

# Final Report

DSCI 551 Fall 2022

---

## Group 106

Edward Harianto, Erick Orozco, Saurabh Yadgire

*{ehariant, erickoro, yadgire}@usc.edu*

## I. Data Description

- Data selected are all from kaggle. As stated in the proposal, the theme is e-commerce. We selected 3 datasets describing electronics listed on Flipkart, an e-commerce giant based in India.

The 3 Datasets are:

- Laptop: <https://www.kaggle.com/datasets/lokareshrikant/flipkart-laptop-dataset>
- Mobile Phones: <https://www.kaggle.com/datasets/devsubhash/flipkart-mobiles-dataset>
- Televisions: <https://www.kaggle.com/datasets/lokareshrikant/flipkart-tv-dateset>

Attributes are general items such as brand, model, and selling price. There are specific attributes that only belong to a particular dataset as well with examples such as processor (laptop), storage (mobile phone) and resolution (television). Some cleaning had to be done for the datasets as well as splitting model and brand from product name for both Televisions and Laptops.

## II. EDFS Implementation

### Firebase

- **mkdir** : function to create a directory at the specified path. If directory exists already, then it wont be created. This has been handled in the code.
- **ls**: It shows the directories and files present at that location.
- **rm** : Removes the mentioned file at the location including all partitions.
- **cat**: returns the contents of the file (dataset) uploaded to the Fibase bucket.
- **put**: Upload the file to the bucket. In firebase, metadata is included to assist in other commands.

- **getPartitionLocations(file):** This method will return the locations of partitions of the file.
- **readPartition(file, partition#):** This method will return the content of partition # of the specified file. The portioned data will be needed in the second task for parallel processing.

## MySQL

All the above mentioned functions have been implemented using MySQL. PyMySQL library has been used to connect to the MySQL Database. A cursor is used which acts as an instance of the database. `cursor.execute()` is used to execute the query.

## III. Partition-Based Map and Reduce

There are 2 main functions: `mapPartition(p)` and `Reduce(*parts)` where `p` is a single partition and `*parts` is a list of outputs produced by `mapPartition()`. When first running the script, global variables are set via the inputs by the search and analytics entries from the website.

Global Variables:

- *file* - name of file to download from firebase and utilize throughout script
- *partitions* - number of partitions for the said file listed above
- *select* - list of features to select for output table
- *where* - Choose whether the search will have no conditions, a single condition, or be a range condition
- *feat* - a single feature to compare for the condition
- *single\_cd* - single condition type (`=`, `<`, `>`, `≤`, `≥`)
- *single\_val* - value for single condition
- *lo\_cd* - single condition type (`<`, `≤`) for lower limit of range
- *hi\_cd* - single condition type (`<`, `≤`) for higher limit of range
- *lo\_val* - value for lower limit of range
- *hi\_val* - value for higher limit of range
- *groupby* - choose None or single feature to groupby
- *atp* - choose type of analytic (None, Count, Average, Max, Min)

We also have a single function, `downloadPartitions(file, partitions)`, that downloads the partitions and formats them as pandas DataFrames. This creates a global variable called *filenames*, a list of these DataFrames.

Now that global variables are set, we now move on to the `main()` function. Here, we loop through all *files* from the *filenames* variable. In each loop we run the `mapPartitions()` function by passing in a *file* variable and appending the output to a *parts* list. In this `mapPartitions()` function quite a number of things happen. First, we check if *groupby* is not None and if *atp* equals 'Count'. If so, *select* now equates to *groupby*. This forces any 'Count' analytic to only have a single feature, for ease of use. It then checks if *groupby* is not None and not 'Count' and adds that feature to *select*. Now check if *where* is None, 'Range' or 'Single'. If not None, `rangeCondition(p)` or `singleCondition(p)` are called,

