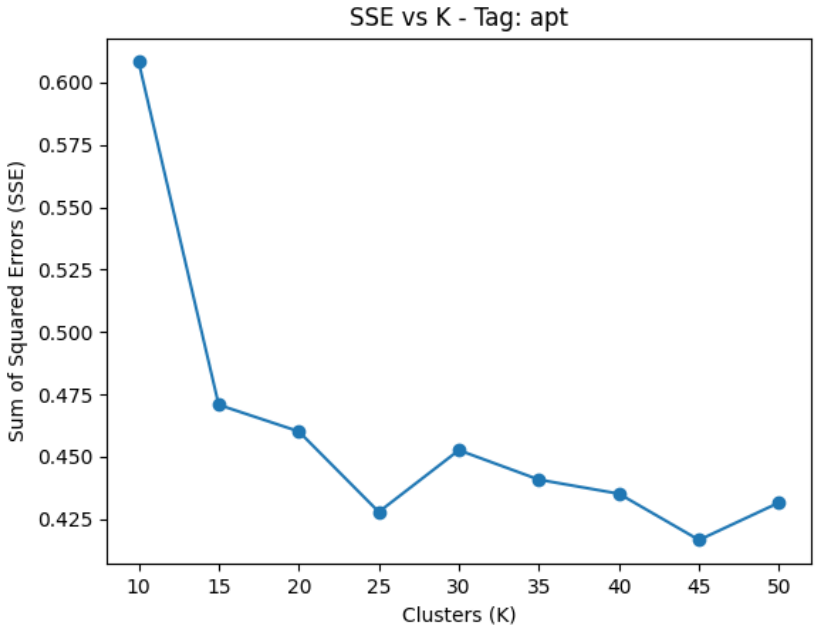
Tag NETWORKING. BEST K = 25

The best value of K is around 25. However, based on the error, 15 can be a choice as well.



Tag APT. BEST K = 15

The best value of K is around 15. There is only a small decrease in error after that.



Tag DRIVERS. BEST K = 15

The best value of K is around 15. There is only a small decrease in error after that.

Question 5: For each of the previous tags, decide which one of the computed number of clusters is the best in each case. For each tag, present an example cluster of posts grouped such that you can explain it.

This program is executed by running "q5.py" from the root folder. I visualize the plots as shown above to identify the optimal number of clusters. This program expects the optimal number of clusters as input provided in "gloabls.py" as the "best_Ks" array following the sequence provided in comment line 14. The sequence must follow the sequence of subset tags used to extract a subset of posts.

The code uses similar operations for creating centroids, updating clusters, and centroids' values as explained in **Q4**. However, the code is run for only a fixed value of K. Other conditions (convergence, iteration) remain the same. At the end of execution, the program takes a random centroid for each collection and creates a subset of the collection with posts belonging to that cluster for the particular tag for further analysis. (e.g., the "posts_boot_cluster1" collection will contain posts from cluster 1 having a boot tag with the best value of K.

The best values for K are; command-line = 20, boot = 20, networking = 25, apt = 30, drivers = 20.

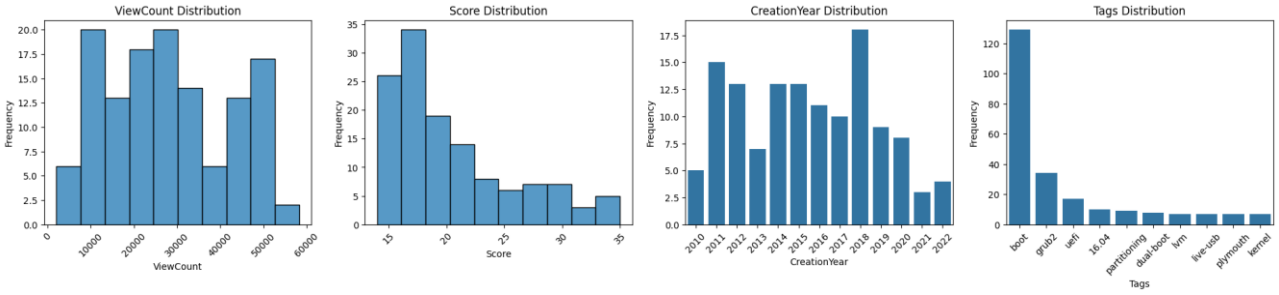
Further analysis is carried out using a jupyter notebook "q5_nb.ipynb". The notebook is provided to re-produce the analysis, however, the results are explained below. For each tag, a random cluster is chosen for analysis. Once the "q5.py" executes, it will produce 5 collections tags. To run the analysis, the database name and collection names are required.

