**INTRODUCTION**

**Background**

Social networking sites connect people in the world, allowing them to share pictures, content, videos and share their first-hand opinions on various issues. Big data analytics techniques are highly applied in the social networks because they are characterized by the 5V (Velocity, Volume, Veracity, Value AND Variety) s of big data.(Bazzaz Abkenar *et al.*, 2021). Some examples of social networking sites include Twitter, Facebook etc. Due to social media providing a big source of data, there has been an increase in application of machine learning, deep learning and time series techniques to better understand various problems. A lot of these analysis has been done mostly on twitter data. Twitter which has over 313 million monthly active users and which in a day over 500 million tweets are made, is one of the most utilised social media platforms when it comes to data.(Jianqiang and Xiaolin, 2017)

Some key techniques employed for understanding social media data include, sentiment analysis, times

**Problem statement**

**DATA**

The dataset used for this project was project Tweets dataset, a csv dataset with 1,600,000 observations and five features extracted using the twitter api. It contains the following 5 fields:

* ids: The id of the tweet (eg. 4587)
* date: the date of the tweet (eg. Sat May 16 23:58:44 UTC 2009)
* flag: The query (eg. lyx). If there is no query, then this value is NO\_QUERY.
* user: the user that tweeted (eg. bobthebuilder)
* text: the text of the tweet (eg. Lyx is cool)

**Big data Processing and Storage**

Big data processing is techniques utilised to access large scale data and extract meaningful information from them for decision making.(Mehdipour, Noori and Javadi, 2016), while big data storage are storage technologies that are not relational database systems that can be able to address the Volume, variety and velocity challenges of data.(Strohbach *et al.*, 2016). There are different big data storage and processing technologies available. Processing technologies include Hadoop Map-Reduce or Apache Spark, etc. While storage include either SQL or NoSQL databases such as HBase, HIVE, Spark SQL, Cassandra, MongoDB. Etc

For this project Apache Spark was used for Processing the Project Tweets Data, while MongoDB and spark SQL were used to Populate, Store and save Processed Data.

**Spark SQL**

This is a spark module for structured data processing.

**Big Data**

1. Details of the data storage and processing activities carried out, including preparation of the data and processing the data in a MapReduce/ Spark environment;**[0-30]**

● Source dataset(s) can be stored into an appropriate SQL/ NoSQL database(s) prior to processing by MapReduce / Spark (HBase / HIVE / Spark SQL /Cassandra / MongoDB / etc.) The data can be populated into the NoSQL database using an appropriate tool (Hadoop/ Spark etc.)

● Post Map-reduce processing dataset(s) can be stored into an appropriate NoSQL database(s) (Follow a similar choice as in the previous step)

● Store the data and then follow-up analysis on the output data. It can be extracted from the NoSQL database into another format, using an appropriate tool, if necessary (e.g. extract to CSV to import into R/ Python etc.).

* Source dataset(s) can be stored into an appropriate SQL/ NoSQL database(s) prior to processing by MapReduce / Spark (HBase / HIVE / Spark SQL /Cassandra / MongoDB / etc.) The data can be populated into the NoSQL database using an appropriate tool (Hadoop/ Spark etc.)
* Post Map-reduce processing dataset(s) can be stored into an appropriate NoSQL database(s) (Follow a similar choice as in the previous step)
* Store the data and then follow-up analysis on the output data. It can be extracted from the NoSQL database into another format, using an appropriate tool, if necessary (e.g. extract to CSV to import into R/ Python etc.).

**3. YCSB FOR Comparative analysis OF MYSQL AND MONGODB**

There are various test strategies that can be implemented in order to perform comparative analysis of the capabilities of various big databases. One of the most common is Yahoo cloud service benchmark client (YCSB). YCSB is an open-source license tool used to benchmark new cloud database systems. Through YCSB one can be able to benchmark multiple systems and compare them by creating “workloads”. (Gaikwad and Goje, 2015). A YCSB Comparative analysis was conducted to compare Capabilities of MySQL and MongoDB.

The comparison involved the following areas: -

1. *Workloads a, b, c*
2. *Read and update options*
3. *Distribution (Zipfian and uniform)*

YCSB has various distributions such as: -

Uniform distribution allows all records in the database to be equally chosen.

Zipfian distribution allows one to choose records in accordance to popularity. The most popular become the heads of the distribution and the least popular the tail.

