# Project Mid-Term Report

## “SMART CHOICE”

### Emulation-based file system for Distributed File Storage & Parallel Computation

**DSCI551-20223-Group- 128**

**Group Members:**


| Name | USC ID | Email id |
|---|---|---|
| Ankit Tripathi | 4612 6769 99 | ankittri@usc.edu |
| Lakshita Shetty | 2162 0892 11 | lshetty@usc.edu |
| Shreya Nayak | 8592 8104 56 | nayakshr@usc.edu |

## Aim:

Our goal is to develop a user-friendly online application that shows off emulated HDFS and employs the MapReduce function's partitioning structure to effectively search and analyze various data sets in the emulations.

## Description:

In today's digital age, where digital media consumption has a long-lasting impact on consumers' day-to-day behavior, it's critical for the audience to watch the right TV shows and movies based on their interests. The widespread availability of digital content is both boon and bane at the same time. Many Over-The-Top (OTT) services, such as Netflix, Amazon Prime Video, HBO Max, and others, strive to provide the best available content to their users in order to keep their memberships for extended periods of time. However, from the standpoint of the audience, it is extremely difficult to keep track of their favorite TV shows and movies. They may recall one or two movies that are available on a specific OTT platform but keeping track of a large number of TV shows and movies is more difficult. In a world when every second counts, it not only wastes customers' time to search every OTT platform for the content they want to see, but it also diminishes their desire to watch it. So, the intention of this project is to save the audiences those precious time they would otherwise spend for irrelevant actions.

The project is an emulation-based system for distributed file storage and capable parallel computation. It is a web-application that will provide audiences with a user-friendly interactive environment in which they can not only look for their favorite TV shows and movies, but also which OTT platforms they are available.

Our application ‘Smart Choice’ aims to cater to the varying needs of diverse users and to help them shortlist a movie or a show that they can relax or unwind to after a long tiring day without having to spend a lot of thought on which of the many options they should consider. The user needs to simply choose from the available options whether to check if a particular movie is on Netflix or not.

## Motivation:

In the present world where there are innumerable choices for everything. It becomes burdensome to make the right choice. There are too many things to choose from. This habitual phenomenon is because of something termed as the 'Choice paralysis'. Choice paralysis is the situation where an individual has trouble deciding owing to the massive amount of information and options available. Although having choices comes with its own set of advantages such as flexibility and freedom. It can also lead to a state where an individual does not make a choice at all. This paralysis is partly due to the belief that the individuals want to find the best possible choice such that it makes the most optimum use of their valuable time.

Research also shows that an excess of choices often leads people to be less, not more, satisfied once they actually make a decision. There's a frequent aching feeling that they could have done better. Understanding how to choose could guide people to make better decisions.
People have several criteria in mind when it comes to deciding which movies or TV shows they want to watch. These criteria include lead actor/actress i.e., the cast, the genre, the IMDB ratings, time-duration etc. Finding an option that satisfies most of their criteria is a good way of ensuring a suitable choice is made. This is the main inspiration for creating our project "Smart Choice"- which will enable the users to opt for an appropriate selection of their movies or tv shows.

## Dataset:

-> Dataset obtained from Kaggle:

This is a dataset that has been collected by a Kaggle user. The dataset lists various different movies & shows and their corresponding rating, genre and many other attributes.

Reference Link:

https://www.kaggle.com/datasets/victorsoeiro/netflix-tv-shows-and- movies

https://www.kaggle.com/datasets/victorsoeiro/amazon-prime-tv- shows-and-movies

https://www.kaggle.com/datasets/victorsoeiro/disney-tv-shows-and- movies

## Implementation Task 1:

Our goal according to the timeline is to complete working on the HDFS commands and implement it on one EDFS which is MySQL-based emulation in our case. The other EDFS is Firebase which has similar implementation. Created PMR queries to perform search and analytics on the dataset. Creating the main EDFS (with a simple and user friendly HTML front end) and decide the flow of the application. Created few Visualizations to understand the data better.

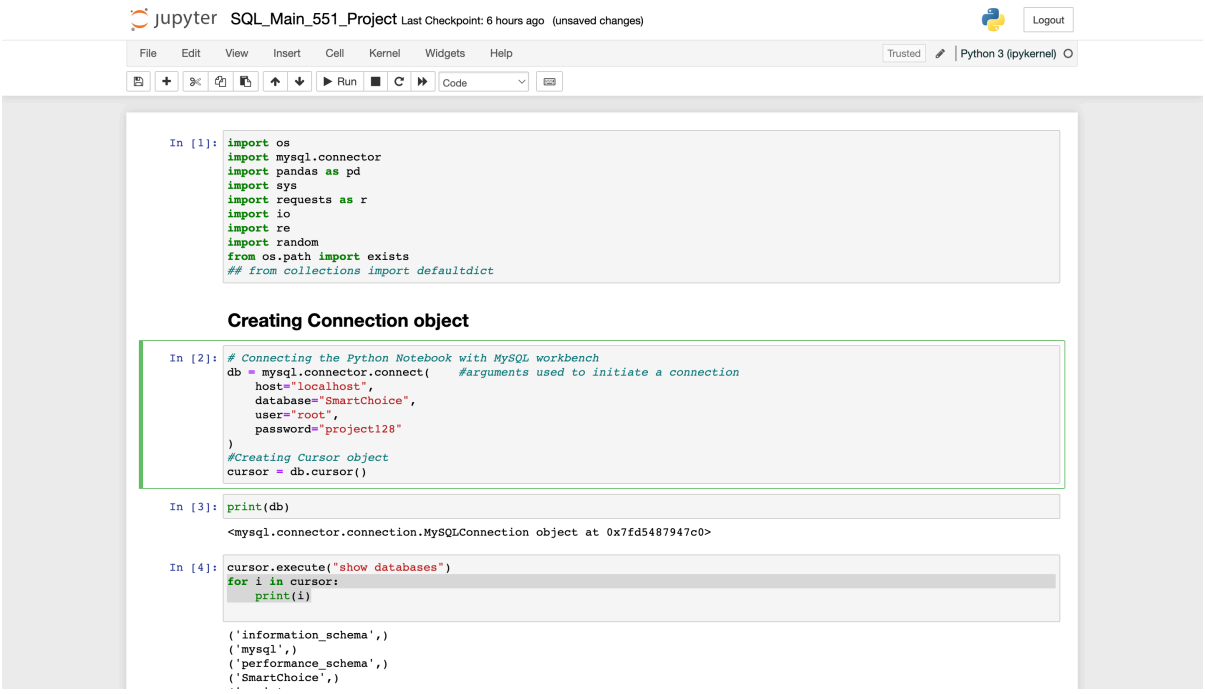
- We have implemented a total of 7 commands in order to emulate HDFS.

- We also created a table called namenode to store the meta data of the directories created and files uploaded. This table is an emulation of namenode in HDFS.

Our application 'Smart Choice' aims to cater to the varying needs of diverse users and to help them shortlist a movie or a show that they can relax or unwind to after a long tiring day without having to spend a lot of thought on which of the many options they should consider. The user needs to simply choose from the available options whether to check if a particular movie is on Netflix or not.

## Task 1 :

## A) MySQL-based emulation:




- Built an emulated distributed file system (EDFS)

Similar to HDFS, which stores metadata in namenodes and actual file data in datanodes, our EDFS would store metadata (i.e., file system structure, file attributes, position of file partition, etc.) in a database and partition data of our file in a second database.

Here Python is used with SQL. For the SQL implementation, a total of 4 tables are made:

```
mysql> desc namenode;
+-------+---------------+------+-----+---------+-------+
| Field | Type          | Null | Key | Default | Extra |
+-------+---------------+------+-----+---------+-------+
| id    | int           | NO   | PRI | NULL    |       |
| name  | varchar(100) | NO   |     | NULL    |       |
| type  | varchar(100) | NO   |     | NULL    |       |
| part  | varchar(100) | YES  |     | NULL    |       |
+-------+---------------+------+-----+---------+-------+
4 rows in set (0.01 sec)

mysql> desc part;
+----------------+---------------+------+-----+---------+-------+
| Field          | Type          | Null | Key | Default | Extra |
+----------------+---------------+------+-----+---------+-------+
| part_id        | int           | NO   | PRI | NULL    |       |
| firebase_link | varchar(200) | YES  |     | NULL    |       |
+----------------+---------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> desc Smart_choice;
+---------------------------+---------------+------+-----+---------+-------+
| Field                     | Type          | Null | Key | Default | Extra |
+---------------------------+---------------+------+-----+---------+-------+
| 'ID'                      | varchar(15)  | NO   |     | NULL    |       |
| 'Title'                   | varchar(100) | NO   |     | NULL    |       |
| 'Type'                    | varchar(10)  | NO   |     | NULL    |       |
| 'Description'             | varchar(100) | NO   |     | NULL    |       |
| 'Release_year'            | int           | NO   |     | NULL    |       |
| 'Genres'                  | varchar(300) | NO   |     | NULL    |       |
| 'Imdb_Score'              | float         | NO   |     | NULL    |       |
| 'Streaming_Platform'      | varchar(100) | NO   |     | NULL    |       |
+---------------------------+---------------+------+-----+---------+-------+
8 rows in set (0.00 sec)
```

**1. File system structure**

- *Primary Key —> parent, child*
- *Foreign Key —> parent and child*
- *Attributes —> parent, child*

In this there will be 2 columns parent and child and combined form the key.