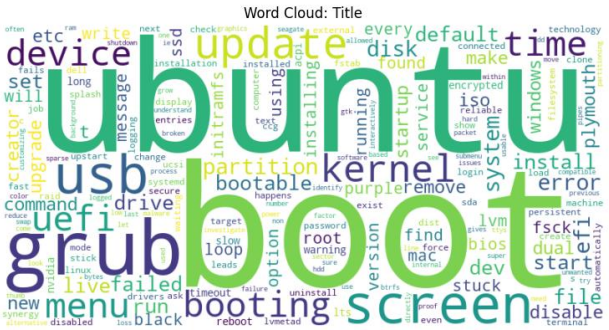
The following section explains the statistics from each chosen random cluster.

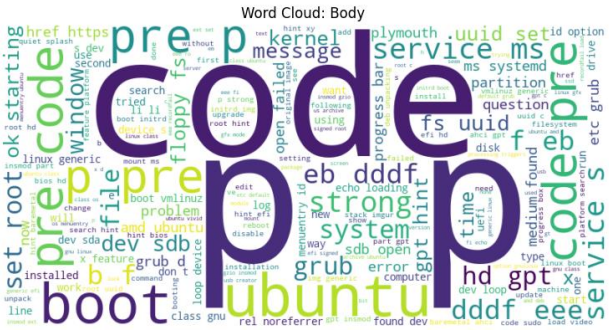
Tag BOOT. BEST K = 20: Random Cluster 11.

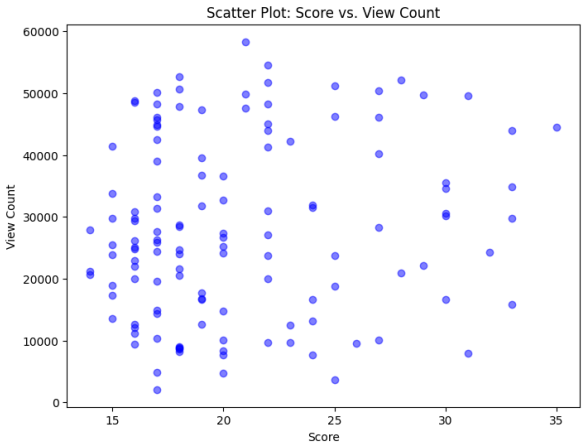
This cluster has 129 posts. Starting from 2010 to 2022. The score ranges from 14 to 19 while the posts have high view counts. Descriptive statistics are shown below. View counts are distributed almost evenly across posts. This tag has consistent presence over the years with a slight decrease in recent years. The tag is frequently used with other tags except for a few (e.g., grub2, UEFI). I word cloud analysis on the title and body of the posts in this cluster. The following figures show similarities in words like Ubuntu, grub, boot, etc. Score vs view count shows good engagement across posts.