The metrics considered for quantitative analyses were: -

1. *Latency*
2. *Throughput*
3. *Runtime*

The comparison involved comparing the two databases performance based on record counts 1,000, 10,000 and 100,000. Further the read and update proportions were also compared for different proportions and lastly the request distributions compared were Zipfian and uniform distribution, as shown in table below

|  |  |  |
| --- | --- | --- |
| **Comparison Parameters** | **MYSQL** | **MONGODB** |
| Record Counts | 1,000, 10,000 and 100,000 | |
| Request distribution | Zipfian and Uniform | |
| Workload A – Update heavy | * Read: 50% and Update: 50% * Read: 70% and Update: 30% | |
| Workload B- Read heavy | * Read: 95% and Update: 5% * Read: 70% and Update: 30/5 | |
| Workload C- Read only | * Read: 100% and Update:0% * Read: 70% and Update: 30% | |

**WORKLOAD A**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **50/50 (Zipfian)** | | | | | | | | | |
| **Read/update proportions** | **MySQL** | | | **MongoDB** | | | | | |
| **Record Counts** | **1000** | **10,000** | **100,000** | **1000** | | **10,000** | | **100,000** | |
| RunTime(ms) | 9957 | 71289 | 873740 | 1074 | | 4381 | | 20910 | |
| Throughput(ops/sec) | 100.43 | 140.2741 | 114.45 | 931.099 | | 2282.584 | | 4782.40077 | |
| Average Latency(us) | 7741.99 | 7075.872 | 8724.64 | 582.738 | | 369.638 | | 195.51202 | |
| Min Latency(us) | 3046 | 3994 | 3294 | 170 | | 56 | | 35 | |
| Max Latency(us) | 128319 | 48863 | 69271551 | 146047 | | 158591 | | 1950719 | |
| 95thPercentileLatency(us) | 12063 | 10143 | 13183 | 1037 | | 796 | | 454 | |
| 99thPercentileLatency(us) | 25023 | 16927 | 29855 | 2001 | | 1394 | | 780 | |
| **70/30 (**Zipfian**)** | | | | | | | | | |
| **Read/update proportions** | **MySQL** | | | **MongoDB** | | | | | |
| **Record Counts** | **1,000** | **10,000** | **100,000** | **1,000** | | **10,000** | | **100,000** | |
| RunTime(ms) | 6865 | 67359 | 706801 | 1864 | | 7666 | | 42974 | |
| Throughput(ops/sec) | 145.666 | 148.458 | 141.4825 | 536.4807 | | 1304.46 | | 2326.98841 | |
| Average Latency(us) | 6386.185 | 6679.51 | 7050.079 | 708.68 | | 650.1487 | | 412.20452 | |
| Min Latency(us) | 3688 | 3582 | 2712 | 157 | | 65 | | 34 | |
| Max Latency(us) | 42591 | 58879 | 169983 | 154879 | | 228991 | | 159999 | |
| 95thPercentileLatency(us) | 9599 | 9199 | 10951 | 1320 | | 1647 | | 884 | |
| 99thPercentileLatency(us) | 19199 | 15391 | 14855 | 3889 | | 5223 | | 2305 | |
| **50/50 (Uniform**) | | | | | | | | | |
| **Read/update proportions** | **MySQL** | | | | **MongoDB** | | | | |
| **Record Counts** | **1000** | **10,000** | **100,000** | | **1000** | | **10,000** | | **100,000** |
| RunTime(ms) | 8267 | 99077 | 1052529 | | 1756 | | 4860 | | 20853 |
| Throughput(ops/sec) | 120.9629 | 100.93159 | 95.009259 | | 569.4761 | | 2057.6132 | | 4795.47307 |
| Average Latency(us) | 7723.14 | 9829.2211 | 10504.895 | | 706.401 | | 382.9785 | | 196.65383 |
| Min Latency(us) | 4528 | 3524 | 3384 | | 142 | | 58 | | 35 |
| Max Latency(us) | 74879 | 272127 | 336895 | | 150911 | | 141695 | | 208511 |
| 95thPercentileLatency(us) | 11439 | 14167 | 16639 | | 1348 | | 744 | | 615 |
| 99thPercentileLatency(us) | 52447 | 72639 | 23951 | | 2921 | | 1543 | | 984 |
| **70/30 (uniform)** | | | | | | | | | |
| **Read/update proportions** | **MySQL** | | | | **MongoDB** | | | | |
| **Record Counts** | **1,000** | **10,000** | **100,000** | | **1,000** | | **10,000** | | **100,000** |
| RunTime(ms) | 13010 | 75478 | 866234 | | 1209 | | 3855 | | 19506 |
| Throughput(ops/sec) | 76.86395081 | 132.4889372 | 115.4422477 | | 827.12986 | | 2594.033722 | | 5126.62770 |
| Average Latency(us) | 11686.936 | 7475.8006 | 8649.02499 | | 489.766 | | 295.5393 | | 182.21613 |
| Min Latency(us) | 6052 | 4108 | 3776 | | 141 | | 44 | | 35 |
| Max Latency(us) | 106239 | 133759 | 365055 | | 142719 | | 231935 | | 280831 |
| 95thPercentileLatency(us) | 18607 | 10679 | 13735 | | 916 | | 647 | | 606 |
| 99thPercentileLatency(us) | 53951 | 24127 | 19855 | | 1481 | | 1475 | | 966 |

