Spark Movie Query application

Report

Student Name	Essa Umar Khan
University	Aston University
Module Code	CS3800
Module Name	Advanced Database Systems
Student Number	170077653
Contact Email	khane2@aston.ac.uk

Table of contents

List of Tables

List of Figures

Relevant Links

GitHub Repository

https://github.com/essakh/spark-application

This repository contains the source code for the application, the report, the dataset, and a tutorial document detailing how to run the application.

YouTube Video

https://youtu.be/Dm_yH_McmbI

This video serves as an evidence of the application working on a 16gb ram windows 10 computer. It is recommended to watch this video as it also serves as a walkthrough of the applications.

1. Introduction

The coursework requirement is to design and implement a data-driven application that is reliant on open-source data or, big data. The intended application does not have to be a fully-fledged software but should be capable of querying the underlying dataset to obtain useful information and present this to the end-user (Wang, 2021).

The remainder of the report is organised as follows: Area of Analysis, Design and Implementation, and concludes with presenting the Results and Reflections on the project.

2. Area of Analysis

2.1 Project Purpose

The aim of this project was to create an application that displays useful information about movie ratings data, based on an input from the user. The input is done by the user by pressing one button out of a set of buttons (where each button represents a request of querying the data). From the button press consequently the application runs a query (Spark Job written as a python file) to execute on the underlying data (stored in HDFS), and then the result is presented back to the user. The information will be displayed in a clear layout and an understandable format.

This application empowers non-technical users to be able to pass useful queries on big data in a user-friendly manner i.e. with a graphical user interface and this normally necessitates knowledge of difficult technologies such as Hadoop and Spark to actually do, but the GUI simplifies it and abstracts all the technical stuff away under the hood where it is automated for the user.

The application is being created to be a starting point that is easy to build on top off to add additional types of queries.

2.2 Intended Application

For this project, a Python application is created to provide a user interface for querying data utilizing Spark scripts on data stored in Hadoop HDFS document store. The Hortonworks Data Platform sandbox[1] virtual machine image is what facilitated the Hadoop cluster and Spark configuration for this Application to work. The cluster only consists one Node (one machine) for simplification purposes and this was sufficient, however it can easily be scaled up for the cluster to consist of more Nodes.

During development the need for a supplementary application to set-up the Hortonworks environment was realised (See section 5.2 for elaboration). Therefore, for this project two applications have been developed – the set-up application (hortonworksSetup.exe[2]) and the main application (MainApplication.exe[3]). Briefly, the horonworksSetup.exe application is essentially a GUI to trigger a shell/ssh script on a press of a button for the purpose of making the users experience simpler through automation in setting up the dataset files onto HDFS on the cluster and get the spark scripts stored onto CentOS[4] local files, as this is required for the MainApplication.exe to work and is intended to be used once prior to the use of the MainApplication.exe which is described as follows.

The MainApplication.exe consists of buttons that upon pressing will execute a shell scripts that submits one of three *Spark Applications*[5] written in Python (as a .py file) onto the cluster to process (query) the data. MainApplication.exe can query and

---

[1] The Sandbox is a virtual machine image and is a straightforward, pre-configured, environment consisting of Hadoop and associated technologies such as Spark. The Sandbox comes packaged in a virtual environment that can run in the cloud or on your personal machine (10minbasics, 2015). This virtual machine runs on CentOS. It has to be running for both the MainApplication.exe & hortonworksSetup.exe to work (see the Tutorial document for mor information).

[2] Can be accessed from project file location: …\spark-application\Applications\hortonworksSetup.exe

[3] Can be accessed from project file location: …\spark-application\Applications\MainApplication.exe

[4] In this report CentOS refers to the Hortonworks Data Platform sandbox virtual machine image's operating system as that runs CentOS (operating system).

[5] Can be found in project file location: …\spark-application\Spark files

retrieve the following information from the underlying dataset.

1) Show top ten worst rated movies
2) Show top ten worst rated movies with >= 10 rating submissions
3) Show top 10 Most Popular Movies

2.3 The Dataset

The datasets to be used are part of the 'MovieLens 100K (ml-100k) Dataset collection[6]'. This is open date (publicly available). From this collection the dataset files 'u.data' and 'u.item' were used for this project. In the README[7] text file of the ml-100k dataset collection the following characteristics about these files are given:

- The u.data file contains the full data set of 100,000 ratings by 943 users on 1682 items (movies).
    - Each user has rated at least 20 movies.
    - Users and items (movies) are numbered consecutively from 1.
    - The data is randomly ordered.
    - This is a tab separated list of user id | movie id | rating | timestamp.
    - The time stamps are unix seconds since 1/1/1970 UTC.
- The u.item file contains information about the items (movies).
    - This is a tab separated list of
        - movie id | movie title | release date | video release date | IMDb URL | unknown | Action | Adventure | Animation | Children's | Comedy | Crime | Documentary | Drama | Fantasy | Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi | Thriller | War | Western |

Moreover:

- ❖ The 'movie id' column of u.item correspond to the 'movie id' of u.data's data set (essentially a primary key-foreign key relationship).
- ❖ The figures for the rating column of u.data are out of 5 (5 being the best rating and 1 being the lowest rating).