	ViewCount	Score	PostTypeId	CreationYear
count	129.000000	129.000000	129.0	129.000000
mean	27997.573643	20.705426	1.0	2015.434109
std	14345.393216	5.127138	0.0	3.259230
min	2079.000000	14.000000	1.0	2010.000000
25%	16581.000000	17.000000	1.0	2012.000000
50%	26310.000000	19.000000	1.0	2015.000000
75%	41331.000000	24.000000	1.0	2018.000000
max	58323.000000	35.000000	1.0	2022.000000





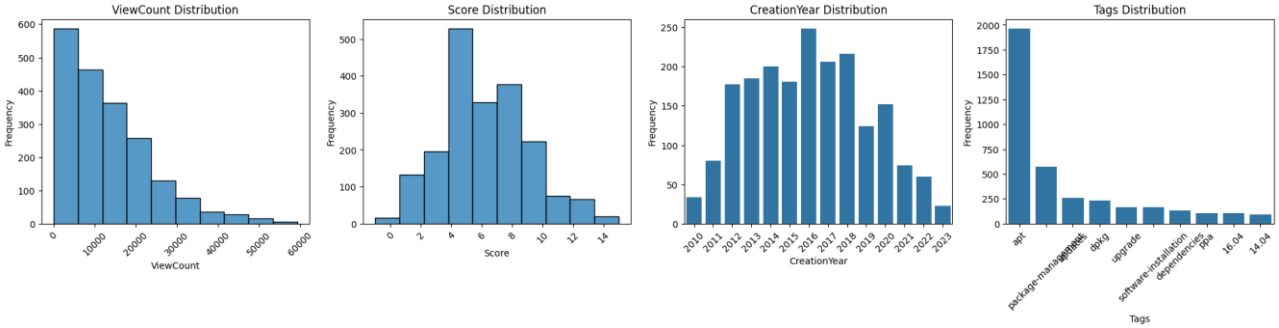


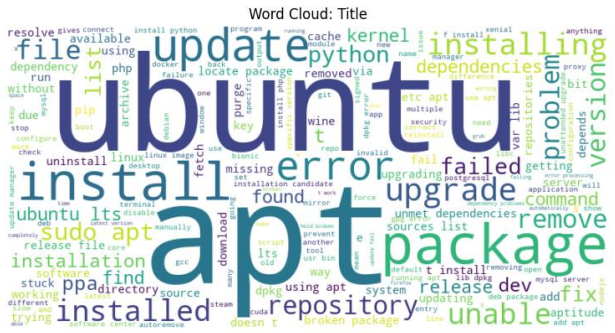


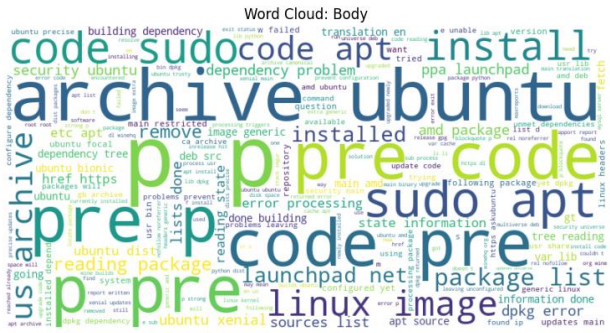
Tag APT. BEST K = 15: Random Cluster 1.

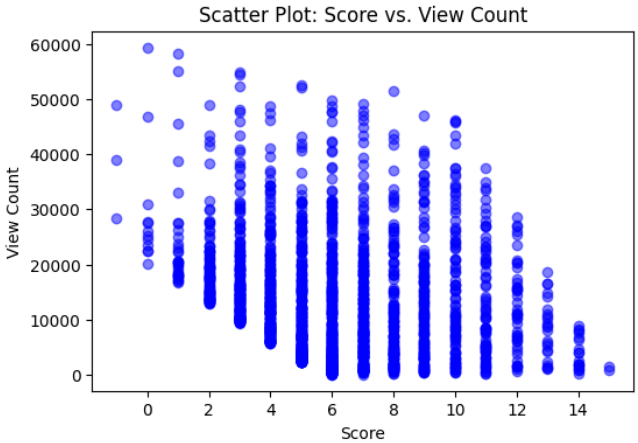
This cluster has 1960 posts. Starting from 2010 to 2023. The score is very low ranging from -1 to 15 as well as the posts have low view counts (few thousands only). Descriptive statistics are shown below. View counts are right skewed while scores are distributed almost evenly across posts. Tag is frequently used with package management, dpkg, upgrade etc. I did a word cloud analysis on the title and body of the posts in this cluster. There is high similarity in common words between title and body. Score vs view count show a slight negative correlation (posts with higher scores are likely to have lower view counts).

