Final Report - DSCI 551

Group: DSCI551-20223-Group-150
Course: DSCI 551 – Fall 2022
Project: Emulation-based system for distributed file storage and parallel computation
Team Member: Shreya Dwivedi (5148430435)

Introduction:
The project is an application based on an emulated distributed file system (EDFS). It will have functionalities such as creating a directory, listing files of a directory, retrieving data from a file or directory, deleting a file or directory. Since, the data is stored in a EDFS the data will be stored in different partitions in the database therefore to retrieve data we will implement map-reduce paradigm for the user to query the data.
The application will also be provided with an interface where the user can access the directory structure of the Cab dataset used also the user is provided a search functionality to get the transactions occurred between a range of date for a particular type of cab.

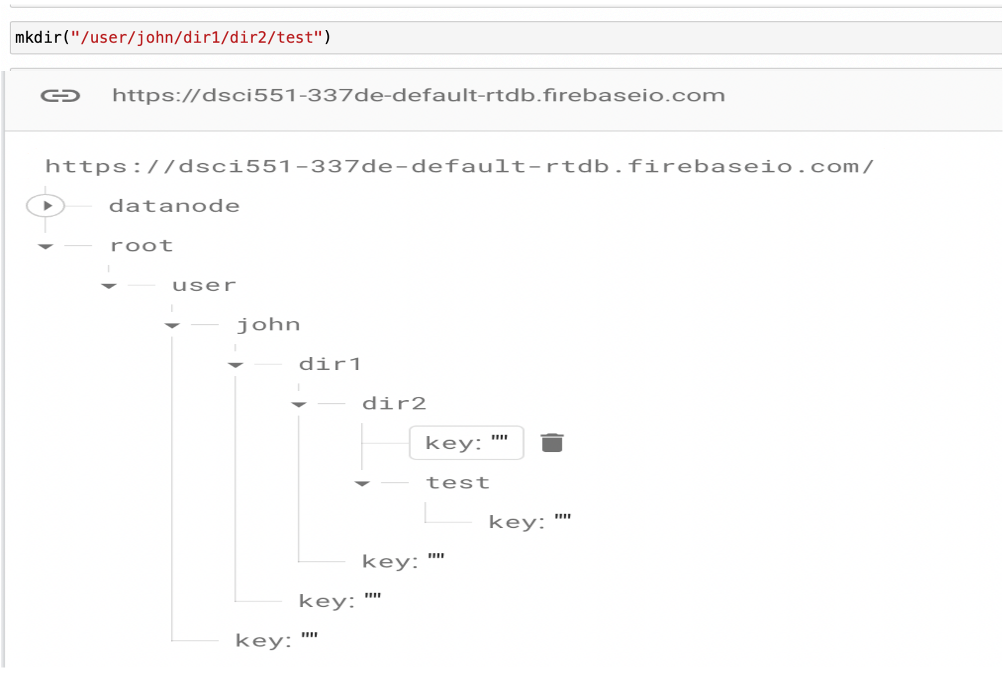
Dataset Used:
To implement this project, I selected two datasets: the Cab_data.csv and the Transaction_ID.csv which are available on Kaggle: https://www.kaggle.com/datasets/nishantdhingra/cabs-fare-data.
The datasets contain information about two cab companies: yellow cab and pink cab and the transaction data for rides taken by customers and the associated customer data across those 2 companies. The dataset was collected for an equity company to decide which cab company can be a better investment. The Velocity of the data is not high as the data is not time series data and each instance corresponds to a specific date. The Veracity of the data is low as there is no Metadata on the collection method of the data and whether it was born digital. The data is Value-adding theoretically speaking as it can be used to compare the performance of 2 different companies to enable the equity company to decide on which company to invest in, however since the Veracity of the data is low this undermines the added value of the data. It is worthy of note however that this is irrelevant to the purposes of this Project. There are no hidden elements in the data (i.e. no data silos) and the data is Open data that is freely accessible to the public, but the exact licensing of the data is not specified on Kaggle.

Technologies Used:

Name	**Use**
Python 3	Backend scripting language
Jupyter Notebook	Backend processing
Tkinter library	User Interface
Firebase	Database

Implementation Details:
This section provides the details about implementation of EDFS, MapReduce Search function as well as User interface. For the entire project the assumption is: input file type is .csv only. The project will have the following functionalities:
mkdir, cat, put, rm, ls, getPartitionLocations and readPartition.