* The runtime(ms) when read and update is 50/50 and the distribution is Zipfian for both MySQL and MongoDB are increasing as the records count increases. MongoDB has a lower runtime compared to MySQL across the three record counts. The runtime(ms) when read and update is 70/30 and the distribution is Zipfian for both MySQL and MongoDB are increasing as the records count increases. MongoDB has a lower runtime compared to MySQL across the three record counts. When you increase the read and update from 50/50 to 70/30, MongoDB still performs well compared to MySQL, but MongoDB with 50/50 has a lower runtime compared mongo with 70/30, while MySQL with 50/50 has higher runtime compared to MySQL with 70/30.
* The runtime(ms) when read and update is 50/50 and the distribution is uniform for both MySQL and MongoDB are increasing as the records count increases. MongoDB has a lower runtime compared to MySQL across the three record counts. The runtime(ms) when read and update is 70/30 and the distribution is uniform for both MySQL and MongoDB are increasing as the records count increases. MongoDB has a lower runtime compared to MySQL across the three record counts. When you increase the read and update from 50/50 to 70/30, MongoDB still performs well compared to MySQL, but MongoDB with 50/50 has a higher runtime compared mongo with 70/30, while MySQL with 50/50 has lower runtime compared to MySQL with 70/30.
* When the read and update is either 70/30 or 50/50 and the distribution is Zipfian for MySQL, the throughput increases, decreases then increases, while for MongoDB it increases only. The throughput is higher in MongoDB than in MySQL for both 50/50 and 70/30. Increasing the read to 70 and reducing update to 30 for both databases, causes an increase in throughput for both databases.
* When the read and update is 70/30 and the distribution is uniform for MySQL, the throughput decreases, increases then decreases, while when it is 50/50 the throughput decreases as the record counts increase, while for MongoDB it only increases in both proportions. The throughput is higher in MongoDB than in MySQL for both 50/50 and 70/30. Increasing the read to 70 and reducing update to 30 for both databases, causes an increase in throughput for both databases.