	ViewCount	Score	PostTypeId	CreationYear
count	1960.000000	1960.000000	1960.0	1960.000000
mean	13247.508163	6.129082	1.0	2016.031122
std	10600.720857	2.791216	0.0	3.069312
min	57.000000	-1.000000	1.0	2010.000000
25%	4722.750000	4.000000	1.0	2014.000000
50%	11055.500000	6.000000	1.0	2016.000000
75%	18785.000000	8.000000	1.0	2018.000000
max	59371.000000	15.000000	1.0	2023.000000





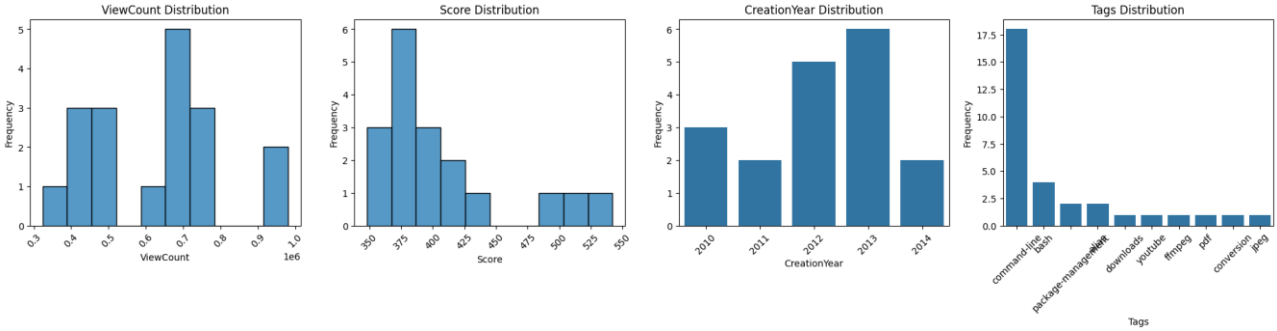


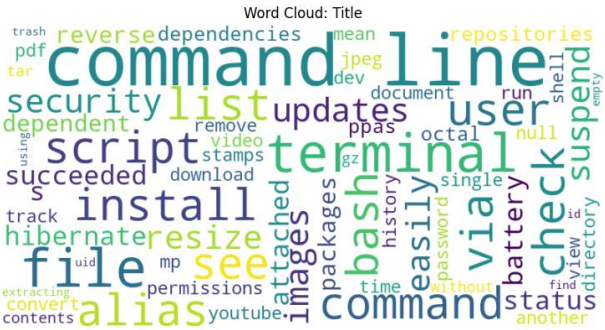


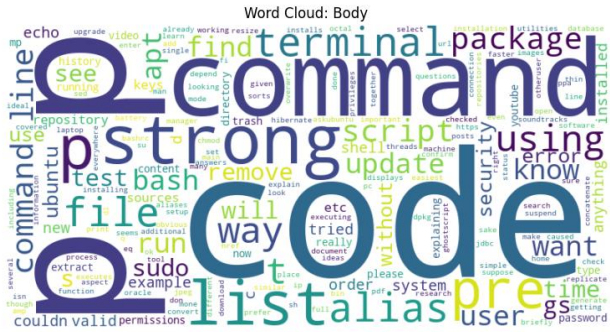
Tag COMMAND-LINE. BEST K = 20: Random Cluster 19.