When observing the datasets 'u.data' and 'u.item', it is apparent that there is more information about each individual movie than that is required for the intended application – most information is superfluous. The intended application only requires from 'u.data' and 'u.item' the columns 'movie id', 'rating' and 'movie id' , 'movie title' respectively, therefore programmatically all other columns not in use are ignored in the Application. See

---

[6] Download link: https://grouplens.org/datasets/movielens/100k/ . The collection is also stored on the github of this project & the submission file of this project (the zip downloaded from blackboard) in location: '…\spark-application\Spark files\ml-100k'

[7] Can be accessed from project file location: …\spark-application\Spark files\ml-100k\README

Table 1 & Table 2 for a visualisation.

Table 1) A tabular visualisation of a subset of the dataset 'u.data' for the purpose of demonstrating which columns (bolded) are relevant to our application.

user id	**movie id**	**rating**	timestamp
196	242	3	881250949
186	302	3	891717742
…	…	…	…

Table 2) A tabular visualisation of a subset of the dataset 'u.item' for the purpose of demonstrating which columns (bolded) are relevant to our application.

movie id	**movie title**	release date	…
1	Toy Story (1995)	01-Jan-1995	…
2	GoldenEye (1995)	01-Jan-1995	…
…	…	…	…

The flexible schema of this dataset also means that attributes don't have a fixed order and are practically optional in each document. Moreover, No data pre-processing was conducted.

It is worth noting that these datasets are not truly Big Data, only containing roughly 100,000 records, however as a Hadoop virtual cluster is being used for this project a smaller dataset is more suitable - as has been recommended in the coursework description (Wang, 2021). A direct benefit that is gained from using a smaller dataset is that they are quicker to process and hence the development is easier as the various Spark Application complete quicker, comparatively a true Big Data dataset will be very slow and cumbersome to process.

3. Design

3.1 Requirements Analysis

This section consists of formulating functional and non-functional requirements.

The formulation of the functional and non-functional requirements was not done within a waterfall methodology context rather a sort of Scrum software development approach was adopted; I would develop a section of the software and would be completely open to re-evaluate and change the approach I am taking, meaning I did not plan concretely ahead towards my final product from the outset. This was necessary as I was inexperienced with the technologies used (Spark 2 & Hadoop) and hence, was learning a lot and consequently changing decisions and approaches reasonably.

One example of such change of initial design was that it was realised close to the end of the development of the main application that another application will be needed to set-up the Hortonworks sandbox virtual machine image for the user with the necessary files in the correct location in an automated manner – this was to save the user from any complex and skill full work (of knowing how to operate Hadoop, Spark and CentOS terminal) and let script handle everything (See section 5.2 for further elaboration).

The following tables 3 & 4 contain the functional and non-functional requirements that reflect the final two applications accurately.

Table 3) Functional & Non-Functional requirements for MainApplication.exe

ID	**Requirement Description**	**Functional (N) or Non-functional (NF)**
A1	Can the application be opened as a GUI .exe application?	F
A2	Can the user interact with the underlying Datasets through the U.I with the use of buttons?	F
A3	Can the application query the Dataset ‘u.data’ for the top 10 most popular movies (i.e. top 10 movies with the highest number of rating submissions)?	F
A4	Can the application query the Dataset ‘u.data’ for the top 10 movies with the lowest average rating (i.e. top 10 worst movies)	F
A5	Can the application query the Dataset ‘u.data’ for the top 10 movies with the lowest average rating with at least 10 user ratings? (i.e. the top 10 worst movies with >= 10 user rating submissions).	F
A6	Can the application show the results back to the user of any queries requested in a format that has such columns along with each row being numbered 1 to 10: For requirement with ID ‘A3’: Movie name | Tot. Num. of rating submissions For requirement with ID ‘A4, A5’: Movie name | Tot. Num. of rating submissions | Avg. rating	F
A7	Can the application respond to a user input (button press) with the results within 3 minutes?	NF
A8	Can the application be opened at any time?	NF

A9	Can the application allow the user to see all the outputs of the application without having to lose/close a previous output?	NF

Table 4) Functional & Non-Functional requirements for hortonworksSetup.exe

ID	Requirement Description	Functional or Non-functional
B1	Can the application be opened as a GUI .exe?	F
B2	Can the application GUI respond to user input through the allocated button on the UI?	F
B3	Can the application download the ‘u.data’ file onto the Hadoop cluster in location hdfs:///user/maria_dev/ml-100k	F
B4	Can the application download the ‘u.item’ file onto the CentOS local location: /home/maria_dev/ml-100k'	F
B5	Can the application download all the Python scripts (Spark Applications) onto the CentOS local location: /home/maria_dev	F
B6	Can the application notify the end use that the set-up was successful?	F
B7	Can the application respond to a user input (button press) with the results within 3 minutes?	NF
B8	Can the application be opened at any time?	NF

3.2 GUI design

Since both of the applications themselves do not require any input other than to have something that allows the user to execute the underlying SSH scripts, a simple interface window was sufficient. What was decided is that for the MainApplication.exe a simple interface with three buttons to execute the three queries; and the results displayed back to the user through a pop-up. And for hortonworksSetup.exe a simple interface window with a button was decided. Both interfaces have instructions on screen, informing the user what to do and the estimated time it could take.