1. **EDFS (Emulated Distributed File System):**
   Use Firebase-emulation system to implement the EDFS.
   JSON object will emulate the directory structure of the file system.
   For example:
   "root":{
   "user":{
   Myteam:{
   "file1.json":{ "p1": <location>, "p2": <location>}
   }
   }
   }
   In the above example we are storing file1 in the following directory structure:
   root>user>Mytem> file1.json
   p1: represents partition1
   p2: represents partition2
   <location>: These tags represent the URL of partitions of the file. The URL will be the location of the partitions of the file in different databases namely i.e., DataNode.
   Hence, we will be storing the metadata i.e., partition location, etc.(permission) in the NameNode and the actual data in the DataNode.
   - **mkdir:** This command is used to create a new directory at a specified location. We use this command to create a directory as a JSON object in our firebase database. The code snippet for the same is as shown below-



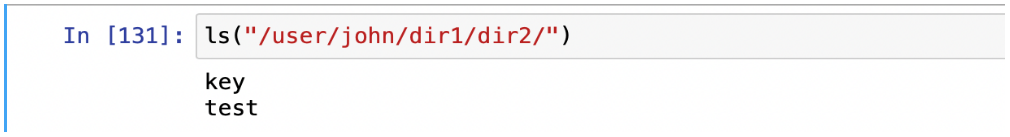
```
mkdir("/user/john/dir1/dir2/test")
```

https://dsci551-337de-default-rtdb.firebaseio.com

```
https://dsci551-337de-default-rtdb.firebaseio.com/
  datanode
  root
    user
      john
        dir1
          dir2
            key: ""
            test
              key: ""
          key: ""
        key: ""
      key: ""
```

The mkdir function takes the directory path as input and creates the directory at the path specified. The function will split the path on '/'. It will recursively create directories if it does not exist. For example- if the input path is 'user/john/dir1/dir2/test' and the directory 'user' does not have 'john' as directory inside it. The mkdir function will create a directory 'john' inside 'user' and then create 'dir1' directory under 'john'. This creation is a patch request to the firebase database.

- **ls:** This command is used to list the contents of the given directory. It will list all the files or directories that exist under the directory. We pass the directory path as input to the function and fetch all the files and folders in that path returns then to the user.

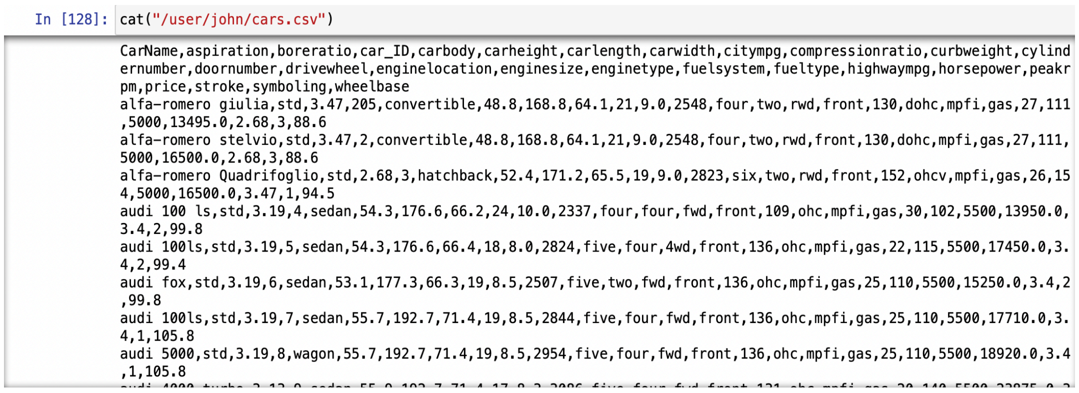


```
In [131]: ls("/user/john/dir1/dir2/")

          key
          test
```

Figure 1: ls()

The ls function takes directory path as input and outputs the list of files and folders in the directory specified by that path. We send a GET request to firebase for the input directory path and get the JSON object as a response. We then traverse it in the code and display it to the user in a directory structure format.

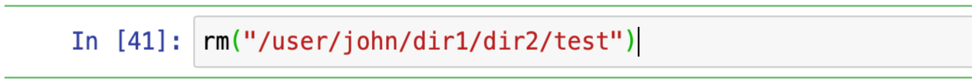
- **cat:** This command is used to display the contents of a particular file. A single file may be stored in multiple partitions. While displaying the output for this command we need to get the data from all the partitions and arrange it as the original file to output it to the user. Since our file format is restricted to csv only, we always display the output as csv.



