Assignment 1 Report

Working with Edgar datasets:

Wrangling, Pre-processing and exploratory data analysis

Course: INFO7390

Advance Data Science & Architecture

Professor:
Srikanth Krishnamurthy

Submitted by:

Team 9

Aashri Tandon

Pragati Shaw

Sarthak Agarwal

Table of Contents

Programming Language and Libraries

To solve this assignment, we have used Python as out programming language and the following libraries to tackle different problems.

- urllib - for web scraping
- BeautifulSoup - for web scraping
- csv - for writing csv
- logging - for logging
- os – for navigating, creating and deleting directories and files
- zipfile – handling zip files
- boto – for handling AWS S3
- shutil – for removing files
- pandas – for putting data into dataframes.
- Glob – for reading files efficiently.

Logging

Log entries are generated at each and every step of the program in problem 1 and 2. We have used the logging module in python which will log the operation with timestamp, levelname and the message that we have customized.

We have maintained three levels in our logs.

1. Error: One can find this level by looking for the string ERROR in the log and the occurrence of this will result in the program to exit after displaying the cause of the error.
2. Warning: This level is raised that needs attention but will not result in the program to fail.
3. Info: These are the messages that gives information about each step as it gets executed in the program.

In addition to logging, the same messages will be outputted on the console as well.

Code

The code and other resources for this assignment is available at the following github repository:

https://github.com/agsarthak/AdvanceDataScience_INFO7390/tree/master/Assignment1

Problem 1: Data wrangling Edgar data from text files

(Combined implementation for Part1 and Part2)

The objective of this problem is to extract all statistical tables from 10Q or 10K filings using Python.

1. Program takes 5 command line arguments for cik, accessionNumber, Amazon accessKey , Amazon secretAccessKey and location in the following format, but the order can be changed:

   For example: python Problem1.py accessKey=*YOUR_AMAZON_ACCESS_KEY* cik=*CIK* location=*S3_LOCATION* secretKey=*YOUR_AMAZON_SECRET_ACCESS_KEY* accessionNumber=*ACCESSION_NUMBER*

2. The program parses the command line arguments and puts them inside local variables.
3. Exception Handling:
   - If amazon keys are not provided, then program exits.
   - If cik or accessionNumber is not provided, we have taken a default cik and accessionNumber for IBM
   - If location is not provided, it creates the bucket in default location.
4. Establish connection to S3, if keys are invalid, the program exits after logging appropriate details.
5. With the help of python's urllib library, we are opening the requested URL. We are using BeautifulSoup library for handling html tags in python.
6. Once the url is open, we find all the <a> tags and their href attributes and then look for '10q' and '10-k' pattern. If nothing is found, we exit out of program, otherwise we open that url.
7. Once the 10q/10k is open, we fetch all the tables by looking for tables present inside <div> tag.
8. Fetching the tables doesn't solve our problem, we need to find refined tables that contain statistical data. We found that the statistical data tables contain either '$' or '%' in table data. So we iterate through tables and look for pattern. If found, we break out of loop.
9. In refined tables, we need to clean the table data which is inside <td> tags. We remove unwanted characters such as '\n' or '\xa0' characters.
10. After the data is clean, we create a corresponding csv file for the table inside extracted_csv folder.
11. The program zips the folder and put inside Problem1.zip.
12. Lastly, we create a bucket and upload the zip file. The bucket name is always unique as it's the concatenation of ACCESS_KEY+Current_Timestamp. If the keys are invalid, the program will log an error and exit.

Running the code for Problem 1

1. Created a docker image file-

```
FROM continuumio/anaconda3
ADD Problem1.py /
CMD [ "python", "./Problem1.py" ]
```

2. Building a docker image:
   docker build -t problem1-image .

3. Running the docker image:
   docker run problem1-image python Problem1.py cik=CIK_HERE accessionNumber=AACESSION_NUMBER_HERE accessKey=AWS_ACCESS_KEY_HERE secretKey=AWS_SECRET_ACCESS_KEY_HERE location=LOCATION_HERE

4. The docker image is available on docker hub. Use the following command:
   docker pull aashritandon/problem1

Problem 2: Missing Data Analysis and Visualization

Part 1:

To tackle Problem 2, we have performed the following steps in our Python program:

1. Initialize log file.

2. Cleanup required directories.
   Exception handling: If desired directories not present, create it.

3. For a year as input, download the log files for all the months.
   Exception Handling:
   - If no year is inputted by user, set year as 2003 by default.
   - If year is not between 2003 and 2016, display warning and exit the program.