**WORKLOAD B**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **95/5 (**Zipfian) | | | | | | |
| **Read/update proportions** | **MySQL** | | | **MongoDB** | | |
| **Record Counts** | **1000** | **10,000** | **100,000** | **1000** | **10,000** | **100,000** |
| RunTime(ms) | 7103 | 74965 | 691730 | 1392 | 6781 | 20141 |
| Throughput(ops/sec) | 140.785584 | 133.395585 | 144.565076 | 718.3908046 | 1474.708745 | 4964.9968 |
| Average Latency(us) | 6577.019 | 7435.8522 | 6906.23396 | 746.279 | 459.3124 | 189.63952 |
| Min Latency(us) | 4056 | 3086 | 3272 | 211 | 62 | 38 |
| Max Latency(us) | 70783 | 106367 | 120959 | 104703 | 258943 | 636415 |
| 95thPercentileLatency(us) | 9927 | 15135 | 10535 | 1222 | 836 | 615 |
| 99thPercentileLatency(us) | 19551 | 22255 | 14239 | 3413 | 2301 | 1162 |
| **70/30 (**Zipfian**)** | | | | | | |
| **Read/update proportions** | **MySQL** | | | **MongoDB** | | |
| **Record Counts** | **1,000** | **10,000** | **100,000** | **1,000** | **10,000** | **100,000** |
| RunTime(ms) | 7790 | 65809 | 720646 | 1609 | 5328 | 21621 |
| Throughput(ops/sec) | 128.3697047 | 151.9548998 | 138.7643864 | 621.5040398 | 1876.876877 | 4625.13297 |
| Average Latency(us) | 7132.117 | 6507.8252 | 7193.95318 | 708.68 | 650.1487 | 412.20452 |
| Min Latency(us) | 4070 | 3522 | 3118 | 952.147 | 468.6321 | 206.54024 |
| Max Latency(us) | 61887 | 52575 | 117567 | 180 | 66 | 40 |
| 95thPercentileLatency(us) | 11391 | 8783 | 11559 | 198783 | 182399 | 135167 |
| 99thPercentileLatency(us) | 22687 | 15287 | 17359 | 1404 | 852 | 641 |
| **95/5(uniform)** | | | | | | |
| **Read/update proportions** | **MySQL** | | | **MongoDB** | | |
| **Record Counts** | **1,000** | **10,000** | **100,000** | **1,000** | **10,000** | **100,000** |
| RunTime(ms) | 10869 | 85038 | 925886 | 1278 | 2987 | 23337 |
| Throughput(ops/sec) | 92.00478425 | 117.5944872 | 108.0046572 | 782.4726135 | 3347.840643 | 4285.0409 |
| Average Latency(us) | 10219.568 | 8427.0489 | 9234.20616 | 647.787 | 218.3129 | 220.75861 |
| Min Latency(us) | 4224 | 3898 | 3974 | 134 | 55 | 35 |
| Max Latency(us) | 197759 | 231679 | 326399 | 139647 | 169471 | 209791 |
| 95thPercentileLatency(us) | 14671 | 12791 | 14583 | 1067 | 492 | 459 |
| 99thPercentileLatency(us) | 127743 | 35167 | 20223 | 2911 | 2527 | 1580 |
| **70/30(uniform)** | | | | | | |
| **Read/update proportions** | **MySQL** | | | **MongoDB** | | |
| **Record Counts** | **1,000** | **10,000** | **100,000** | **1,000** | **10,000** | **100,000** |
| RunTime(ms) | 12660 | 107940 | 893522 | 1355 | 3375 | 22295 |
| Throughput(ops/sec) | 78.98894155 | 92.64406152 | 111.9166624 | 738.0073801 | 2962.962963 | 4485.3106 |
| Average Latency(us) | 11138.58 | 10692.0564 | 8916.45785 | 703.248 | 274.3139 | 212.46296 |
| Min Latency(us) | 4120 | 4900 | 3554 | 104 | 40 | 34 |
| Max Latency(us) | 154239 | 167935 | 386559 | 215679 | 188799 | 173055 |
| 95thPercentileLatency(us) | 29759 | 15927 | 14695 | 953 | 566 | 490 |
| 99thPercentileLatency(us) | 65503 | 36927 | 21135 | 3143 | 1521 | 784 |

* The runtime(ms) when read and update is 50/50 and the distribution is Zipfian for both MySQL and MongoDB are increasing as the records count increases. MongoDB has a lower runtime compared to MySQL across the three record counts. The runtime(ms) when read and update is 70/30 and the distribution is Zipfian for both MySQL and MongoDB are increasing as the records count increases. MongoDB has a lower runtime compared to MySQL across the three record counts. When you increase the read and update from 50/50 to 70/30, MongoDB still performs well compared to MySQL, but MongoDB with 50/50 has a lower runtime compared mongo with 70/30, while MySQL with 50/50 has higher runtime compared to MySQL with 70/30.
* The runtime(ms) when read and update is 50/50 and the distribution is uniform for both MySQL and MongoDB are increasing as the records count increases. MongoDB has a lower runtime compared to MySQL across the three record counts. The runtime(ms) when read and update is 70/30 and the distribution is uniform for both MySQL and MongoDB are increasing as the records count increases. MongoDB has a lower runtime compared to MySQL across the three record counts. When you increase the read and update from 50/50 to 70/30, MongoDB still performs well compared to MySQL, but MongoDB with 50/50 has a higher runtime compared mongo with 70/30, while MySQL with 50/50 has lower runtime compared to MySQL with 70/30.
* When the read and update is either 70/30 or 50/50 and the distribution is Zipfian for MySQL, the throughput increases, decreases then increases, while for MongoDB it increases only. The throughput is higher in MongoDB than in MySQL for both 50/50 and 70/30. Increasing the read to 70 and reducing update to 30 for both databases, causes an increase in throughput for both databases.
* When the read and update is 70/30 and the distribution is uniform for MySQL, the throughput decreases, increases then decreases, while when it is 50/50 the throughput decreases as the record counts increase, while for MongoDB it only increases in both proportions. The throughput is higher in MongoDB than in MySQL for both 50/50 and 70/30. Increasing the read to 70 and reducing update to 30 for both databases, causes an increase in throughput for both databases.