Making detailed wireframes appeared to be futile from the outset and a approach were the design would be established without having a rigid plan was more appropriate due to the simplicity of the application and a personal reason of lacking knowledge when embarking on the project of what Python is capable of in terms of UI creation. Therefore, no Wireframes were produced, only a rough mental idea of what the application should look like was the only UI plan.

4. Implementation

4.1 Brief Overview

There are two applications. Both are completely implemented in the Python language – all the way from the GUIs to the logic. For developing both MainApplication.exe & hortonworksSetup.exe the python package Spur[8] was used to run commands and manipulate files over SSH and the package PySimpleGUI[9] was used to facilitate the GUI creation. Moreover, for the Spark Applications (See Section 5.4 for more information) the standard Spark 2 packages were imported and utilised in Python.

Additional notable technologies that were used in the development of the Application are:

-Apache Ambari[10]: It comes pre-installed and configured in the Hortonworks Sandbox virtual machine image. It facilitated me to interact with HDFS and Spark technologies in a visual way through an Interface which made development of the Applications easier.

-PuTTY[11]: An SSH client utilised to SSH into the Hortonworks sandbox virtual machine image and run commands manually to HDFS and CentOS – this was vital during the development of the SSH scripts and was a steppingstone to formulate the SSH scripts to automate the process.

4.2 HortonworksSetup.exe

As discussed in previous sections of this report the Hortonworks Sandbox virtual machine image was utilized to facilitate the Hadoop cluster and Spark 2 configuration. The hortonworksSetup.exe is an application that downloads the necessary files onto the correct locations for the MainApplication.exe to work.

This application was created in response to an issue encountered with exporting the manually set-up Hortonworks image that I was using on the VDI desktop through Oracle VirtualBox software – this is also where I was developing the MainApplication.exe. The problem was that the VDI desktop had insufficient amount

---

[8] https://github.com/mwilliamson/spur.py

[9] https://pysimplegui.readthedocs.io/en/latest/

[10] https://ambari.apache.org/

[11] https://www.chiark.greenend.org.uk/~sgtatham/putty/

of hard drive space to facilitate the exportation of this manually set-up Hortonworks image to an Appliance so that it could be downloadable and importable by someone else with all the necessary files ready for directly using MainApplication.exe. Therefore, as a solution to this problem the decision was made to have anyone that desires to make use of the MainApplication.exe to first download a fresh (unaltered) image of the Hortonworks sandbox virtual machine and in order to download the necessary files onto it to have a separate application (hortonworksSetup.exe) that does it for the user in an automated manner quickly rather than having the user do complex interactions with a Shell through an SSH client.

The hortonworksSetup.exe application is a user-friendly way to launch a shell script through the click of a button. This will set-up all the necessary files in the right locations on the cluster's HDFS and virtual machine's CentOS in an automated manner. So that the actual main application (MainApplication.exe) functionality works.

The python file where the code lies for this application is **hortonworksSetup.py**[12]. Both the GUI and Shell script are coded in this one file. In Figure 1 a flowchart is given to demonstrate what the application on the shell script level is abstractly doing (high-level overview). It also elucidates where each of the key files that MainApplication.exe depends on are downloaded and stored to.

[12] Can be accessed from project file location: …\spark-application\Applications\Hortonworks Sandbox set-up**hortonworksSetup.py**

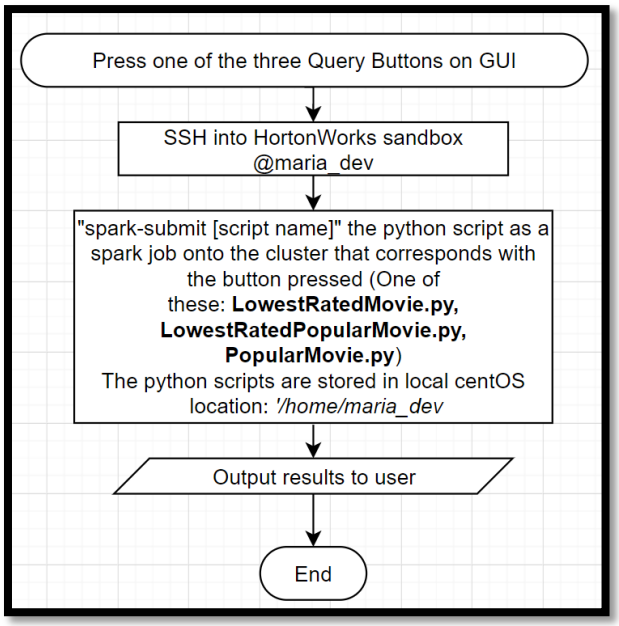

Figure 1) A flowchart showing a high-level overview of the actions that are performed by hortonworksSetup.py