   Handling missing data:
   - If log is not available for first day of the month, check for next day and so forth.
   - If log file is empty, download log for next date.

4. Unzip the downloaded log files and extract the csv.

5. Load all the csv into individual dataframe.

6. For each log file we check for anomalies like:
   a. Count the null values for each variable.
   b. Check if the variables *idx*, *norefer* and *noagent* have any other values except zero and one.

7. For each log file handle missing value.
   a. If any of the variables: cik, ip, accession, data or time are null(NaN), then skip that row. We choose this approach because these variables are most important for the log file in Edgar and no analysis is correct without them.
   b. For the variable: *browser*, replace NaN with the most used *browser* in the data.

c. For the variable: *idx*, replace NaN with the most used *idx* in the data.
d. For the variable: *norefer*, replace NaN with 1 because as per the definition of this variable it takes on value of one if the Apache log file user agent field is empty.
e. For the variable: *noagent*, replace NaN with 1 because as per the definition of this variable it takes on value of one if the Apache log file referrer field is empty.
f. For the variable: *code*, replace NaN with the most used *code* in the data.
g. For the variable: *find*, replace NaN with the most used *find* in the data.
h. For the variable: *crawler*, replace NaN with 0 because as per the definition of this variable it takes on a value of one if the user agent self-identifies as one of the following webcrawlers or has a user code of 404. Hence, we assume that empty means zero.
i. For the variable: *extension*, replace NaN with the most used *extension* in the data.
j. For the variable: *zone*, replace NaN with the most used *zone* in the data.
k. For the variable: *size,* replace NaN with the mean of the *size* in the data.

8. For each log file, compute the summary metrics.
    a. Calculate the mean of the variable size and append it to the dataframe as a new column.
    b. Calculate the maximum used browser and append it to the dataframe as a new column.

9. Check for anomalies in the data and remove the row if found.

10. Combine all the individual logs (which are currently individual dataframes) into a single dataframe and export it to a csv.

11. Compute the summary metrics for the entire log file generated above and export it to a csv.

12. Lastly, create a bucket on S3 and upload both the files.
    Exception handling:
    - Made the bucket name unique by making it as a concatenation of ACCESS_KEY + Current_Timestamp.
    - If the keys are invalid, the program will log an error and exit.
    - If amazon keys are not provided, then program exits after displaying the appropriate message on console.
    - If location is not provided, it creates the bucket in default location.

Running the code for Problem 2

1. Created a docker image file-
    ```
    FROM continuumio/anaconda3
    ADD Problem2.py /
    CMD [ "python", "./Problem2.py" ]
    ```

2. Building a docker image:
    ```
    docker build -t problem2-image .
    ```

3. Running the docker image:

The program takes four arguments. Year, Amazon accessKey , Amazon secretAccessKey and location in the following format, but the order can be changed:

docker run problem2-image python Problem2.py year= YEAR_HERE accessKey=AWS_ACCESS_KEY_HERE secretKey=AWS_SECRET_ACCESS_KEY_HERE location=LOCATION_HERE