**2. Namenode**

- *Primary Key —> Id*
- *Attributes —> Id, Path, Type*

In this there will be 3 columns:

- Id – Each item of meta data is given an id. For the table, it is set to AUTO INCREMENT and PRIMARY KEY.
- Path – Path of all the Directories and Files created in the EDFS
- Type – Type of the newly created file/directory.

**3. File partition**

- *Primary Key —> Id, p_ Id*
- *Foreign Key —> Id*
- *Attributes —> Id, p_ Id*

In this there will be 2 columns Id and p_Id and combined form the key.

**4. File partition Table**

In this there will be tables with respect to their partition.

## B) Firebase-based emulation:

### EDFS will use the following command:

1. **mkdir** – This function does two tasks with the user-supplied argument name. It does this by first making a new folder or directory. It will then incorporate meta information produced in the namenode.(a file system directory should be created)

```
def mkdir(directory):
    chck_path='/'.join(directory.split('/')[:-1])
    name=directory.split('/')[-1]
    print(name)
    data_values='{"'+directory.split('/')[-1]+'":""}'
    path = namenode+name+'.json'
    print(str(path))
    lst_nd=int(list(requests.get('https://project-dsci-551-default-rtdb.firebaseio.com/Namenode/.json?orderBy="id"&print=pretty&limitToLast=1').json().values())[0]['id'])
    requests.patch(namenode+name+'.json',data='{"id":'+str(lst_nd+1)+',"type":"DIRECTORY","path":"'+path+'"}').json()
    return "A New Folder '{}' has been created".format(name)
```

2. **ls** – This function merely displays all of the current folders and files. It outputs a list of paths to show the list of current files and folders and requires no input from the user.

```
def ls(directory):
    data_val=requests.get(namenode+directory+'.json').json()
    return list(data_val.keys())
```

3. **cat** – This function accepts a user-supplied file name and outputs the file's content. CSV files will display a data frame in the output, whereas txt files will just display all of the material, much like a nano.

   For example, if the user gives input as ‘Netflix-tites.csv’ as input it will show the data in a pandas data frame.

```
def cat(directory):

    val=requests.get(datanode+directory+'.json').json()
    return list(val.values())
```

4. **rm** – The directory or file that the user specifies is simply deleted by this procedure. It requests the file name from the user as an argument and deletes the file from the namenode and datanode tables.

5. **put** – This function will upload a file to file system, will upload a file csv file to the directory in EDFS. It will then update 2 tables. Firstly, it will create an entry in the namenode and then it will store the data in the datanode table.The file will be storing in k partitions, and the file system should remember where the partitions are stored.

```
def put(filename,k):
    if filename.split('.')[-1]=='csv' or filename.split('.')[-1]=='json':
        file=filename.split('.')[0]
    df=pd.read_csv(filename)
    size=int(len(df)/k)
    # print(df)
    # print(size)
    col_list=df.columns
    for i in range(0,k):
        df1=df.iloc[(i)*size:(i+1)*size]
        list_of_jsons= df1.to_dict(orient='records')
        fnl_list=list_of_jsons

        #URL="https://project-dsci-551-default-rtdb.firebaseio.com/Namenode/.json"

        print(fnl_list)
        #Make a dict and directly pass as the data-->json
        data_values=fnl_list
        requests.put(url = 'https://project-dsci-551-default-rtdb.firebaseio.com/Datanode/'+file+'/part'+str(i+1)+'/.json', json = data_values)

    path_datanode=datanode+file+'.json'
    lst_nd=int(list(requests.get('https://project-dsci-551-default-rtdb.firebaseio.com/Namenode/.json?orderBy="id"&print=pretty&limitToLast=1').json().values())[0]['id'])
    requests.patch(namenode+file+'.json',data='{"id":'+str(lst_nd+1)+',"type":"File","path":"'+path_datanode+'"}').json()
    return "A New Folder '{}' has been created".format(file)
```

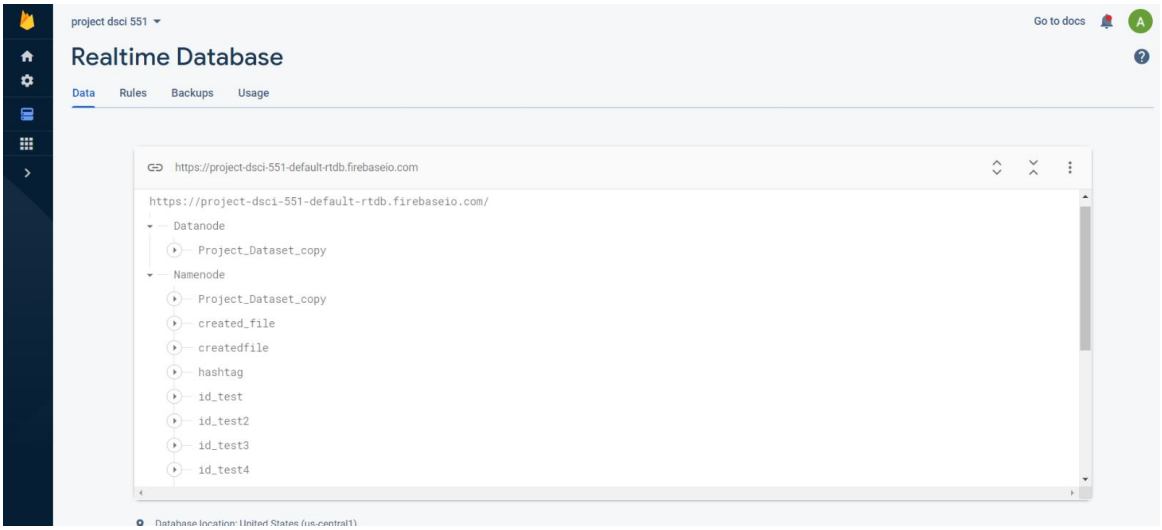
6. **getPartitionLoc** – This function will return the locations of partitions of the file that is created by the EDFS.

```
def getPartitionLoc(directory):
    val=requests.get(datanode+directory+'.json').json()
    val_namenode=requests.get(namenode+directory+'.json').json()
    id_value=val_namenode['id']
    x=list(val.keys())
    dict={"id_value":id_value,"parts":x}
    return dict
```

7. **readPartition** – This function will return the content of partition # of the specified file. The portioned data will be needed in the second task for parallel processing.

```
def readPartitionLocations(directory,part_no):
    k=int(part_no)
    data_output=requests.get(datanode+directory+'/part'+str(k)+'.json').json()
    return data_output
```

**Firebase:**



## Task 2 :
## Implement partition-based MapReduce (PMR) for search/analytics

We will implement our mapPartition function as described in the guidelines (Map() function would process every partition and return the desired list. Reduce() function would return an aggregate of outputs from Map())

Examples of search and analytics queries:

We will implement Search queries and Analytics queries on the data sets mentioned above -

for example, searching the movies within a given rating or from a particular genre, comparative study across the 3 platforms etc.

Based on the user choice, our flask framework would take a partition number and use the MapReduce function to send requests/queries to the database and receive a final output for every user operation.

We aim to enable the user and carry out a basic exploratory data search/analysis on the chosen dataset.

## PMR:

The SQL queries that will be used for PMR are listed below. On either the entire dataset or a partition of it, all queries can run and execute. The user will indicate whether they want to run the task on the entire dataset or only a certain segment.