A point to note is that 'u.data' file has been stored onto HDFS but 'u.item' has been stored on a local location of the CentOS. The reason for this is that 'u.item' contains the movie names and a limited amount of associated metadata because of this it was rationalised that since such data is not Big Data and realistically will not scale up to be of the Big Data category then it will be more efficient to just store it in the local CentOS file directory location as a normal file would; since, it is certain that this file will be small then it is more efficient and safe to load it up into memory/ram.

On the other hand, the 'u.data' contains ratings data, albeit at the moment it is only 2mb of size but since the dataset consists of user ratings data and such type of data can realistically scale up to become of the Big Data category, hence it was decided to have this stored in HDFS on a cluster and let the Spark Applications (see section 5.4) run queries (processes) and retrieve data from the HDFS location rather then loading the data up into memory.

4.3 MainApplication.exe

This application is intended to run after hortonworksSetup.exe application has ran – else it won't work as the necessary files won't exist on the Hortonworks Sandbox virtual machine image in such a case.

Essentially a button press on the MainApplication.exe GUI launches a shell script that submits a spark job onto the cluster. On the CentOS' local file directory there are three python (.py) files in existence and each is launched on to the HDFS cluster

utilizing the *spark-submit*[13] command to execute the Spark job on the underlying dataset stored in HDFS (u.data) for the purpose of querying it.

The python file where the code lies for this application is **MainApplication.py**[14] & **ssh.py**[15], where the GUI and Shell script are coded respectively. Below a flowchart is given to demonstrate what the shell script is abstractly doing (high-level overview) – this figure 2 will also elucidate from which location the files are launched.

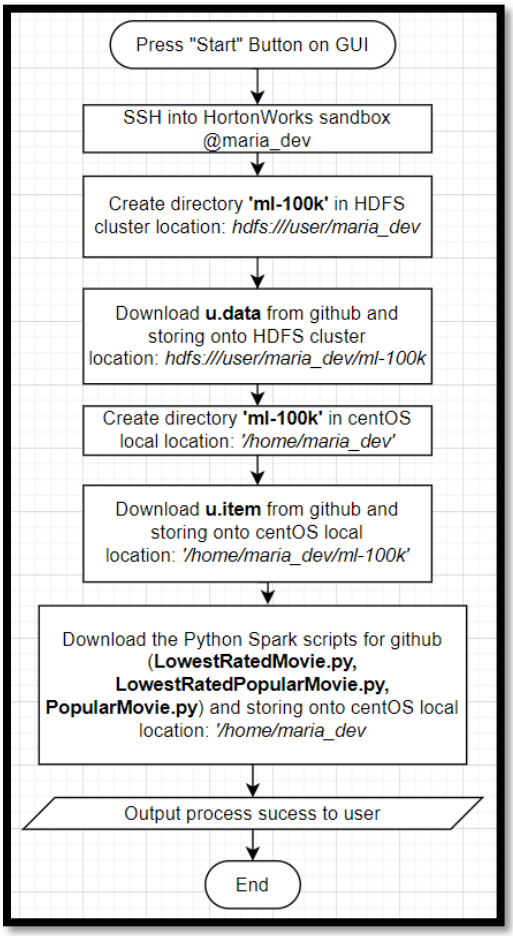

Figure 2) A flowchart showing a high-level overview of the actions that are performed by ssh.py

[13] To read more about this command: https://spark.apache.org/docs/latest/submitting-applications.html

[14] Can be accessed from project file location: …\spark-application\Applications\Main Application**MainApplication.py**

[15] Can be accessed from project file location: …\spark-application\Applications\Main Application**ssh.py**

4.4 The Three Spark Applications (Python Scripts)

The code is well documented and self-explanatory, but in this section notable points are expounded that are relevant to all scripts and in the following subsections particular noteworthy points for each specific script are touched upon. The three Spark Applications are LowestRatedMovie.py[16], LowestRatedPopularMovie.py[17]and PopularMovie.py[18]. They are executed as a Spark Job onto HDFS. Figure 4 shows which buttons on the MainApplication.exe GUI corresponds with the execution of what Spark Application across the cluster.