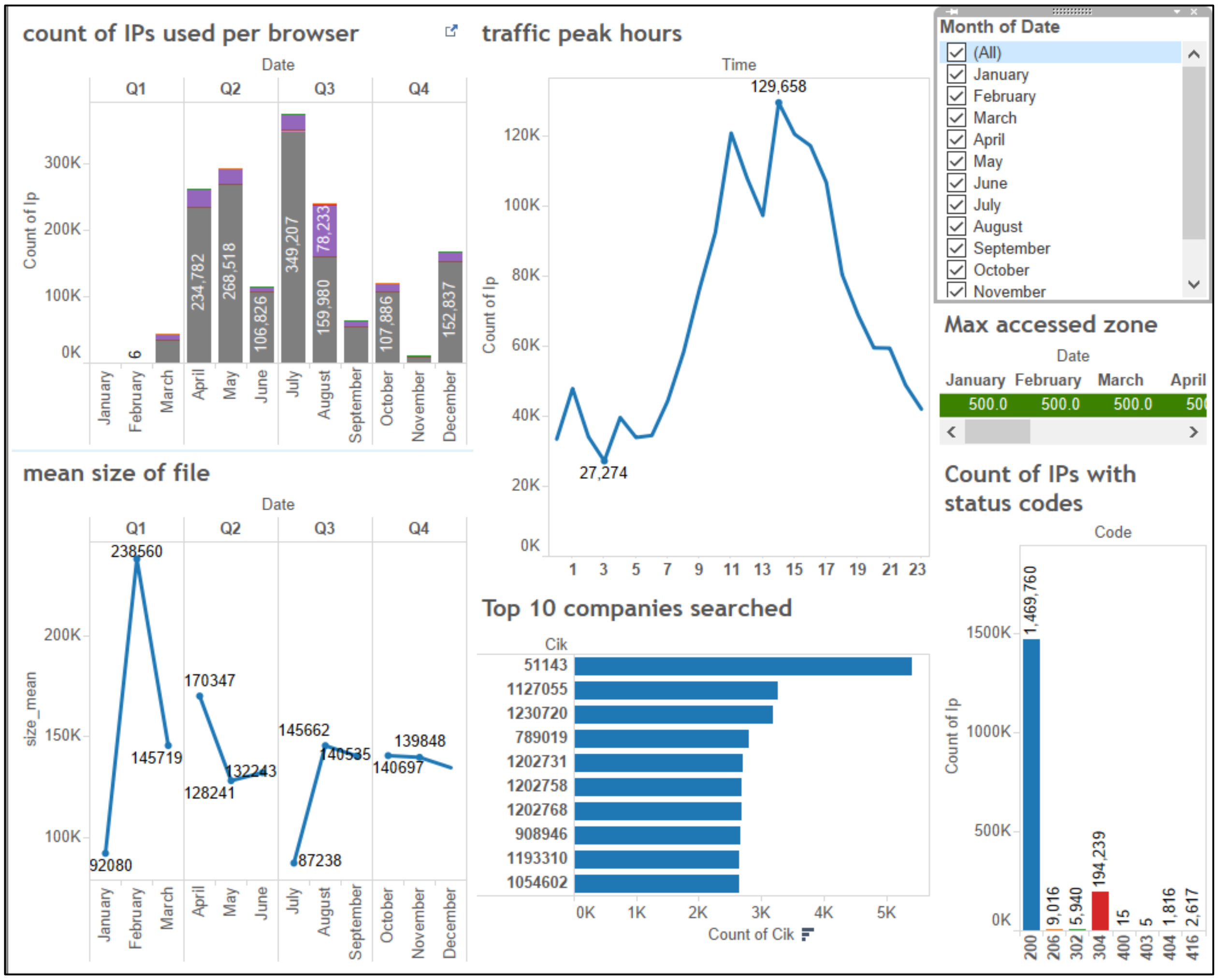
4. The docker image is available on docker hub. Use the following command:

   docker pull aashritandon/problem2

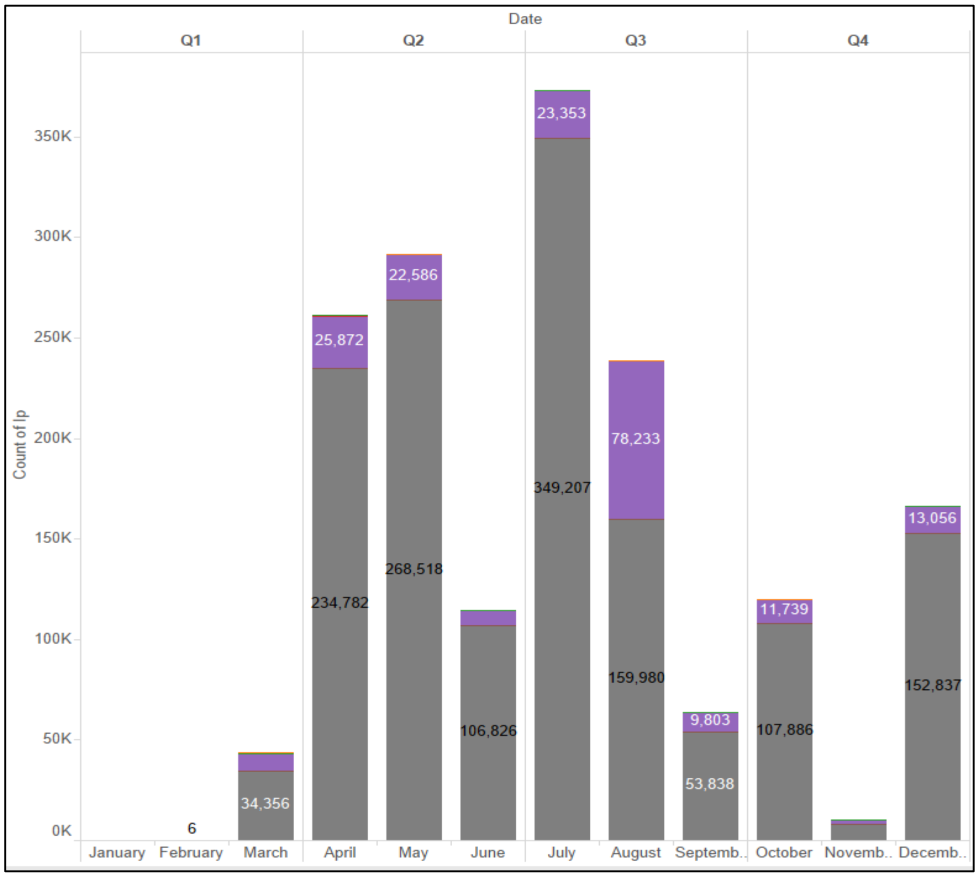
Part 2: Visualization on Tableau:

1. Downloaded the data from S3 and extracted the data in Tableau.
2. The data visualization is done to analyze the extracted data.
3. The tableau workbook has been published, and can be accessed by clicking on the following link: https://public.tableau.com/profile/publish/ADSTeam9Assignment1/Dashboard1

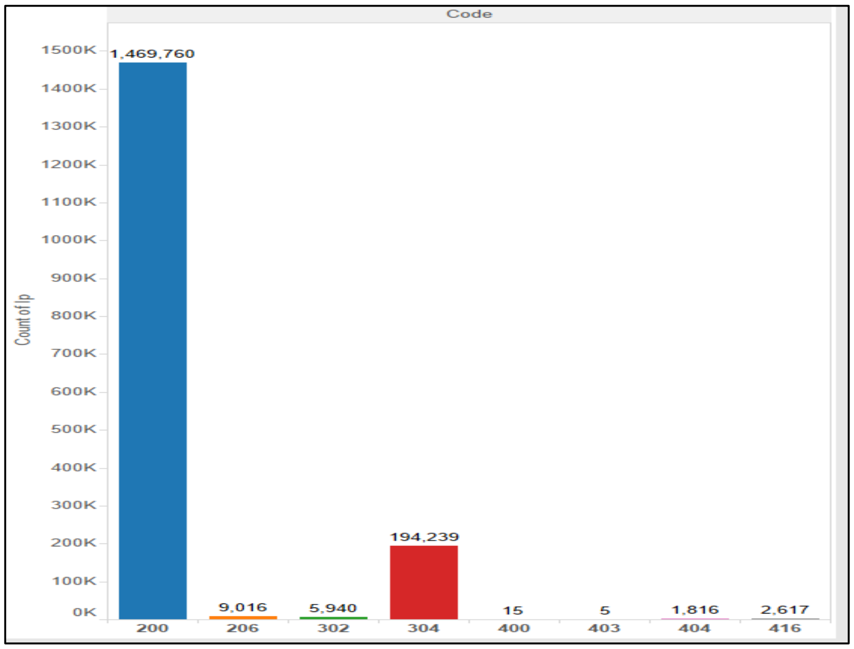
The dashboard helps us to analyze the following while applying filter on month/peak hour/browser :