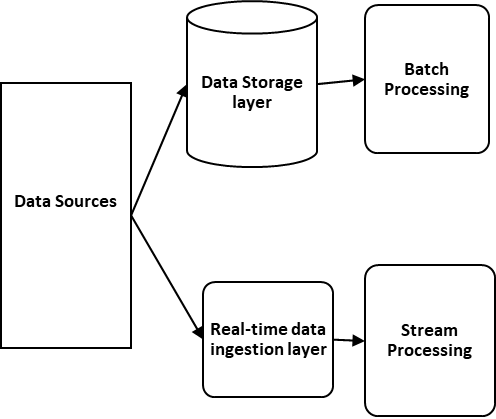
**WORKLOAD C**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **100/0 (Zipfian**) | | | | | | |
| **Read/update proportions** | **MySQL** | | | **MongoDB** | | |
| **Record Counts** | **1000** | **10,000** | **100,000** | **1000** | **10,000** | **100,000** |
| RunTime(ms) | 6775 | 83461 | 729684 | 1317 | 10183 | 31389 |
| Throughput(ops/sec) | 147.6015 | 119.81644 | 137.04563 | 759.3014 | 982.02887 | 3185.8294 |
| Average Latency(us) | 6358.314 | 8269.7365 | 7286.0314 | 751.502 | 920.2061 | 297.82761 |
| Min Latency(us) | 4140 | 3608 | 2876 | 185 | 53 | 35 |
| Max Latency(us) | 43423 | 150911 | 138239 | 166911 | 316927 | 240127 |
| 95thPercentileLatency(us) | 9455 | 15631 | 11743 | 1291 | 2327 | 1051 |
| 99thPercentileLatency(us) | 23391 | 48799 | 18271 | 5603 | 8343 | 2249 |
| **70/30 (Zipfian)** | | | | | | |
| **Read/update proportions** | **MySQL** | | | **MongoDB** | | |
| **Record Counts** | **1,000** | **10,000** | **100,000** | **1,000** | **10,000** | **100,000** |
| RunTime(ms) | 6888 | 67839 | 806177 | 4332 | 4076 | 26532 |
| Throughput(ops/sec) | 145.1800 | 147.40783 | 124.04224 | 230.8403 | 2453.3857 | 3769.0336 |
| Average Latency(us) | 6417.907 | 6723.8165 | 8046.5100 | 1383.361 | 337.5972 | 247.6615 |
| Min Latency(us) | 2812 | 4068 | 3056 | 160 | 59 | 35 |
| Max Latency(us) | 57087 | 55231 | 317951 | 447487 | 272639 | 318719 |
| 95thPercentileLatency(us) | 9311 | 9335 | 13703 | 2763 | 777 | 770 |
| 99thPercentileLatency(us) | 21071 | 15215 | 23631 | 8831 | 1930 | 1547 |
| **100/0(uniform)** | | | | | | |
| **Read/update proportions** | **MySQL** | | | **MongoDB** | | |
| **Record Counts** | **1,000** | **10,000** | **100,000** | **1,000** | **10,000** | **100,000** |
| RunTime(ms) | 13301 | 102915 | 1052980 | 1140 | 2960 | 27718 |
| Throughput(ops/sec) | 75.18232 | 97.167565 | 94.968565 | 877.193 | 3378.3784 | 3607.7639 |
| Average Latency(us) | 11910.05 | 10211.023 | 10480.258 | 457.643 | 215.4459 | 267.83158 |
| Min Latency(us) | 4800 | 5148 | 3802 | 129 | 57 | 35 |
| Max Latency(us) | 181375 | 291839 | 513535 | 153727 | 242047 | 157951 |
| 95thPercentileLatency(us) | 26463 | 14951 | 16735 | 690 | 438 | 478 |
| 99thPercentileLatency(us) | 82687 | 29439 | 67199 | 1569 | 1118 | 667 |
| **70/30(uniform)** | | | | | | |
| **Read/update proportions** | **MySQL** | | | **MongoDB** | | |
| **Record Counts** | **1,000** | **10,000** | **100,000** | **1,000** | **10,000** | **100,000** |
| RunTime(ms) | 9059 | 79598 | 1035559 | 1236 | 3631 | 24200 |
| Throughput(ops/sec) | 110.39 | 125.6313 | 96.566202 | 809.0615 | 2754.0622 | 4132.2314 |
| Average Latency(us) | 7699.7 | 7868.9411 | 10332.534 | 635.32 | 297.9136 | 233.74234 |
| Min Latency(us) | 3732 | 3962 | 3864 | 125 | 47 | 35 |
| Max Latency(us) | 120255 | 109695 | 360703 | 233727 | 134527 | 206207 |
| 95thPercentileLatency(us) | 10431 | 11415 | 17695 | 885 | 528 | 475 |
| 99thPercentileLatency(us) | 44351 | 21567 | 26527 | 2209 | 1496 | 745 |