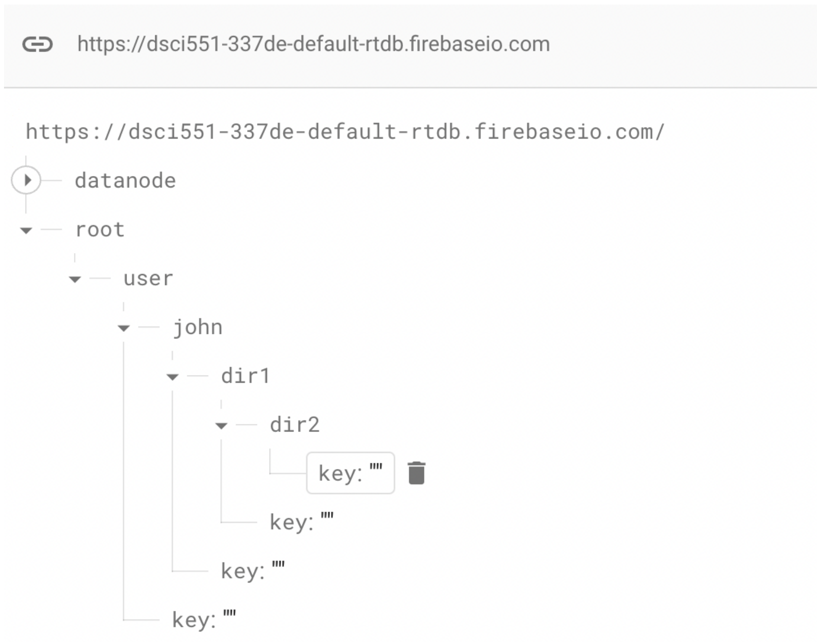
```
In [128]: cat("/user/john/cars.csv")

          CarName,aspiration,boreratio,car_ID,carbody,carheight,carlength,carwidth,citympg,compressionratio,curbweight,cylind
          ernumber,doornumber,drivewheel,enginelocation,enginesize,enginetype,fuelsystem,fueltype,highwaympg,horsepower,peakr
          pm,price,stroke,symboling,wheelbase
          alfa-romero giulia,std,3.47,205,convertible,48.8,168.8,64.1,21,9.0,2548,four,two,rwd,front,130,dohc,mpfi,gas,27,111
          ,5000,13495.0,2.68,3,88.6
          alfa-romero stelvio,std,3.47,2,convertible,48.8,168.8,64.1,21,9.0,2548,four,two,rwd,front,130,dohc,mpfi,gas,27,111,
          5000,16500.0,2.68,3,88.6
          alfa-romero Quadrifoglio,std,2.68,3,hatchback,52.4,171.2,65.5,19,9.0,2823,six,two,rwd,front,152,ohcv,mpfi,gas,26,15
          4,5000,16500.0,3.47,1,94.5
          audi 100 ls,std,3.19,4,sedan,54.3,176.6,66.2,24,10.0,2337,four,four,fwd,front,109,ohc,mpfi,gas,30,102,5500,13950.0,
          3.4,2,99.8
          audi 100ls,std,3.19,5,sedan,54.3,176.6,66.4,18,8.0,2824,five,four,4wd,front,136,ohc,mpfi,gas,22,115,5500,17450.0,3.
          4,2,99.4
          audi fox,std,3.19,6,sedan,53.1,177.3,66.3,19,8.5,2507,five,two,fwd,front,136,ohc,mpfi,gas,25,110,5500,15250.0,3.4,2
          ,99.8
          audi 100ls,std,3.19,7,sedan,55.7,192.7,71.4,19,8.5,2844,five,four,fwd,front,136,ohc,mpfi,gas,25,110,5500,17710.0,3.
          4,1,105.8
          audi 5000,std,3.19,8,wagon,55.7,192.7,71.4,19,8.5,2954,five,four,fwd,front,136,ohc,mpfi,gas,25,110,5500,18920.0,3.4
          ,1,105.8
```

Figure 2: cat()

- **rm:** This command removes a particular file or directory. When a directory is removed all its contents are removed too.



```
In [41]: rm("/user/john/dir1/dir2/test")
```

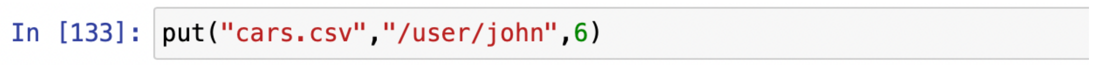


```
https://dsci551-337de-default-rtdb.firebaseio.com

https://dsci551-337de-default-rtdb.firebaseio.com/
  datanode
  root
    user
      john
        dir1
          dir2
            key: ""
          key: ""
        key: ""
      key: ""
```