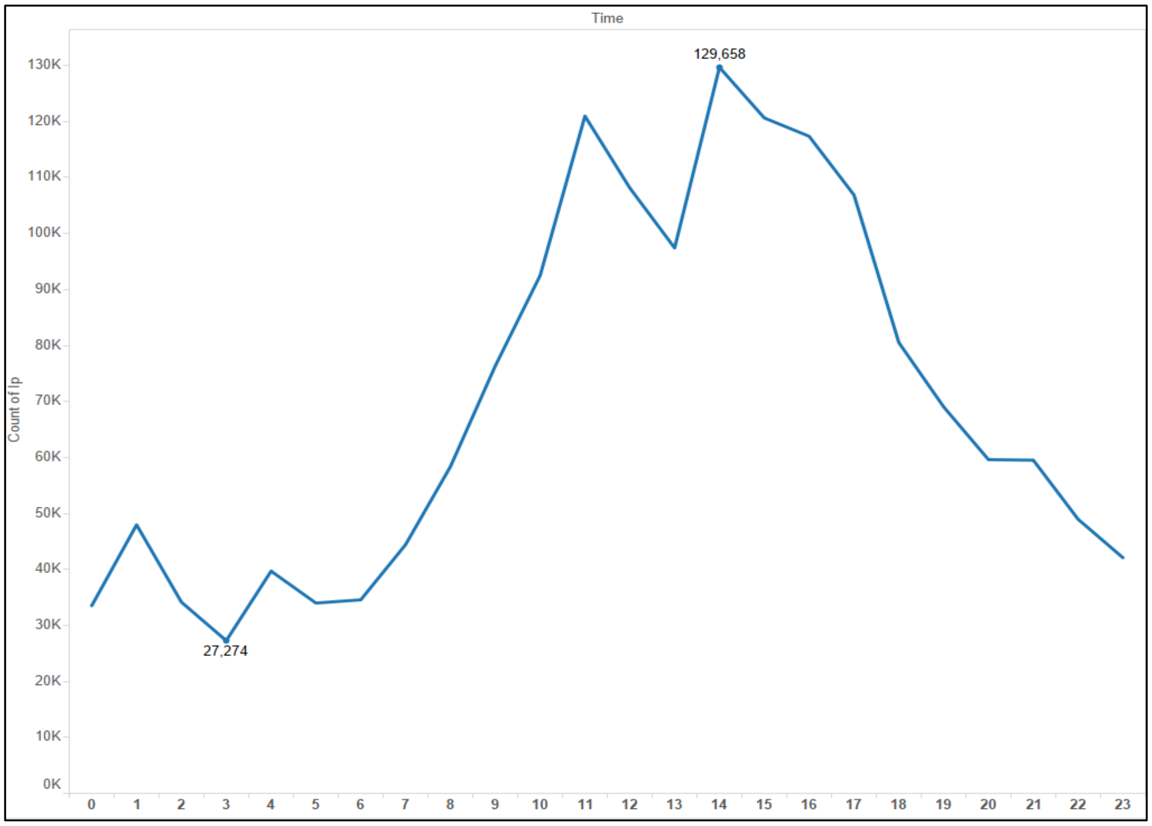
- **Count of users for types browsers that are used for EDGAR filings.**



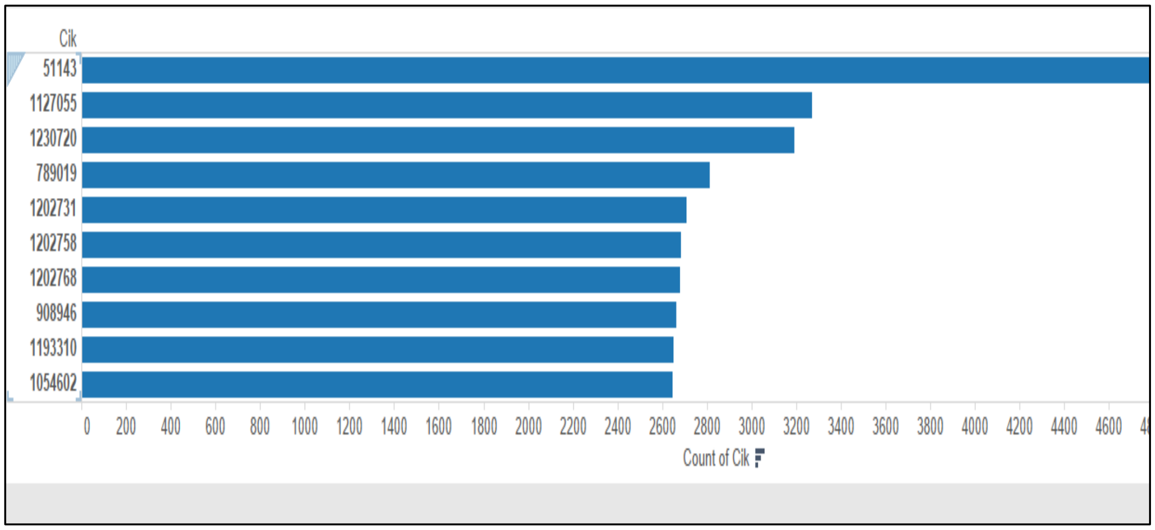
- **Count of IPs for every status code.**



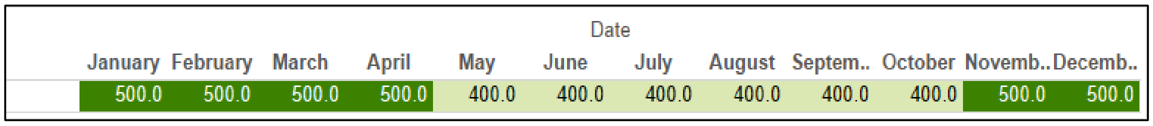
- **Peak hours of traffic during the day**