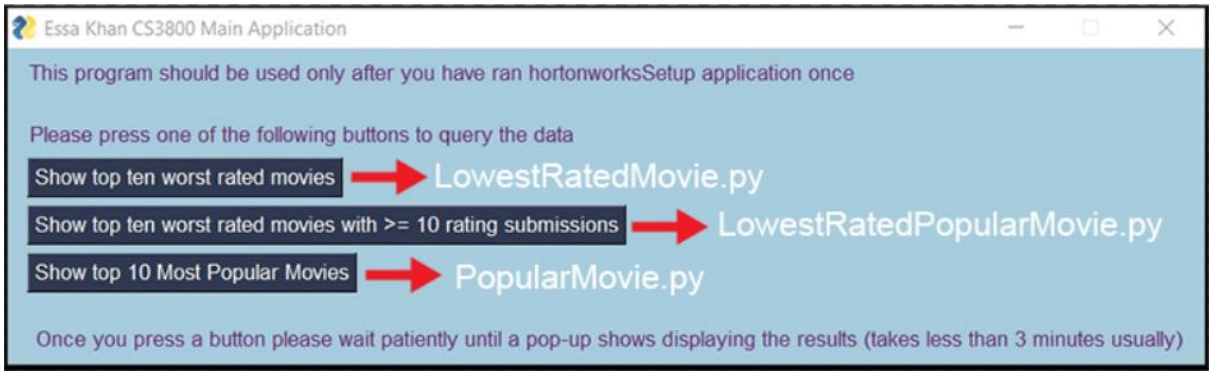

Figure 3) Image of the MainApplication.exe with annotations showing which Spark Application's execution corresponds with what button.

All of these scripts include the code lines in Figure 4, 5 & 6, and the following paragraph rationalises what is happening in these lines.

[16] Can be accessed from project file location: …\spark-application\Spark files\LowestRatedMovie.py

[17] Can be accessed from project file location: …\spark-application\Spark files\LowestRatedPopularMovie.py

[18] Can be accessed from project file location: …\ spark-application\Spark files\PopularMovie.py

```
5  def loadMovieNames():
6      movieNames = {}
7      with open("ml-100k/u.item") as f:
8          for line in f:
9              fields = line.split('|')
10             movieNames[int(fields[0])] = fields[1]
11     return movieNames
```

Figure 4) An extract from LowestRatedMovie.py but these code lines are common to all 3 Spark Application's python files.

```
21     # Load up movie ID -> name dictionary
22     movieNames = loadMovieNames()
23
24     # Get the raw data
25     lines = spark.sparkContext.textFile("hdfs:///user/maria_dev/ml-100k/u.data")
26     # Convert it to a RDD of Row objects with (movieID, rating)
27     movies = lines.map(parseInput)
28     # Convert that to a DataFrame
29     movieDataset = spark.createDataFrame(movies)
```

Figure 5) An extract from LowestRatedMovie.py but these code lines are common to all 3 Spark Application's python files.

In line 22 of figure 5, the method *loadMovieNames* is called *(see figure 5* for its definition). This method is not utilizing any of the various spark libraries imported in each of the Spark Application's .py files. Rather, it is creating a python dictionary that maps movie ID's to movie names from the file stored in CentOS named 'u.item' – and since we know this will always be a small amount of data, it is loaded up into memory for faster use. This dictionary is utilised to produce our final output as defined in the functional requirements in section 4.1 of the report.

Next in line 24 of figure 5, the data (u.data) is first loaded up into an RDD named *lines.* The benefit of spark here is apparent that it abstracts away a lot of complexity like how the data is actually spread across the HDFS cluster and retrieved (theoretically this point remains true for any number of nodes in the cluster, even though at the moment there is only one node in the cluster in use for this project).

```
13 def parseInput(line):
14     fields = line.split()
15     return Row(movieID = int(fields[1]), rating = float(fields[2]))
```

Figure 6) An extract from LowestRatedMovie.py but these code lines are common to all 3 Spark Application's python files.

Next in line 27 of figure 5, the RDD created in line 24 is parsed to formulate a new RDD with only the columns movie ids and ratings as this is all what is needed for the queries. It does this by calling a method *parseInput* (see figure 6 for its definition). What *parseInput* is ultimately doing is that it extracts the 'Movie Id' and 'Rating' from 'u.data' whilst casting them to an appropriate datatype and returning a RDD of row objects that has these as fields.

Next, in line 29 of figure 5: this RDD is converted into a Spark DataFrame object that has all the information it needs built in (the types & column names). The benefit of using a Spark DataFrame is that it allows to do the required queries in a few lines of code compared to RDD where a single simple query would take several lines of code. Moreover, these DataFrames are interchangeable with other components of spark, such as mlLiB and spark streaming, so certain parts of the code – for further future development - could be adjusted for such functionality as well.