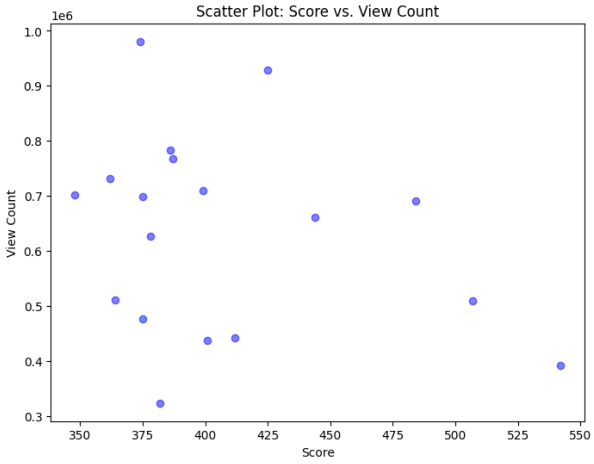
This cluster has 18 posts. Starting from 2010 to 2014. The score is very high ranging from 348 to 542 as well as the posts have high view counts (thousands). Descriptive statistics are shown below. Views counts are not evenly distributed with left and right skewed. Scores are also right skewed. Tags are frequently used with bash or package management. I did a word cloud analysis on the title and body of the posts in this cluster. There is a high similarity in common words between title and body (e.g., command, bash, etc.). Score vs view count somewhat shows a negative correlation (posts with higher scores have low view counts).

	ViewCount	Score	PostTypeId	CreationYear
count	18.000000	18.000000	18.0	18.000000
mean	631675.944444	408.055556	1.0	2012.111111
std	181827.365617	53.550935	0.0	1.278275
min	323189.000000	348.000000	1.0	2010.000000
25%	484476.000000	375.000000	1.0	2011.250000
50%	675933.000000	386.500000	1.0	2012.000000
75%	726002.750000	421.750000	1.0	2013.000000
max	980748.000000	542.000000	1.0	2014.000000