- **Top 10 companies that were accessed**

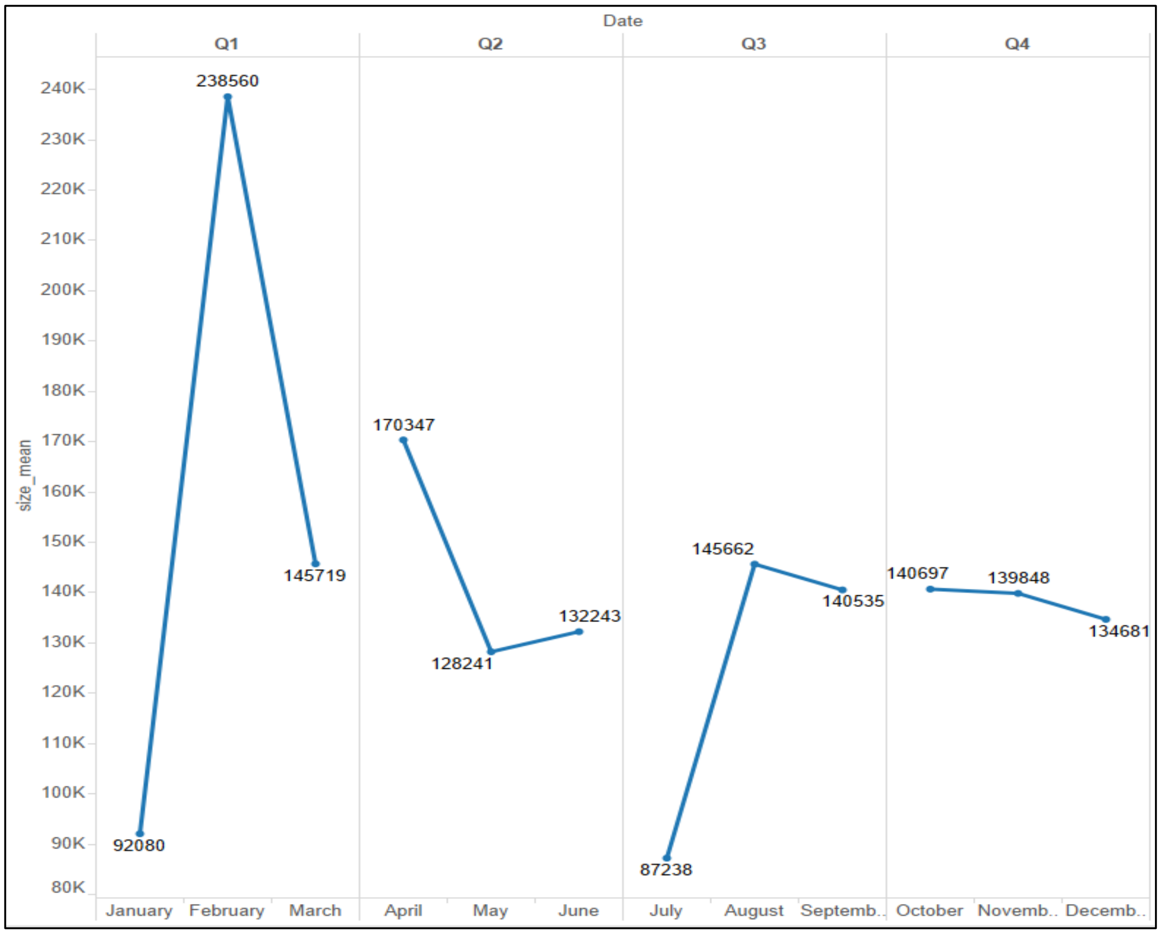


- **Maximum zone from where data was accessed**



Date											
January	February	March	April	May	June	July	August	Septem..	October	Novemb..	Decemb..
500.0	500.0	500.0	500.0	400.0	400.0	400.0	400.0	400.0	400.0	500.0	500.0

- **Average size of log file**


Date
Q1
Q2
Q3
Q4
size_mean
240K
230K
220K
210K
200K
190K
180K
170K
160K
150K
140K
130K
120K
110K
100K
90K
80K
January
February
March
April
May
June
July
August
Septemb..
October
Novemb..
Decemb..