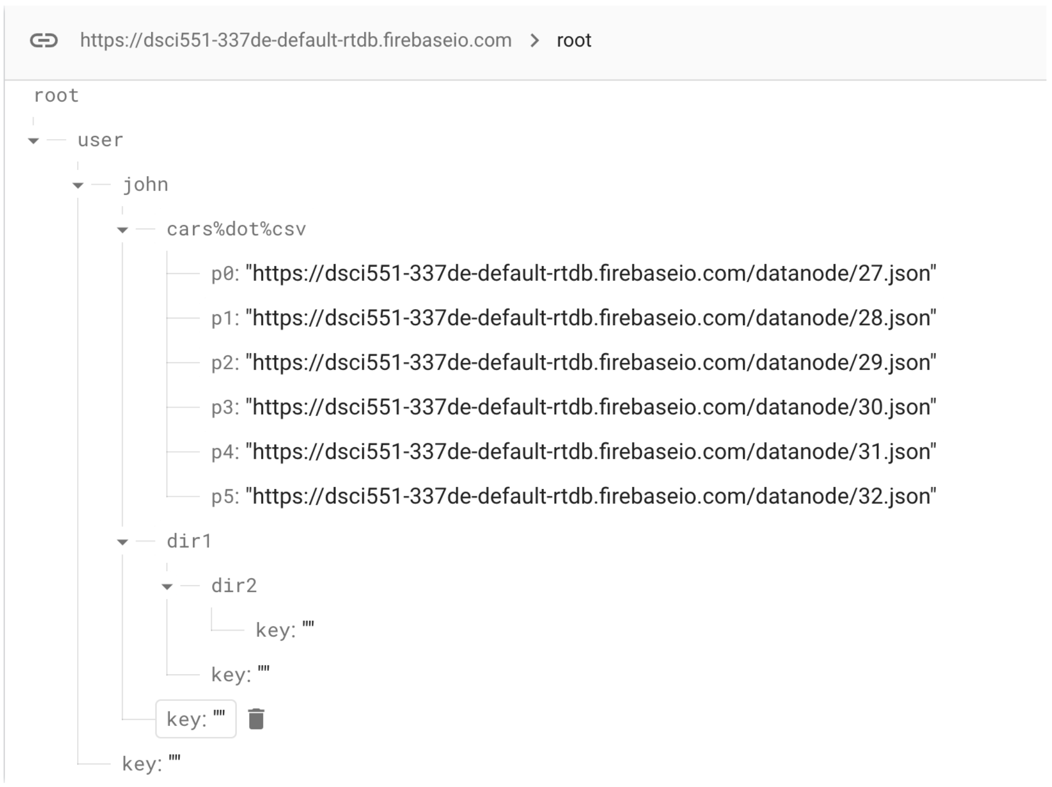
Figure 3: rm()

Referring to the figure 1 for mkdir we can see there was a test directory under the dir2 when run the command rm("/user/john/dir1/dir2/test") it removes the test directory from the database.

- **put:** This command puts the specified file into the database. The file is split into multiple partitions taken as input from user. Then it will be put in the database datanode. The metadata for the file is added to the namenode.



```
In [133]: put("cars.csv","/user/john",6)
```

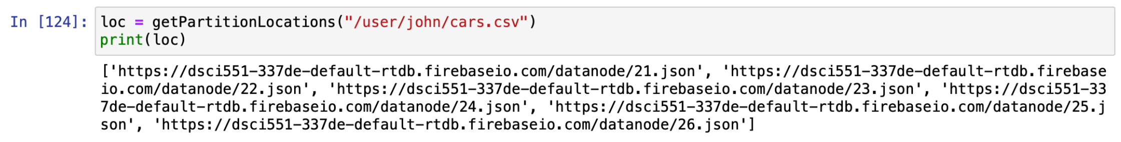


```
https://dsci551-337de-default-rtdb.firebaseio.com > root
root
▾ user
  ▾ john
    ▾ cars%dot%csv
        p0: "https://dsci551-337de-default-rtdb.firebaseio.com/datanode/27.json"
        p1: "https://dsci551-337de-default-rtdb.firebaseio.com/datanode/28.json"
        p2: "https://dsci551-337de-default-rtdb.firebaseio.com/datanode/29.json"
        p3: "https://dsci551-337de-default-rtdb.firebaseio.com/datanode/30.json"
        p4: "https://dsci551-337de-default-rtdb.firebaseio.com/datanode/31.json"
        p5: "https://dsci551-337de-default-rtdb.firebaseio.com/datanode/32.json"
    ▾ dir1
      ▾ dir2
          key: ""
        key: ""
      key: ""
    key: ""
```

Figure 4: put()

The input to put command is the location where the file we want to upload is located. If the file exists we divide the file into chunks. The size of each chunk is calculated based on the number of partitions that we want or the file. Thus, size of each chunk will be-(file size/number of partitions). Then each chunk is uploaded to the datanode.

- **getPartitionLocations():** This command is used to get the location of all the partitions of a particular file. Since, a file can be stored in multiple partitions, we will require to get all the locations while performing other functionalities such as read on the file.

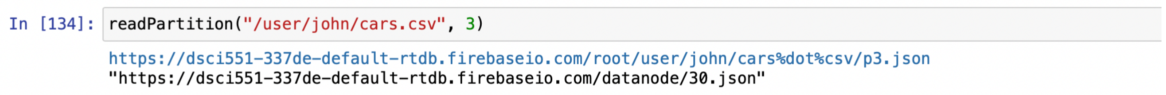


```
In [124]: loc = getPartitionLocations("/user/john/cars.csv")
          print(loc)

          ['https://dsci551-337de-default-rtdb.firebaseio.com/datanode/21.json', 'https://dsci551-337de-default-rtdb.firebase
          io.com/datanode/22.json', 'https://dsci551-337de-default-rtdb.firebaseio.com/datanode/23.json', 'https://dsci551-33
          7de-default-rtdb.firebaseio.com/datanode/24.json', 'https://dsci551-337de-default-rtdb.firebaseio.com/datanode/25.j
          son', 'https://dsci551-337de-default-rtdb.firebaseio.com/datanode/26.json']
```

Figure 5: getPartitions()

The partition locations are stored as metadata in the namenode of the database. Thus, we query the namenode to get all these locations. We take the input as the filename and GET request to the namenode database for the file and return all the URL locations stored for all the partitions.