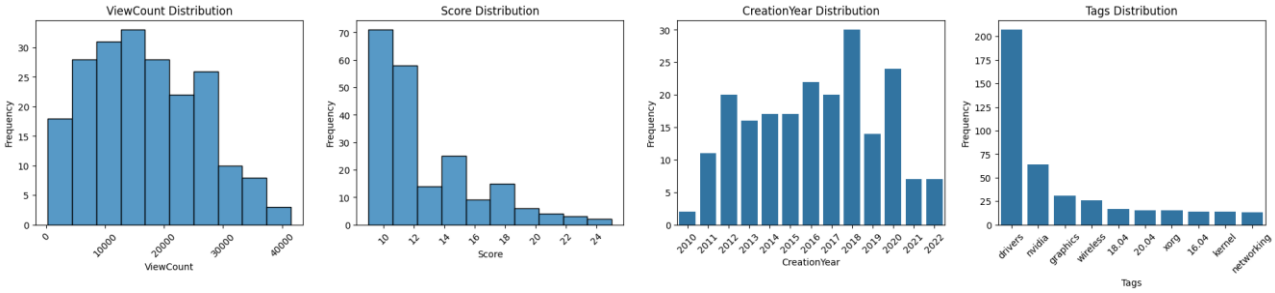


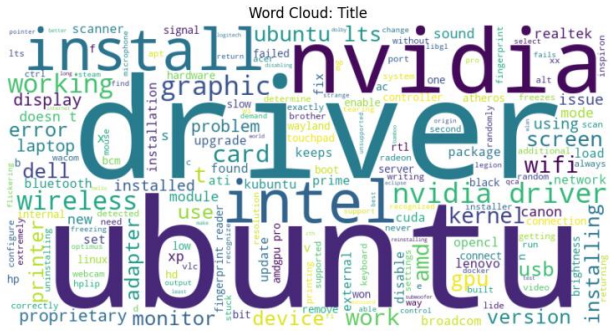


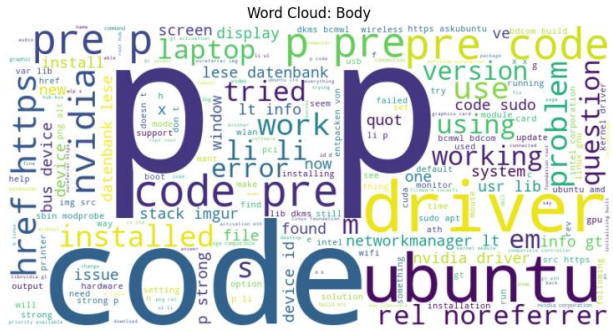
Tag DRIVERS. BEST K = 15: Random Cluster 14.

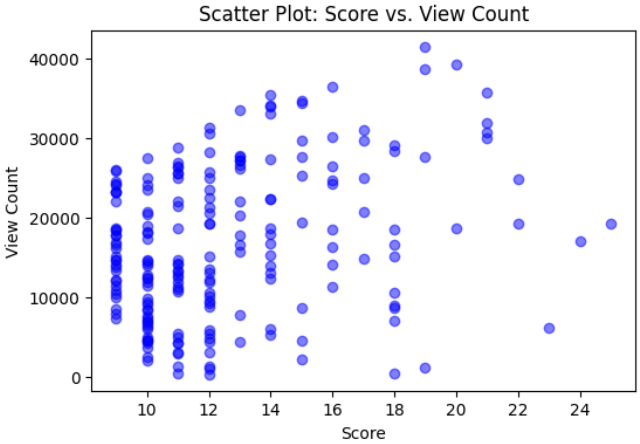
This cluster has 207 posts. Starting from 2010 to 2022. The score is slightly low ranging from 9 to 25 and the posts have high low view counts (few thousands). Descriptive statistics are shown below. View counts are not evenly distributed and left-skewed. Scores are almost normally distributed. Tags are frequently used with NVidia, cuda, etc. I did a word cloud analysis on the title and body of the posts in this cluster. There is a high similarity in common words between title and body (e.g., NVidia, Ubuntu, driver, etc.). Tags also frequently occur are NVidia, graphics, wireless etc. Score vs view count somewhat do not show a correlation.

	ViewCount	Score	PostTypeId	CreationYear
count	207.000000	207.000000	207.0	207.000000
mean	16773.077295	12.613527	1.0	2016.236715
std	9413.194307	3.459251	0.0	3.085671
min	295.000000	9.000000	1.0	2010.000000
25%	9279.000000	10.000000	1.0	2014.000000
50%	15759.000000	12.000000	1.0	2016.000000
75%	24413.500000	14.000000	1.0	2018.500000
max	41497.000000	25.000000	1.0	2022.000000