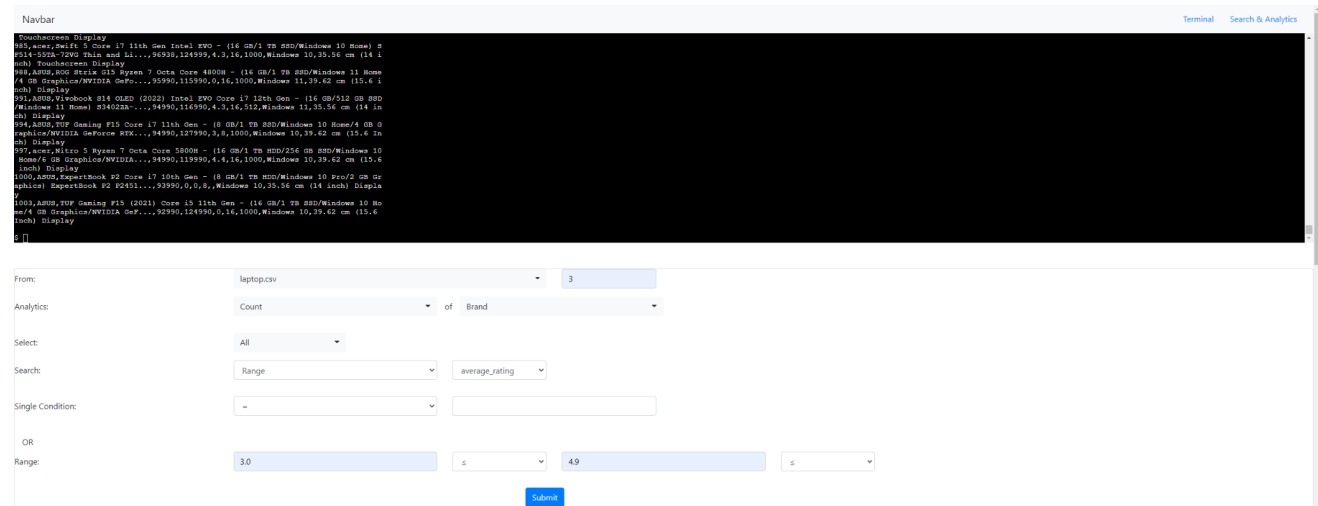
respectively. Based on the global variables, these functions will return the DataFrame with the set conditions in place. This is followed by checking what values *select* has and only pulling those features from the DataFrames. Next we drop any duplicate columns (this can happen when we add a feature from *groupby*). Finally, we call the *analyze(df)* function if *groupby* is not None. In this function, we analyze the approach given by *atp* and return the altered df.

Once all those *parts* are finished appending, we enter this list *parts* into our *Reduce()* function. Here we simply concatenate all DataFrames together and either sum the counts, average the averages, max the max, or min the min, based on the corresponding *atp* value. Reduce then outputs this now single DataFrame into the *data* variable.

We send *data* to the website. Lastly, we send the *parts* to the website as well, to help explain what the outputs of *mapPartition()* are.

## IV. Web User Interface

The Web User Interface was made using node.js that serves a html page. The page consists of two components which is an online terminal and a form for the Search & Analytics feature. The client talks to the server using a Websocket.



The screenshot displays a web application interface. At the top, there is a 'Navbar' and a 'Terminal Search & Analytics' header. The main content area is divided into two sections. The upper section is a terminal window with a black background and white text, displaying system information for various laptops, including details like 'Touchscreen Display', 'RAM', 'Storage', 'Processor', and 'Graphics'. The lower section is a search and analytics form. It includes a 'From' dropdown set to 'laptops.csv' with a value of '3'. The 'Analytics' section has a 'Count' dropdown, a 'of' dropdown set to 'Brand', and a 'Select' dropdown set to 'All'. The 'Search' section has a 'Range' dropdown and an 'average\_rating' dropdown. The 'Single Condition' section has a dropdown set to '=' and an empty input field. The 'OR' section has a 'Range' dropdown set to '3.0', a dropdown set to '<=', an input field set to '4.9', and a dropdown set to '<='. A 'Submit' button is located at the bottom right of the form.

The terminal is implemented using Xterm.js which is a frontend terminal for browsers. When a command is inputted into the terminal, the websocket sends that data to the server. In the server, we utilized Node.js child process to execute subprocesses. The parameters are sent as arguments to the firebase implementation of the EDFS and the output is then sent to the client to be displayed on the terminal.

The Search & Analytics feature is created using FormData objects which capture html information that can be processed in javascript. Similar to the terminal, when the user presses the submit button, an event handler sends the information to the server via a Websocket. In the server, a child process is spawned that executes the partition based map and reduce as seen in part 2 of the project. The results received is the output for

each partition, and the data. Once the client receives the information from the server, a html table is created by dynamically inserting the data received. The tables created are the overall map-reduce, and the map-reduce for each partition that was in the input.

The EDFS file structure can also be explored as long as the user knows the commands in part 1, allowing the user to explore the folder structure with *ls* and the contents of available files using *cat* and *readPartition*.

## **V. Scripts**

Below is the drive link containing all scripts and our datasets.

<https://drive.google.com/drive/u/4/folders/1qOVrScDPGwK7Guznf5CN1y0ZbwNr0stO>

## **VI. Video**

<https://youtu.be/j1PPcf-csLg>