- **readPartition():** This command is used to read the data from the specified partition of the file. To read a complete file we must read data from all the partitions the file is stored on. First, we will have to get the locations of all the partitions using the getPartitionLocations() function and then run the readPartition() on each of the partition.



```
In [134]: readPartition("/user/john/cars.csv", 3)

          https://dsci551-337de-default-rtdb.firebaseio.com/root/user/john/cars%dot%csv/p3.json
          "https://dsci551-337de-default-rtdb.firebaseio.com/datanode/30.json"
```

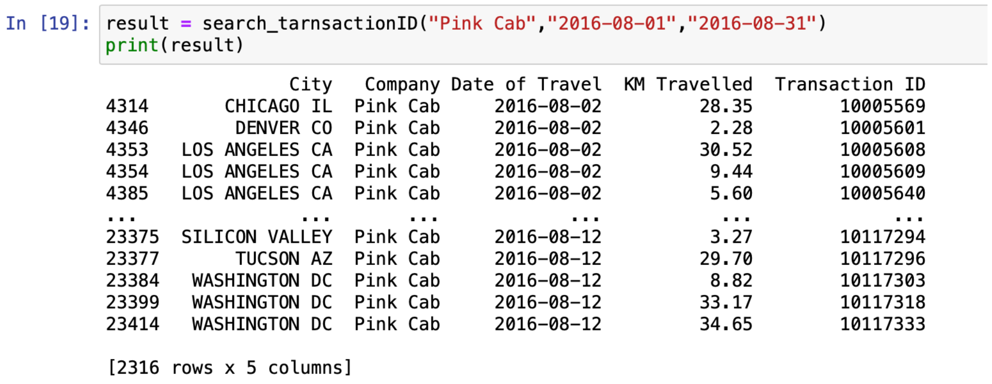
Figure 6: readPartition()

This command takes the partition location, partition number as input and outputs the data from datanode that belongs to that partition. We send a GET request to the NameNode database for the file and the partition number, get the datanode URL, then send a GET request to the datanode URL to fetch the partitioned data and return that to the user.

2. **Search Functionality:**
   This section focuses on the search functionality. Here is the description of the search functionalities implements in this project and how it is implemented with the help of MapReduce. The dataset we have used here is a cab dataset which has all the information about the transactions and customers, etc. for two cab companies.

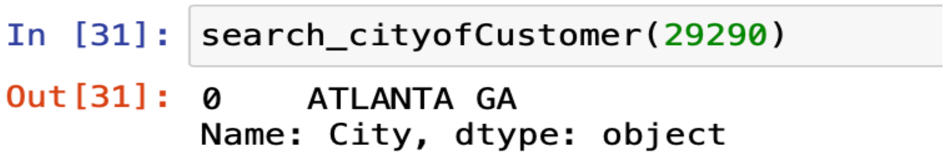
   The following search functionalities are implemented in the project:

- **List the transactions occurred between a range of dates for a particular cab company**: The search function takes the company name, from date and to date as arguments. It uses getPartitions method to access all the partitions of the file the mapPartition_getTransacitons() function is used to map the required data from the partitions i.e. the transactions that occurred between the range of dates. The reduce_getTransactions() function merges the searches of all the partitions and return them.