- Searching and returning a list of titles of movies/show who have IMDB Rating greater than user input

  for example:
  user input = 8
  SQL query running at the backend:

  select title
  from Netflix-titles Where tmdb_score>8

- Searching and returning a list of titles of movies/show who have genres by the user

  for example:
  user input = ['drama', 'crime']
  SQL query running at the backend:

  select title
  from Netflix-titles contains genres like ‘drama' and 'crime'

- Searching and returning a list of titles of movies from who have IMDB Rating 8 and genres by the user.

  for example:
  user input = ['drama', 'crime']

SQL query running at the backend:

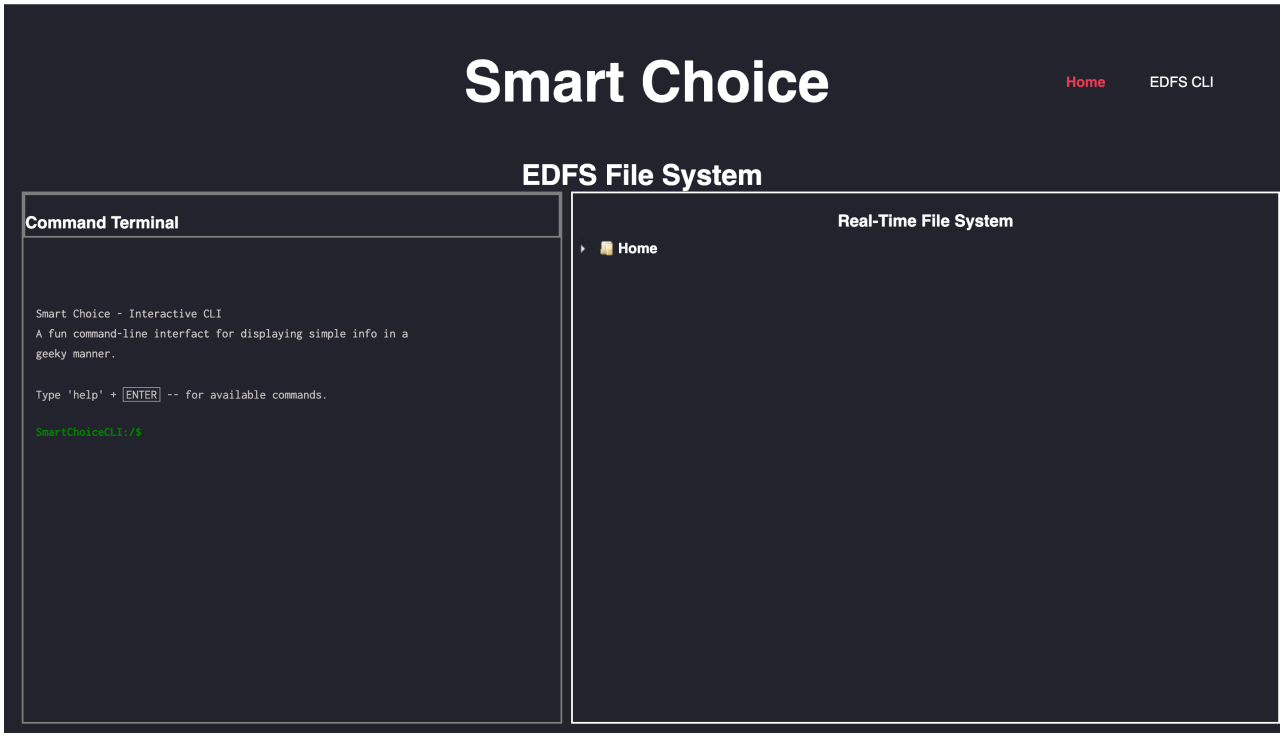
```
select title
from Netflix-titles\movies Where mdb_score>8 Group by genres
```

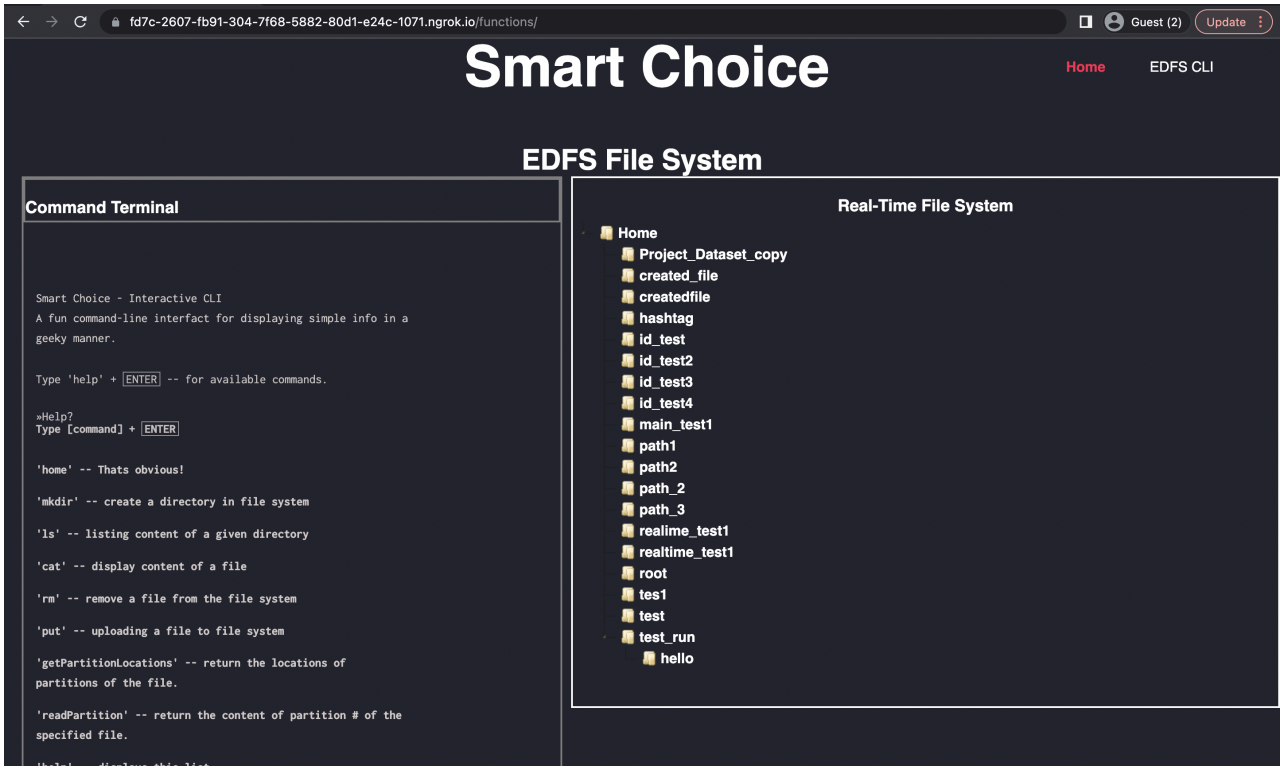
We can implement our mapPartition function as described in the guidelines (Map () function would process every partition and return the desired list. Reduce() function would return an aggregate of outputs from Map())
Examples of search and analytics queries:

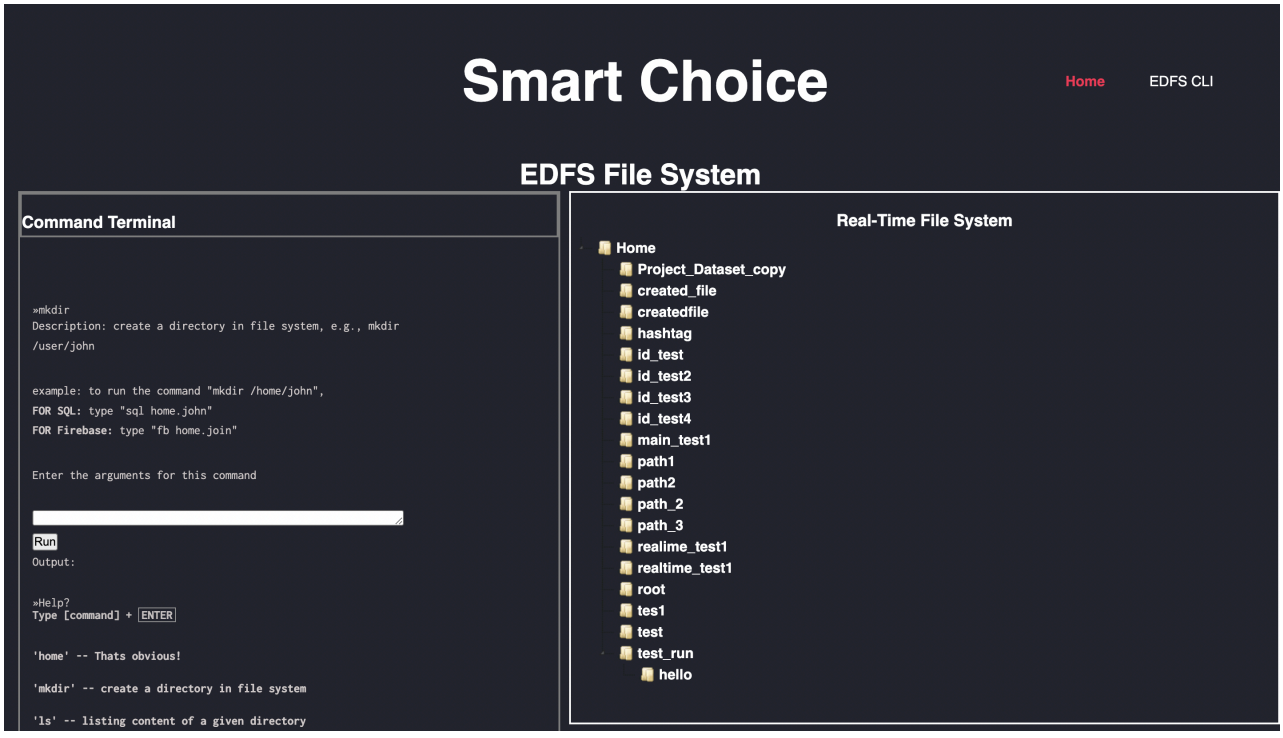
We will implement Search queries and Analytics queries on the data sets mentioned above - for example, searching the movies within a given rating or from a particular genre, comparative study across the 3 platforms etc.

Based on the user choice, our flask framework would take a partition number and use the MapReduce function to send requests/queries to the database and receive a final output for every user operation.

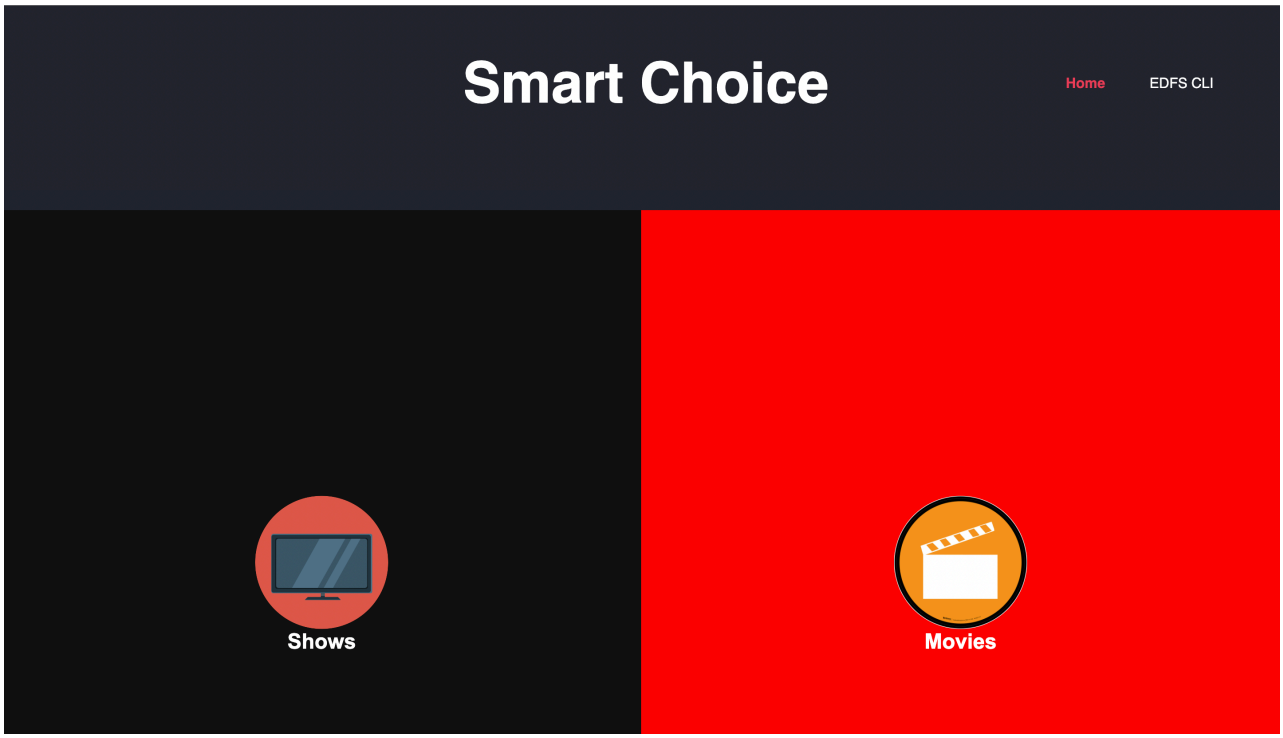
We aim to enable the user and carry out a basic exploratory data search/analysis on the chosen dataset.

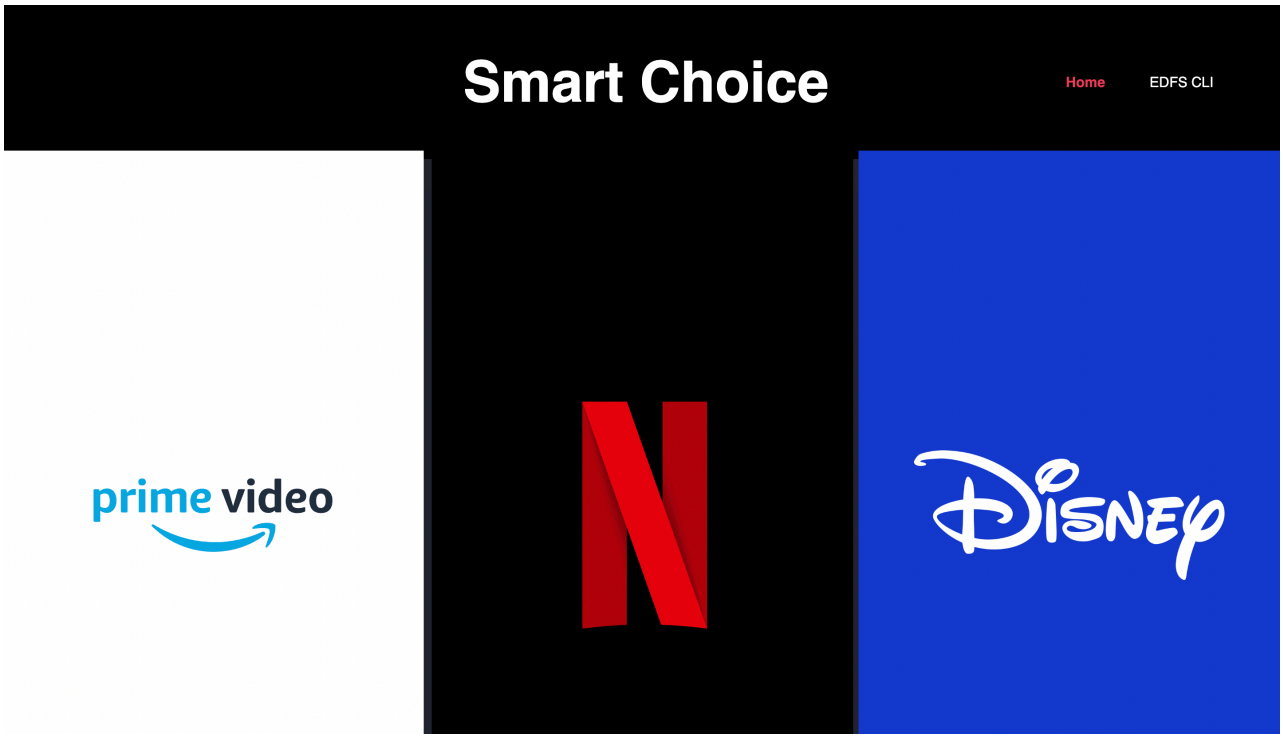



fd7c-2607-fb91-304-7f68-5882-80d1-e24c-1071.ngrok.io/functions/
Guest (2)
Update
Smart Choice
Home
EDFS CLI
EDFS File System
Command Terminal
Smart Choice - Interactive CLI
A fun command-line interfact for displaying simple info in a geeky manner.
Type 'help' + ENTER -- for available commands.
»Help?
Type [command] + ENTER
'home' -- Thats obvious!
'mkdir' -- create a directory in file system
'ls' -- listing content of a given directory
'cat' -- display content of a file
'rm' -- remove a file from the file system
'put' -- uploading a file to file system
'getPartitionLocations' -- return the locations of partitions of the file.
'readPartition' -- return the content of partition # of the specified file.
Real-Time File System
Home
Project_Dataset_copy
created_file
createdfile
hashtag
id_test
id_test2
id_test3
id_test4
main_test1
path1
path2
path_2
path_3
realime_test1
realtime_test1
root
tes1
test
test_run
hello