4.4.1 LowestRatedMovie.py & LowestRatedPopularMovie.py

LowestRatedPopularMovie.py was developed in reaction to a problem encountered with the usefulness of the results produced by LowestRatedMovie.py – this has been expounded upon in section 6.5 of the report.

```
37    # Join the two together (We now have movieID, avg(rating), and count columns)
38    averagesAndCounts = counts.join(averageRatings, "movieID")
39
40    # Filter movies rated 10 or fewer times
41    popularAveragesAndCounts = averagesAndCounts.filter("count > 10")
42
43    # Pull the top 10 results
44    topTen = popularAveragesAndCounts.orderBy("avg(rating)").take(10)
```

Figure 7) An extract from LowestRatedPopularMovie.py

To create LowestRatedPopularMovie.py only small adjustments and additions were needed to what was already developed for LowestRatedMovie.py where line 41 (see figure 7) is the main change made. The line of code is a filter command – all the rows where the count was less than 10 is filtered out.

4.4.2 PopularMovie.py

For this application, the most Popular Movies has been defined as the movies with the most amount of user rating submissions; in other words, the amount of times a particular movie appeared in the 'u.data' dataset. A noteworthy point about PopularMovie.py is that it leverages the power of Spark 2's DataFrame objects and does the main part of this process of all in one line as shown in figure 8.

```
31    # SQL-style technique to sort all movies by popularity in one line!
32    topMovieIDs = movieDataset.groupBy("movieID").count().orderBy("count", ascending=False).cache()
```

Figure 8) An extract from PopularMovie.py

5. Results & Reflections

All the functional & non-functional requirements listed in section 4.1 have been satisfied by the implementation – this in itself marks the success of the deliverable. The following is thorough systematic evaluation of the software product rather then just a mere checklist of which requirements have been met.

5.1 Source code quality

The code does what it should reliably. It is well documented – annotation wise and follows good practises in naming conventions of the various methods & variables which also makes it easy to understand. Moreover, all the various .py files involved from the Spark Applications all the way to the GUI are modular either as the python module itself or the functions that make them up therefore it facilitates isolating code, and automated testing – making it very testable. All of this demonstrated good source code quality.

The code at this moment in time - provided that the source code consists of only a handful of files in total – is maintainable. However, there remains scope for improvement. For example, the SSH related code within hortonworksSetup.py from line 11-27 (see figure 9) consists of commands to run on the shell. Rather, then individually running each command on a separate line of code, instead the commands could be held in a list object and looped over and executed in such manner instead – this would have improved maintainability of the code due to it becoming more succinct and facilitates easier future additions to the code. There are a few such kind improvements to maintainability that can be made throughout the code files.

```
################################SSH Script#################################################

def SSHscript():
    shell = spur.SshShell(hostname="127.0.0.1", port=2222, username="maria_dev", password="maria_dev",
                          missing_host_key=spur.ssh.MissingHostKey.accept)

    with shell:
        #Storing u.data into HDFS cluster
        shell.run(["curl", "-O", "https://raw.githubusercontent.com/essakh/spark-application/master/Spark%20files/ml-100k/u.data"])
        #shell.run(["hadoop", "fs", "-rm", "-r", "ml-100k"])
        shell.run(["hadoop", "fs", "-mkdir", "ml-100k"])
        shell.run(["hadoop", "fs", "-copyFromLocal", "u.data", "ml-100k/u.data"])
        shell.run(["rm", "u.data"])

        #Storing u.item onto local
        shell.run(["curl", "-O", "https://raw.githubusercontent.com/essakh/spark-application/master/Spark%20files/ml-100k/u.item"])
        #shell.run(["rm", "-rf", "ml-100k"])
        shell.run(["mkdir", "ml-100k"])
        shell.run(["mv", "u.item" ,"ml-100k"])

        #Storing Spark Scripts onto local
        shell.run(["curl", "-O", "https://raw.githubusercontent.com/essakh/spark-application/master/Spark%20files/LowestRatedMovie.py"])
        shell.run(["curl", "-O", "https://raw.githubusercontent.com/essakh/spark-application/master/Spark%20files/LowestRatedPopularMovie.p
        shell.run(["curl", "-O", "https://raw.githubusercontent.com/essakh/spark-application/master/Spark%20files/PopularMovie.py"])
```

Figure 9) An extract from hortonworksSetup.py

5.2 Usability, UI

Usability is about how well humans can use the applications to achieve a desired goal. The GUIs is what truly makes the application's usability user friendly. Due to it only having the necessary input elements (i.e. the buttons) it is has a negligible learning curve to learn to use – it is intuitive. Minus the GUIs, the alternative would have had been handing the user a list of shell commands to execute to achieve the desired goals – this of course has many practical drawbacks such as being complex and tiresome. The UIs are fully functional and reliable.