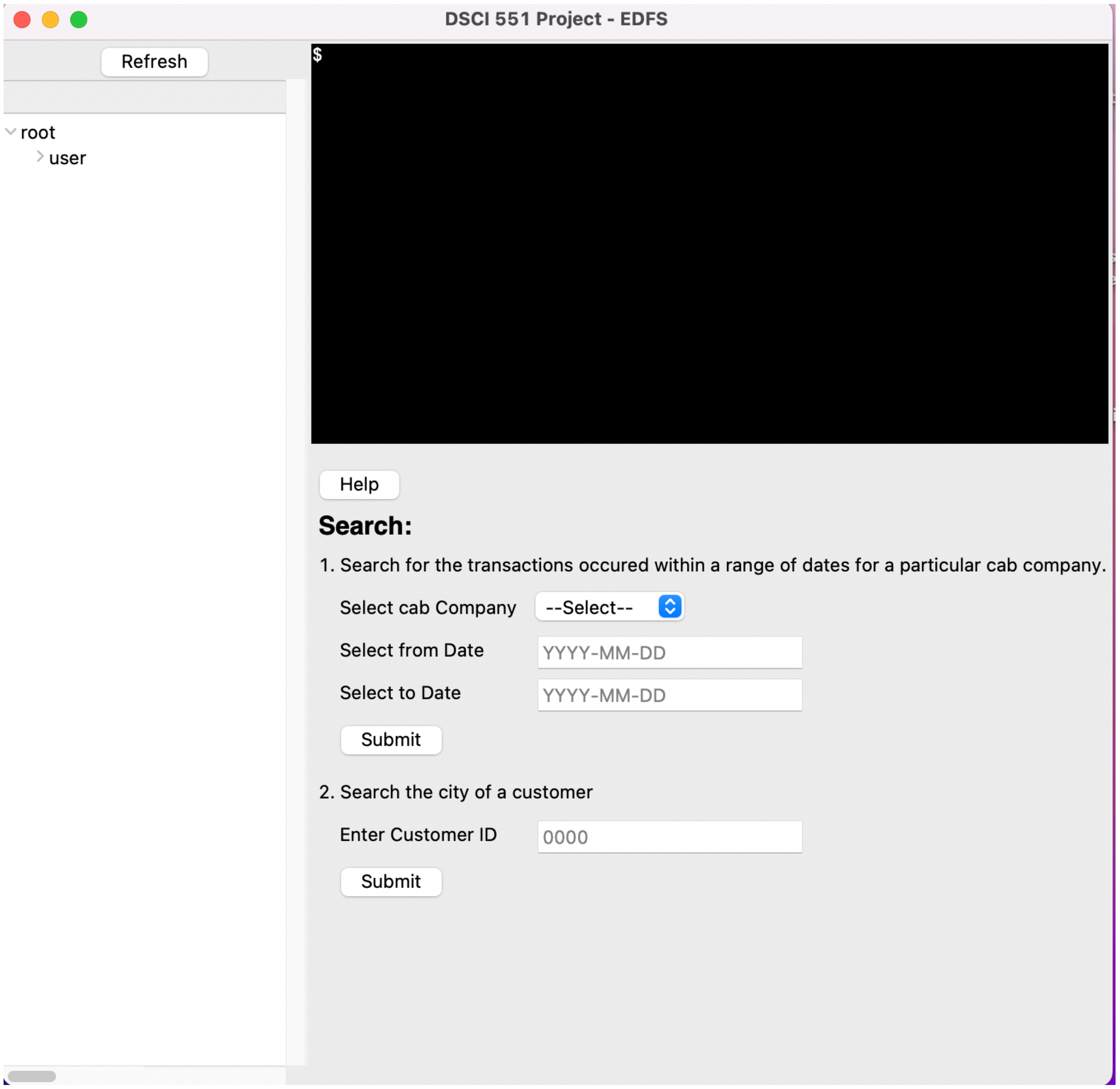
```
In [19]: result = search_tarnsactionID("Pink Cab","2016-08-01","2016-08-31")
         print(result)

                      City   Company Date of Travel  KM Travelled  Transaction ID
4314        CHICAGO IL  Pink Cab     2016-08-02         28.35        10005569
4346         DENVER CO  Pink Cab     2016-08-02          2.28        10005601
4353    LOS ANGELES CA  Pink Cab     2016-08-02         30.52        10005608
4354    LOS ANGELES CA  Pink Cab     2016-08-02          9.44        10005609
4385    LOS ANGELES CA  Pink Cab     2016-08-02          5.60        10005640
...              ...       ...            ...           ...             ...
23375   SILICON VALLEY  Pink Cab     2016-08-12          3.27        10117294
23377        TUCSON AZ  Pink Cab     2016-08-12         29.70        10117296
23384    WASHINGTON DC  Pink Cab     2016-08-12          8.82        10117303
23399    WASHINGTON DC  Pink Cab     2016-08-12         33.17        10117318
23414    WASHINGTON DC  Pink Cab     2016-08-12         34.65        10117333

[2316 rows x 5 columns]
```

- **Search the city of a particular customer:** The search function takes the customer ID as argument and return the city of that customer. The getPartitions() function retrieves all the partitions and the mapPartition_getCity() function maps the customer ID to the partitions and fetches the city where the customer took the cab ride. The reduce_getTransactions() function merges the searches of all the partitions and return them.



```
In [31]: search_cityofCustomer(29290)

Out[31]: 0    ATLANTA GA
         Name: City, dtype: object
```

3. **User Interface:** The user interface has been implemented using python tkinter, there are buttons for the user to search the cities where a particular customer took a cab and the transactions that occurred between a range of dates for a particular cab company. Also, there is a command line interface where the user can execute commands such as mkdir, ls, rm, etc. and there is a description for each of the commands for the ease of the user. The user can also access the directory structure that has been created in the database. There is a snippet of how the UI looks like:



DSCI 551 Project - EDFS

```
$
```

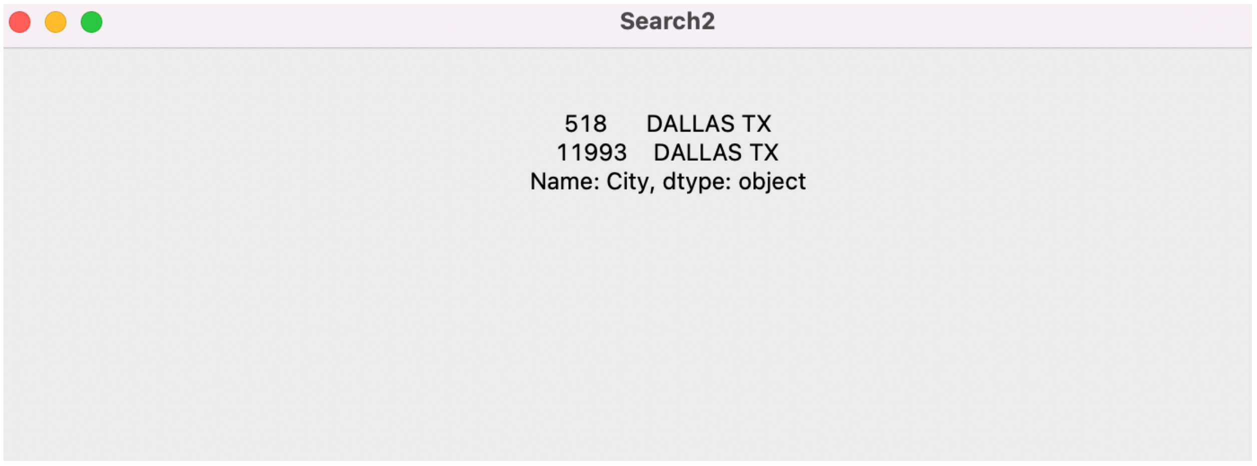
Search:

1. Search for the transactions occured within a range of dates for a particular cab company.

Select cab Company	--Select--
Select from Date	YYYY-MM-DD
Select to Date	YYYY-MM-DD