Smart Choice
Home
EDFS CLI
EDFS File System
Command Terminal
»mkdir
Description: create a directory in file system, e.g., mkdir /user/john
example: to run the command "mkdir /home/john",
FOR SQL: type "sql home.john"
FOR Firebase: type "fb home.join"
Enter the arguments for this command
Run
Output:
»Help?
Type [command] + ENTER
'home' -- Thats obvious!
'mkdir' -- create a directory in file system
'ls' -- listing content of a given directory
Real-Time File System
Home
Project_Dataset_copy
created_file
createdfile
hashtag
id_test
id_test2
id_test3
id_test4
main_test1
path1
path2
path_2
path_3
realime_test1
realtime_test1
root
tes1
test
test_run
hello

**Solving MapReduce query:**



# Smart Choice

Home EDFS CLI

**Filters**

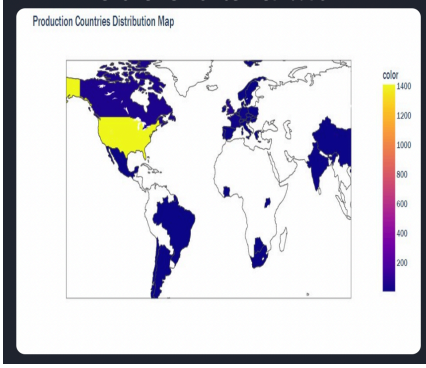
All Filters | On Ratings | On Genres | On Release Year

## DATA TABLE: MOVIES ON DISNEY

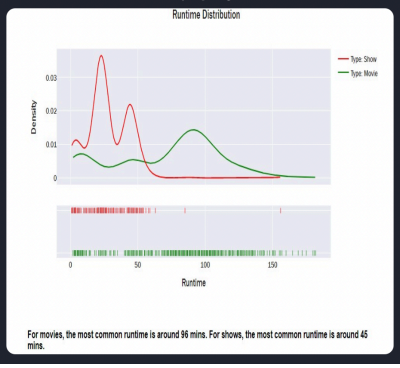
| Title | Description | Genres | Imdb_Score | Release_year |
|---|---|---|---|---|
| Fantasia | Walt Disney's timeless masterpiece is an extravaganza of sight and sound! See the music come to life... | animation, family, music, fantasy | 7.7 | 1940 |
| Snow White | A beautiful girl, Snow White, takes refuge in the forest in the | fantasy, family, romance | | |
| The Adventures of Ichabod and Mr. Toad | Willows. Concise version of Kenneth Grahame's story of the same name. J. Thaddeus To... | fantasy, horror, animation, comedy, family | 6.9 | 1949 |
| | Cinderella has faith her dreams of a | fantasy, animation | | |

## Analysis of Disney data

### Shows VS Movies Distribution

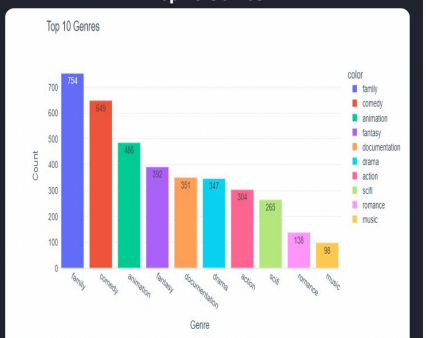



### Runtime




For movies, the most common runtime is around 96 mins. For shows, the most common runtime is around 45 mins.

### Top 10 Genres



## Task 3 :
## Creating an app that uses the implemented PMR methods.

The web application will be a user-friendly interface that the user can use to achieve the above-mentioned goals.

The application will allow users to select from any of the databases listed via the input/choose box. They can then choose their interest-specific options from a wide range of selection criteria, which can subsequently be analyzed/searched within the database of their choice.

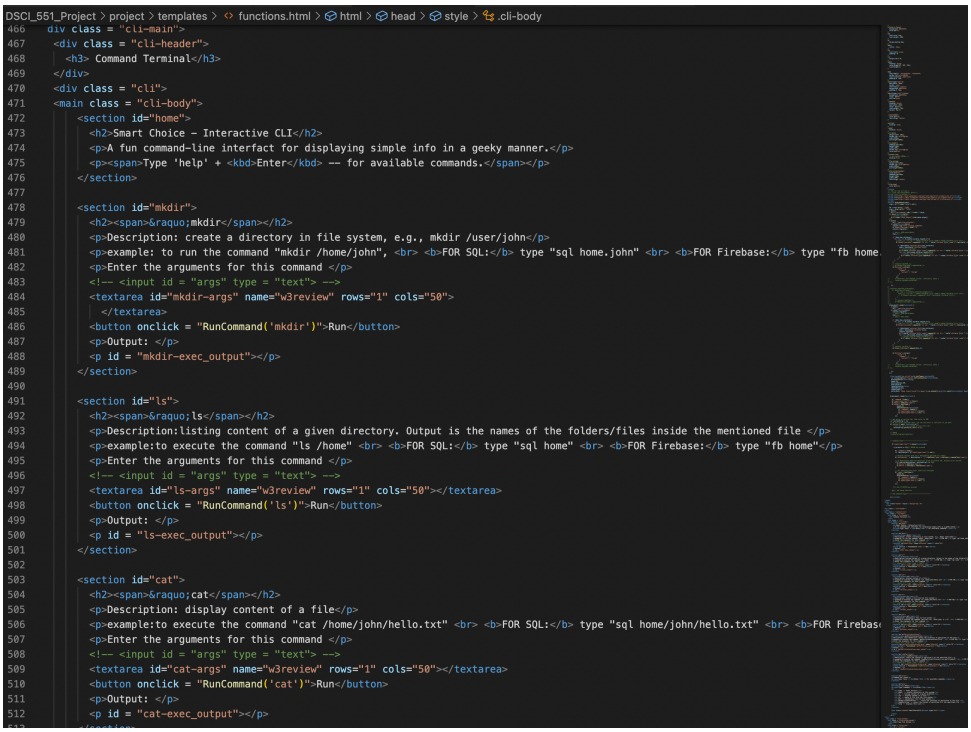
The Frontend of the application will be designed using HTML5/CSS/Javascript and will be integrated with the Django framework of Python. The application will also be connected to both the databases Firebase and MySQL.

The input/choose boxes will provide users with selection options for whose inputs will be utilized to perform CRUD (Create, Retrieve, Update, and Delete) operations like those performed in HDFS.

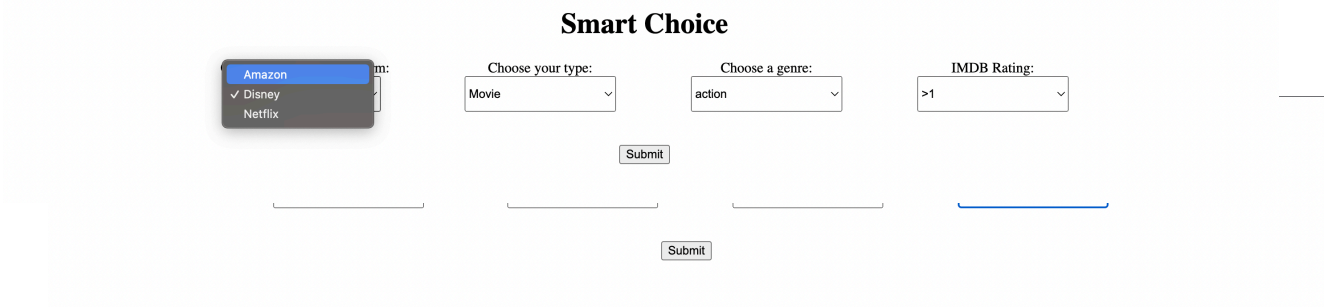
The user will first select the database to which they want to gain access, and then they will be able to execute the following functions:

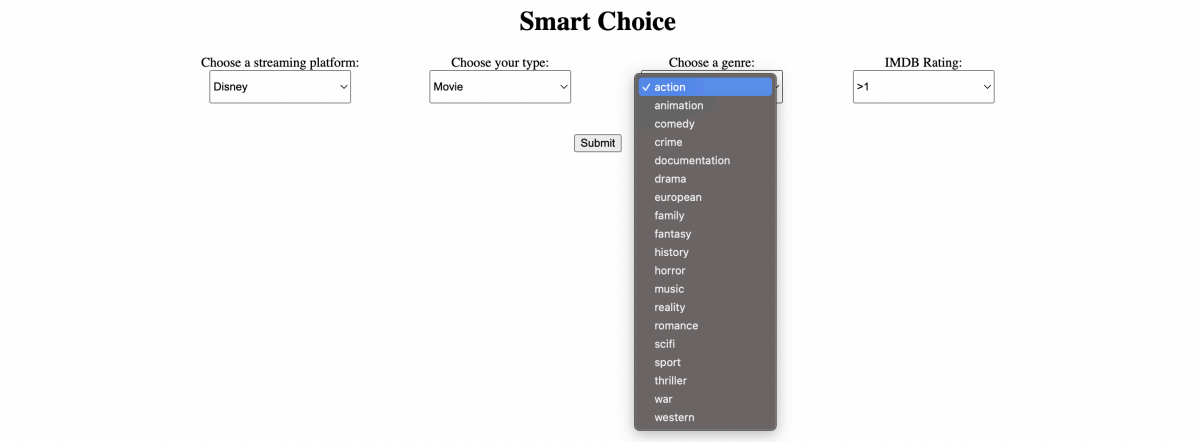
1. Filter movies and TV Shows based on IMDB ratings.
2. Filter the content by Genre (Action, Thriller, Comedy, Drama)
3. The OTT platform where the corresponding Tv show, Movie is available.

Our initial plan is to take the user inputs from the HTML web page and use Django to send appropriate queries to the backend databases to perform the necessary CRUD operations and the functions mentioned above.