* The runtime(ms) when read and update is 50/50 and the distribution is Zipfian for both MySQL and MongoDB are increasing as the records count increases. MongoDB has a lower runtime compared to MySQL across the three record counts. The runtime(ms) when read and update is 70/30 and the distribution is Zipfian for both MySQL and MongoDB are increasing as the records count increases. MongoDB has a lower runtime compared to MySQL across the three record counts. When you increase the read and update from 50/50 to 70/30, MongoDB still performs well compared to MySQL, but MongoDB with 50/50 has a lower runtime compared mongo with 70/30, while MySQL with 50/50 has higher runtime compared to MySQL with 70/30.
* The runtime(ms) when read and update is 50/50 and the distribution is uniform for both MySQL and MongoDB are increasing as the records count increases. MongoDB has a lower runtime compared to MySQL across the three record counts. The runtime(ms) when read and update is 70/30 and the distribution is uniform for both MySQL and MongoDB are increasing as the records count increases. MongoDB has a lower runtime compared to MySQL across the three record counts. When you increase the read and update from 50/50 to 70/30, MongoDB still performs well compared to MySQL, but MongoDB with 50/50 has a higher runtime compared mongo with 70/30, while MySQL with 50/50 has lower runtime compared to MySQL with 70/30.
* When the read and update is either 70/30 or 50/50 and the distribution is Zipfian for MySQL, the throughput increases, decreases then increases, while for MongoDB it increases only. The throughput is higher in MongoDB than in MySQL for both 50/50 and 70/30. Increasing the read to 70 and reducing update to 30 for both databases, causes an increase in throughput for both databases.
* When the read and update is 70/30 and the distribution is uniform for MySQL, the throughput decreases, increases then decreases, while when it is 50/50 the throughput decreases as the record counts increase, while for MongoDB it only increases in both proportions. The throughput is higher in MongoDB than in MySQL for both 50/50 and 70/30. Increasing the read to 70 and reducing update to 30 for both databases, causes an increase in throughput for both databases.

1. A discussion of the rationale and justification for the choices you have made in terms of data processing and storage, programming language choice, that you have implemented.**[0-20]**
2. Design the architecture for the processing of big data using all the necessary technologies (HADOOP/SPARK,NOSQL/SQL databases and programming). Present your Design in the form of a diagram and discussion in your report **.[0-20]**

Big data architecture handles the ingestion, storage, processing and analysis of data that is big data.



**Data Analytics and Machine Learning**

**Data Visualization and Reporting**

**Advanced Data Analytics**

1. A discussion of the rationale, evaluation, and justification for the choices you have made in terms of EDA, data wrangling, machine learning models and algorithms that you have implemented**.[0-40]**
2. Provide evidence and justification of your choice of sentiment extraction techniques.
3. Explore at least 2 methods of time-series forecasting including at least 1 Neural Network and 1 autoregressive model (ARIMA, SARIMA etc…) . (Hint: that this is a Short time series, How are you going to handle this?)
4. Evidence and justify your choices for your final analysis and include your forecasts at 1 day, 3 days and 7 days going forward.
5. Your dashboard must be dynamic and interactive. Include your design rationale expressing Tufts principles.
6. **E**valuation and justification of the hyperparameter tuning techniques that you have used **[0-20]**
7. Your analysis of any change sentiment that occurs and your forecast of the sentiment at 1 day, 3 days and 7 days going forward**[0-20]**
8. Presentation of results by making appropriate use of figures along with caption, tables, etc and your dashboard for your forecast, Discuss Tufts Principles in relation to your Dashboard **.[0-20]**