A point to note is that the non-functional requirement with the ID 'A9' in table 3 (section 4.1 of the report) has also been met as the MainApplication.exe allows the user to see all the outputs of the application without having to lose a previous output i.e. once one pop-up displays, it does not have to be closed for another Spark Application to start and display it's results back as a pop-up. Figure 10 demonstrates this:

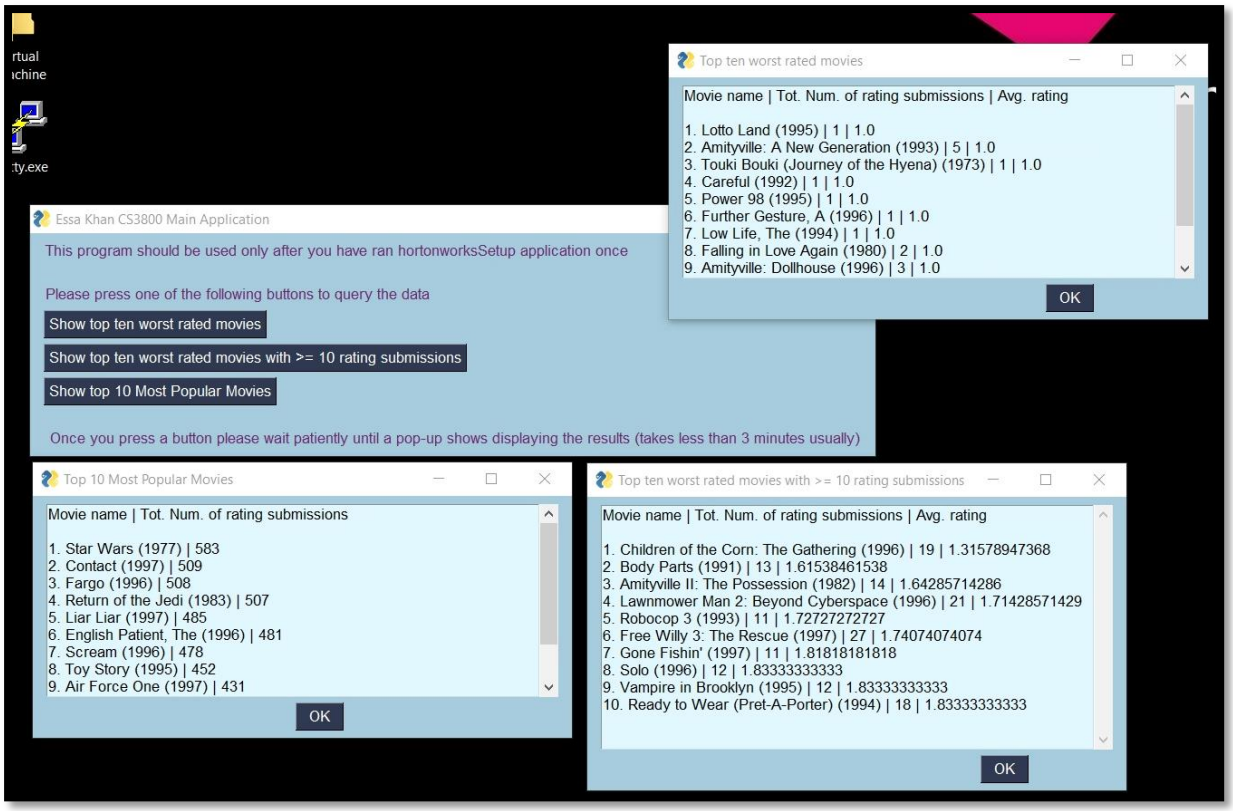

Figure 10) MainApplication.exe in a state where all results are open at once on screen.

Nevertheless, there is room for improvement in the actual aesthetics of the GUIs – in terms of making them look more professional. Take for example the pop-windows for displaying the results back to the user, here (see figure 11) an improvement would be to display the results back in a proper table rather than being '|' character separated.

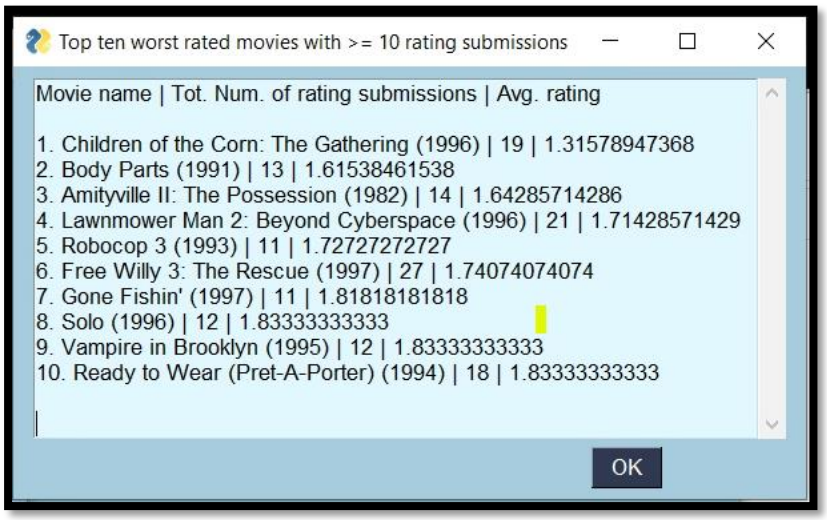

Figure 11) A pop-window displaying the results of a button press on MainApplication.exe.