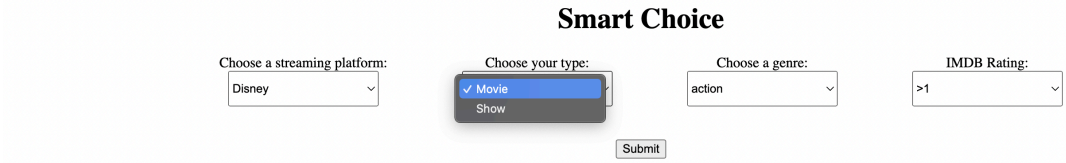


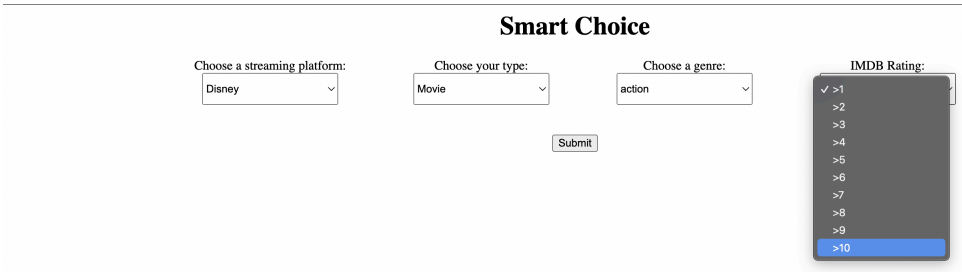
## Old Website Framework

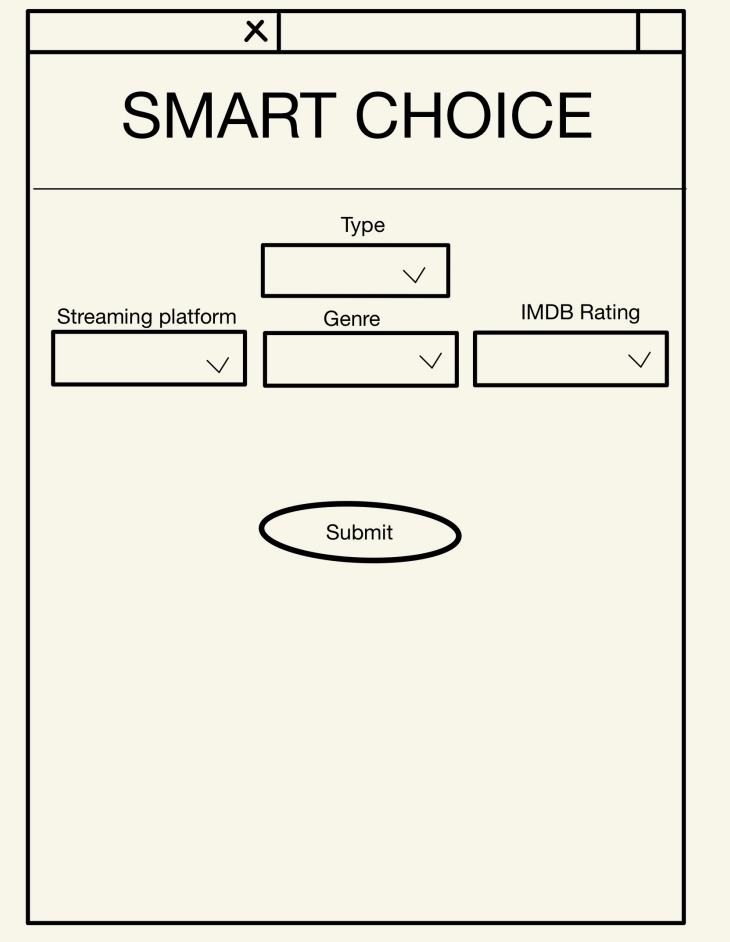


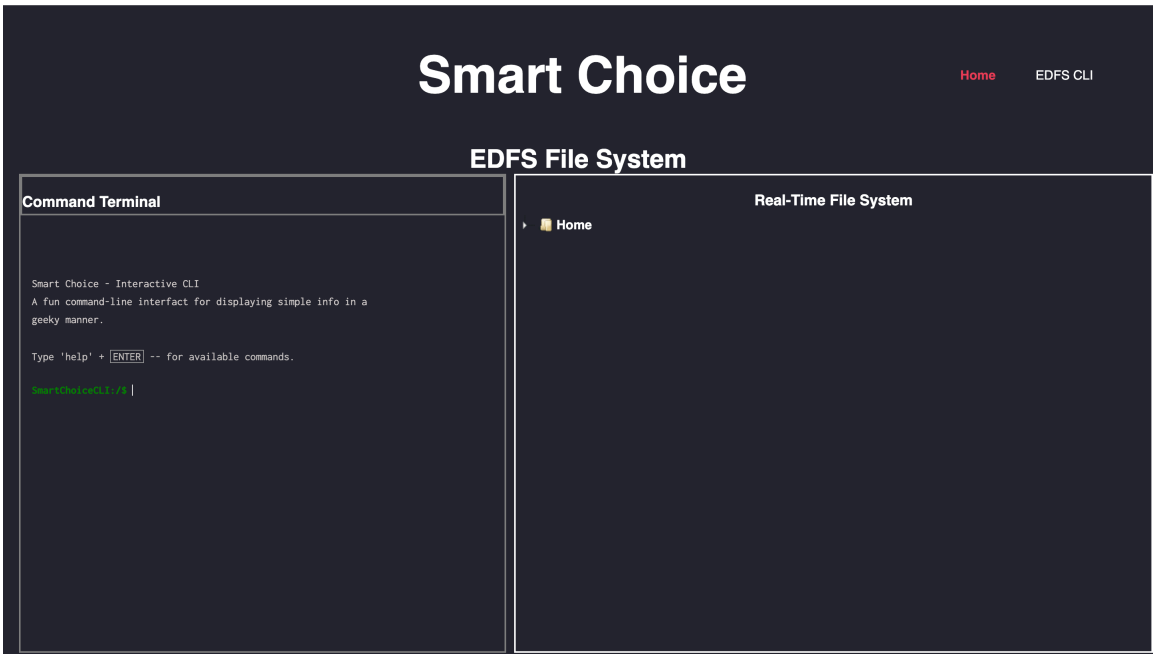

# Code Snippets of HTML



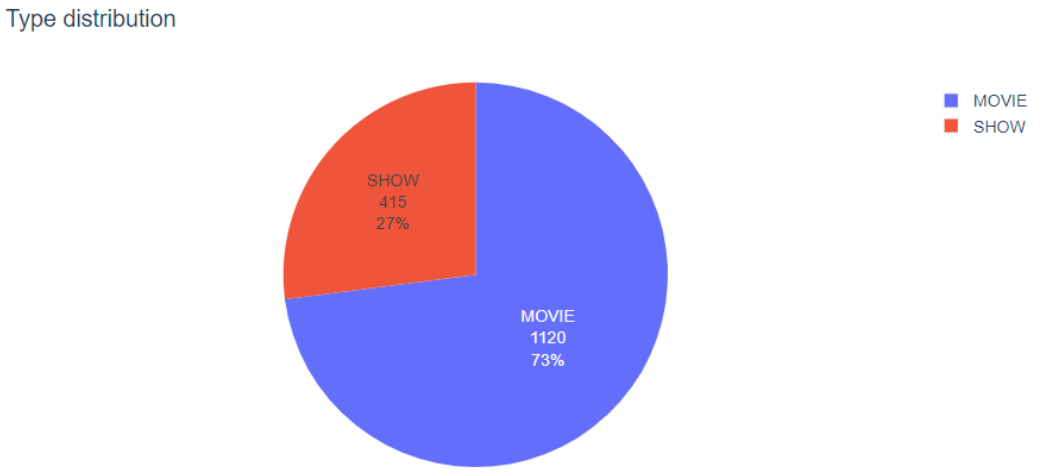



## Latest Implementation CLI

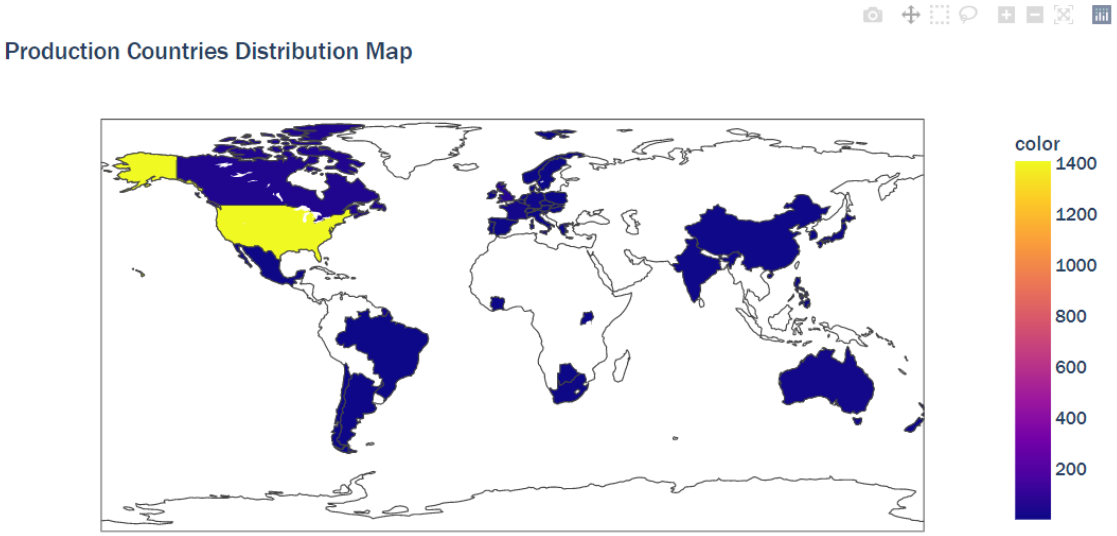


# Data Visualization:

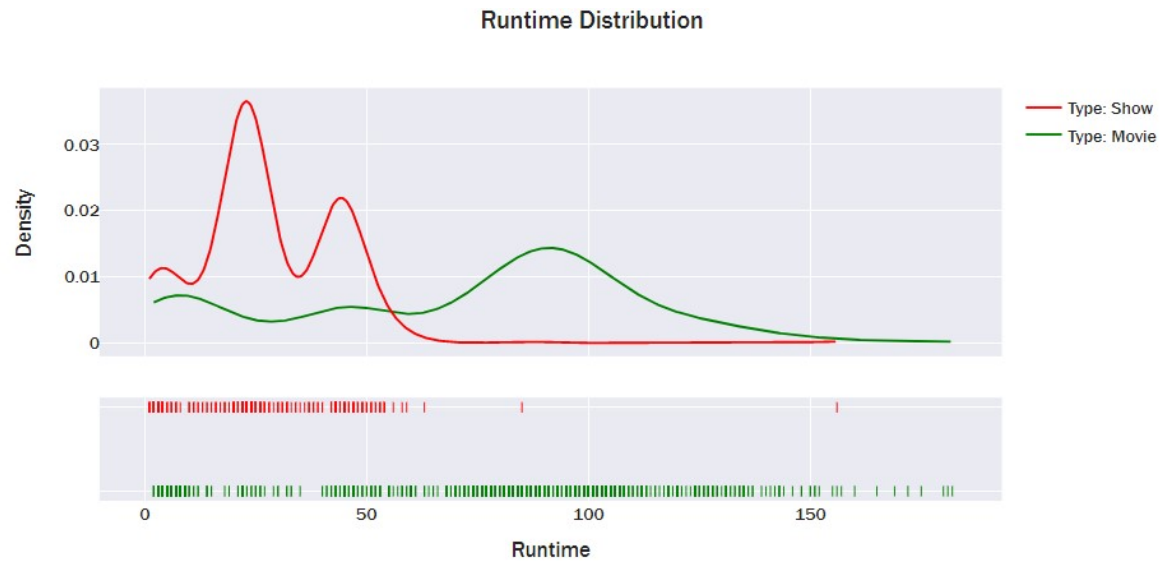
```
In [18]: type_count = df_disney["type"].value_counts()

type_fig = px.pie(values=type_count.values,names=type_count.index,template="plotly_white",title="Type distribution")
type_fig.update_traces(textinfo='label+percent+value')
type_fig.update_layout(font = dict(size= 15, family="Arial"))
type_fig.show()
```