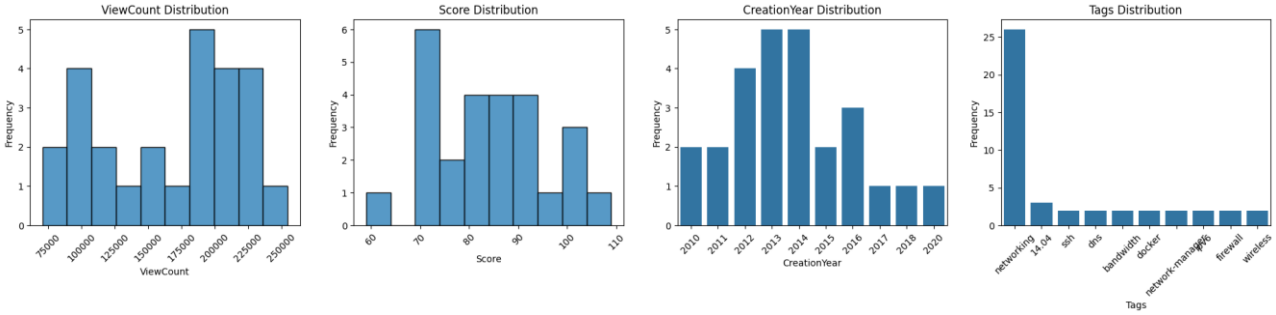


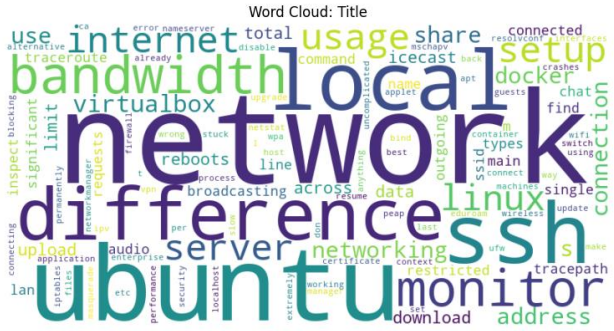


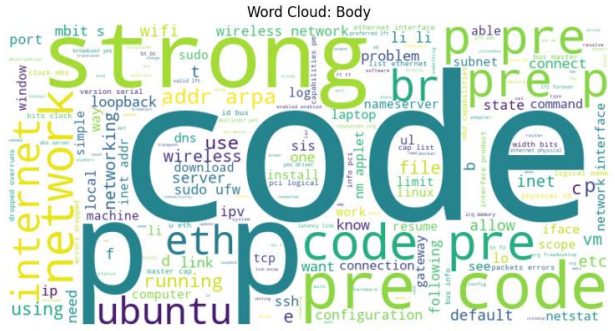
Tag NETWORKING. BEST K = 25: Random Cluster 11.

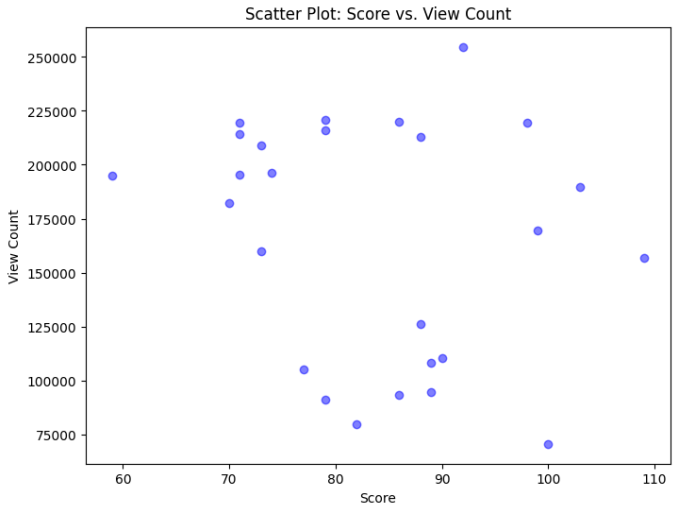
This cluster has 26 posts. Starting from 2010 to 2020. The score is slightly low ranging from 59 to 109 and the posts have slightly high view counts (thousands). Descriptive statistics are shown below. Views counts are evenly distributed are the scores. Tags are not frequently used with other tags to most extent. I did a word cloud analysis on the title and body of the posts in this cluster. There is a high similarity in common words between title and body (e.g., Ubuntu, network, SSH, etc.). Score vs view count somewhat shows a positive correlation (posts with higher scores usually have higher view counts).

	ViewCount	Score	PostTypeId	CreationYear
count	26.000000	26.000000	26.0	26.000000
mean	165795.730769	83.653846	1.0	2013.769231
std	55191.507675	12.079544	0.0	2.405123
min	70626.000000	59.000000	1.0	2010.000000
25%	108688.000000	73.250000	1.0	2012.000000
50%	185876.000000	84.000000	1.0	2013.500000
75%	214003.250000	89.750000	1.0	2015.000000
max	254677.000000	109.000000	1.0	2020.000000









Note: Instructions to run the code are provided in the README.md file in the root folder.