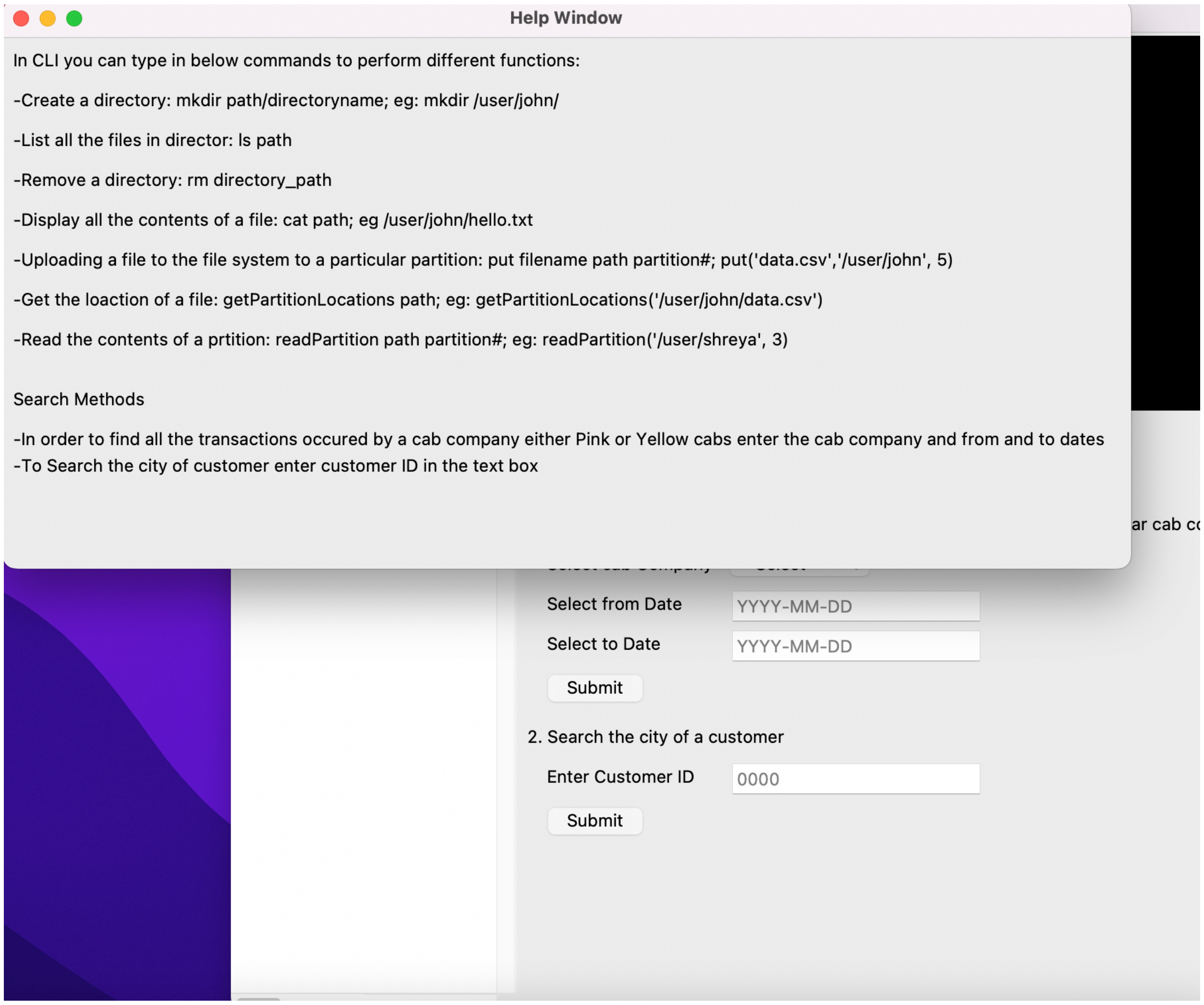
2. Search the city of a customer

Enter Customer ID	0000



Search2

```
518      DALLAS TX
11993    DALLAS TX
Name: City, dtype: object
```


Help Window

In CLI you can type in below commands to perform different functions:

-Create a directory: mkdir path/directoryname; eg: mkdir /user/john/

-List all the files in director: ls path

-Remove a directory: rm directory_path

-Display all the contents of a file: cat path; eg /user/john/hello.txt

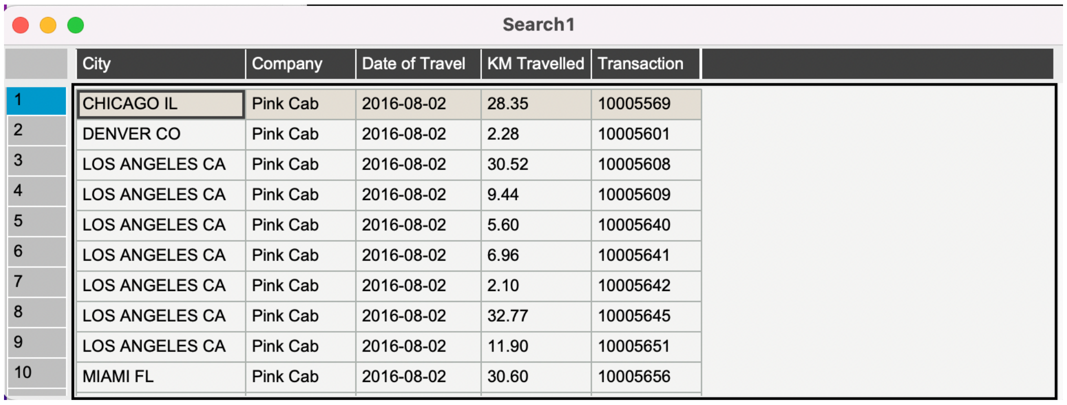
-Uploading a file to the file system to a particular partition: put filename path partition#; put('data.csv','/user/john', 5)

-Get the loaction of a file: getPartitionLocations path; eg: getPartitionLocations('/user/john/data.csv')

-Read the contents of a prtition: readPartition path partition#; eg: readPartition('/user/shreya', 3)

Search Methods

-In order to find all the transactions occured by a cab company either Pink or Yellow cabs enter the cab company and from and to dates
-To Search the city of customer enter customer ID in the text box


Search1

	City	Company	Date of Travel	KM Travelled	Transaction
1	CHICAGO IL	Pink Cab	2016-08-02	28.35	10005569
2	DENVER CO	Pink Cab	2016-08-02	2.28	10005601
3	LOS ANGELES CA	Pink Cab	2016-08-02	30.52	10005608
4	LOS ANGELES CA	Pink Cab	2016-08-02	9.44	10005609
5	LOS ANGELES CA	Pink Cab	2016-08-02	5.60	10005640
6	LOS ANGELES CA	Pink Cab	2016-08-02	6.96	10005641
7	LOS ANGELES CA	Pink Cab	2016-08-02	2.10	10005642
8	LOS ANGELES CA	Pink Cab	2016-08-02	32.77	10005645
9	LOS ANGELES CA	Pink Cab	2016-08-02	11.90	10005651
10	MIAMI FL	Pink Cab	2016-08-02	30.60	10005656

Link to the code:

Project(Google Drive):
https://drive.google.com/drive/folders/1tjyNYBrKSBtQdk4lAgLqA3pcu1ZU6yiC?usp=sharing

Video Link(Youtube Link): https://youtu.be/Rye0XTUt5Yg

Learning Experience

- Working on this project was a great learning experience for me I got a better understanding of how the partitioning works in HDFS and how is the data processed for multiple partitions using MapReduce.
- I implemented the search function which goes through all the partitions to find the required information and then uses reduce to give the combined result from all the partitions.
- I used Firebase for database where I stored the dataset, that is used to implement search functions also when the user creates the directory structure it will be created in the firebase itself. It gave me a better understanding of firebase-based emulation system.
- I also used python tkinter to implement the UI and hence I got to learn a new library in python tkinter and how it can be used to create desktop-based applications.