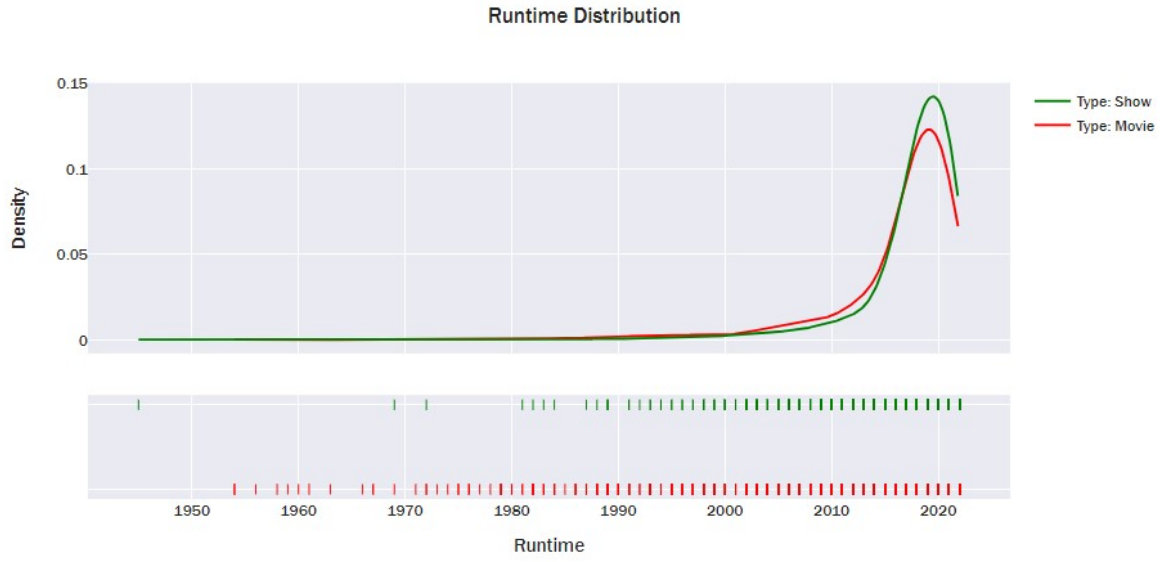

**From the piechart it can be seen that 36% data is Show and 64% data is Movie.**



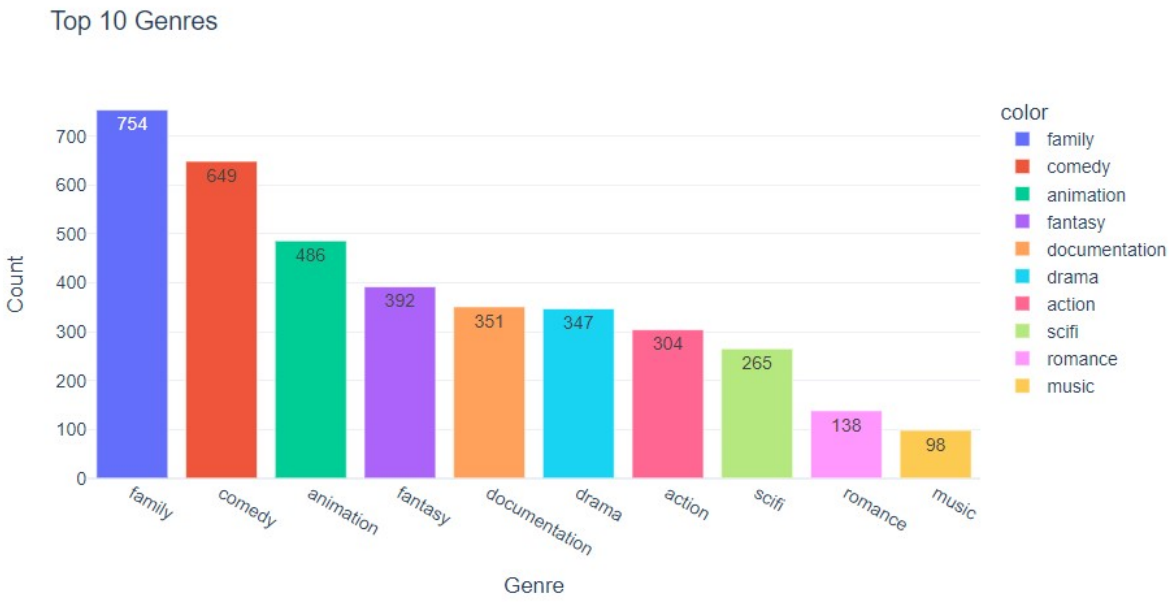

**United States has the yellow color, which means, for this dataset, the United States produces the most of shows and movies.**

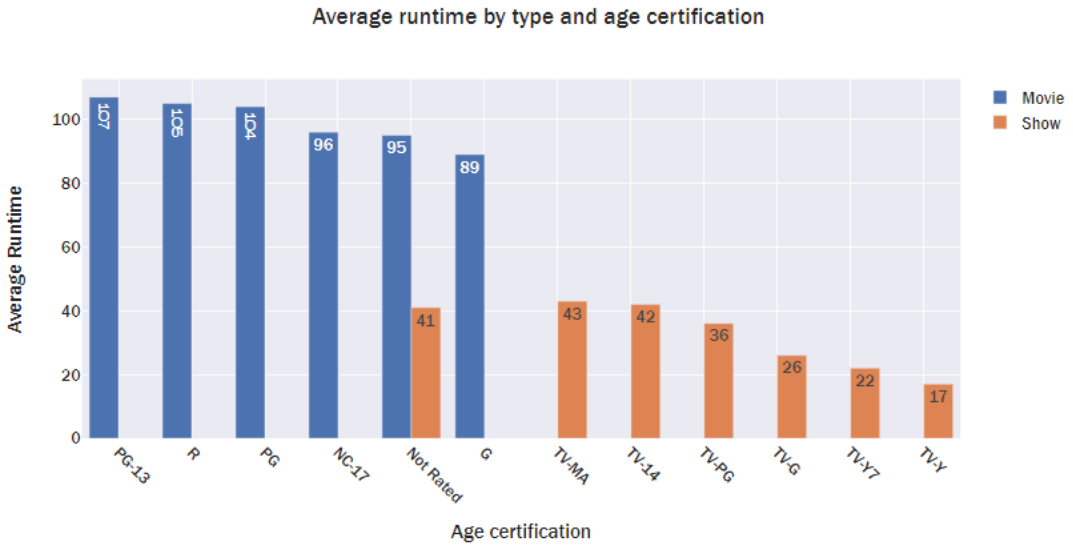



**For movies, the most common runtime is around 96 mins. For shows, the most common runtime is around 45 mins.**




**Most of the shows in the dataset relased around the second half of 2019, and most of the movies released around the first half of 2019.**




**For movies, the age certification of PG-13 has the highest average runtime of 107 mins, the age certification of G has the lowest average runtime of 89 mins.**

**For shows, the age certification of TV-MA has the highest average runtime of 43 mins, the age certification of TV-Y has the lowest average runtime of 17 mins.**

**IMPORTANT LINKS:**

**Drive Link:**

**https://drive.google.com/drive/folders/15kufgDTL731QXDmW-3votNl92Du_5ih3?usp=share_link**

**Video Link:**

**https://drive.google.com/drive/folders/1ygz_qg0KK70lO0V1BiYEcJU4Br0kiKJX?usp=share_link**

## Learning Experience:

Through this project we learnt the importance of distributed file systems i.e., it can share data from a single computing system among various servers, so client systems can use multiple storage resources as if they were local storage. Distributed file systems enabled us to access data in an easily scalable, secure, and convenient way. Using a distributed system improves scalability and performance because they can draw on the capabilities of other computing devices and processes.

Firebase allowed us to establish a real-time database connection, which meant multiple users could see the changes in the data when the data was created or edited.

It is not always convenient to analyze data manually when we have a huge database. SQL queries make it efficient to perform various operations such as getting rows based on certain filter criteria from the huge database and even performing manipulation to it as suited.
Using Partition Based MapReduce allowed us to scale the data processing easily over several computing nodes. It made the data retrieval efficient.

Django framework's REST (Representational State Transfer) framework which is a renowned toolkit for creating web APIs. This was an added benefit with Django as it was powerful enough to build a full-fledged API in just two or three lines of code. Django is independent and a complete framework. It does not require any other external solution. It is everything, from an ORM to a web server. This enabled to use various databases and switch them accordingly.

Creating a web browser-based User Interface made us realize the usefulness of planning out the layout of the web page before implementing it. It taught us that to make our interface easily navigable to any external user, it was essential to keep the interface simple and intuitive.

At first glance, the data in the databases is not easily readable to a user who has no experience with firebase or sql. Thus, it was important to show the same in a more understandable format. Developing a tree-view to emulate a computer's file system was essential to make viewing these databases as easy as an everyday browsing through computer files. Using Javascript to display the tree as well as change it as per the realtime database through django was not only a challenge but also a great opportunity to learn and appreciate the power of these programming languages to simultaneously change graphics while loading data stored far away from the user's computer. It helped us understand how a server can behave as a proxy for a frontend client to access and display data dynamically.

## Timeline & Contribution to the Project:

| Task | Member | Timeline |
|---|---|---|
| Data Extraction and Data pre- processing | Lakshita Shetty | September 25, 2022 |
| DataSet Generation | Ankit Tripathi | September 25, 2022 |
| Data Preparation<br>(which includes data visualization) | Shreya Nayak, Ankit Tripathi | September 28, 2022 |
| Creating the main EDFS (with a simple and user-friendly HTML front end)<br>Decide the flow of the app | Lakshita Shetty, Shreya Nayak, Ankit Tripathi | October 8, 2022 |
| Implementing MapReduce() for basic search/ analytics for at least one data set | Lakshita Shetty | October 25, 2022 |
| Development of Django | Shreya Nayak | October 25, 2022 |
| Implementing MapReduce() for the rest of the data sets | Lakshita Shetty | November 8,2022 |
| Development of Frontend-UI | Shreya Nayak | November 8,2022 |
| Final Compilation and Documentation with video and final report | Shreya Nayak, Lakshita Shetty | |

**Team Members:**

| Name | Current Degree | Previous Degree | Skills |
|---|---|---|---|
| Ankit Tripathi | MS in Applied Data Science | B.E Electronics and Telecommunication Engineering | Python(Numpy, Pandas, Scikit learn, etc.), SQL, C, C++, HTML & Java, java script, Flask, MATLAB, Big Data Frameworks, Machine Learning |
| Lakshita Shetty | MS in Applied Data Science | B.E Electronics and Telecommunication Engineering | Python(Numpy, Pandas, Scikit learn, etc.), SQL, R, C, C++, HTML & Java, Power BI, Tableau, MATLAB, Big Data Frameworks, Machine Learning |
| Shreya Nayak | MS in Applied Data Science | B.E Electronics and Telecommunication Engineering | Python (Numpy, Pandas, Scikit learn, etc.), C, R, SQL, Machine Learning, Big Data Frameworks, HTML, CSS, Tableau, Power BI, MATLAB,SCILAB |