Moreover, for both the GUIs a visual progress bar or loading animation whenever a process is executing would make the user experience better. As at the moment, the UI freezes until a triggered process is completed.

Lastly, currently if the Application is used in unintended cases (intended cases can be gauged from the Tutorial document) such as when the MainApplication.exe is used before having executed hortonworksSetup.exe process at least once, or when any of the two applications are used before even having launched the Hortonworks Sandbox virtual machine image properly etc. then an appropriate useful error message should show. Currently no such error messages are displayed for any unintended case of use – which can leave the user confused.

5.3 Results

To verify the results produced manually would be difficult since the datasets are large. However, based on first principles (i.e. based on the actual queries – the code written - in the Spark Applications [.py files]) there is confidence in the veracity of the results produced and displayed back to the user through MainApplication.exe. Table 5 shows the results produced from pressing the buttons on MainApplication.exe.

Table 5) All results output by MainApplication.exe

Button on MainApplication.exe: **'Show top 10 worst rated movies'** *Spark Application triggered:* **'LowestRatedMovie.py'**	*Button on MainApplication.exe:* ***'Show top 10 worst rated movies with >= 10 rating submissions'*** *Spark Application triggered:* **'LowestRatedPopularMovie.py'**	*Button on MainApplication.exe:* **'Show top 10 most popular movies'** *Spark Application triggered:* **'PopularMovie.py'**
Movie name | Tot. Num. of rating submissions | Avg. rating 1. Lotto Land (1995) | 1 | 1.0 2. Touki Bouki (Journey of the Hyena) (1973) | 1 | 1.0	Movie name | Tot. Num. of rating submissions | Avg. rating 1. Children of the Corn: The Gathering (1996) | 19 | 1.31578947368 2. Body Parts (1991) | 13 | 1.61538461538	Movie name | Tot. Num. of rating submissions 1. Star Wars (1977) | 583 2. Contact (1997) | 509

3. Amityville: A New Generation (1993) | 5 | 1.0 4. Careful (1992) | 1 | 1.0 5. Falling in Love Again (1980) | 2 | 1.0 6. Power 98 (1995) | 1 | 1.0 7. Low Life, The (1994) | 1 | 1.0 8. Amityville: Dollhouse (1996) | 3 | 1.0 9. Further Gesture, A (1996) | 1 | 1.0 10. Hostile Intentions (1994) | 1 | 1.0	3. Amityville II: The Possession (1982) | 14 | 1.64285714286 4. Lawnmower Man 2: Beyond Cyberspace (1996) | 21 | 1.71428571429 5. Robocop 3 (1993) | 11 | 1.72727272727 6. Free Willy 3: The Rescue (1997) | 27 | 1.74074074074 7. Gone Fishin' (1997) | 11 | 1.81818181818 8. Vampire in Brooklyn (1995) | 12 | 1.83333333333 9. Solo (1996) | 12 | 1.83333333333 10. Ready to Wear (Pret-A-Porter) (1994) | 18 | 1.83333333333	3. Fargo (1996) | 508 4. Return of the Jedi (1983) | 507 5. Liar Liar (1997) | 485 6. English Patient, The (1996) | 481 7. Scream (1996) | 478 8. Toy Story (1995) | 452 9. Air Force One (1997) | 431 10. Independence Day (ID4) (1996) | 429

A noteworthy point is that the Spark Application LowestRatedPopularMovie.py was developed after the realisation that the results of LowestRatedMovie.py were not really meaningful. Firstly, because at the very least 10 movies in the dataset have a rating of 1 out of 5, so the application returns a set of 10 movies out of the movies that have a one start rating; therefore, this application tends to return a different result for the top 10 worst rated movies during each run as there is no other criteria other than the movies' ratings to rank them. Moreover, a majority of these movies that have a 1-star rating are rated by one person and therefore it doesn't give a strong validity to the movie being one of the worst movies in the dataset as maybe the person had a unique taste of their own – the sample size is too little for comfort. Thus, 'LowestRatedPopularMovie.py' was created which queries the dataset for the lowest rated movies with at least 10 people having had rated the movie. This gives more confidence that these low rated moves are truly bad movies.

5.4 Performance

The non-functional requirement with the ID 'A9' in table 3 (section 4.1 of the report), related to the amount of time it takes to launch the processes from the Applications, have been met and exceeded. As all the processes related to MainApplicaiton.exe &

hortonworkSetup.exe have been benchmarked and shown to execute much faster then the times aimed for in the Non-Functional requirements.

5.5 Conclusion

In this report the documentation of the development process of a data-driven application against an underlying dataset has been covered. It has discussed analysis, design, implementation and reflected on the results obtained against the project objectives in a systematic manner.

References

These were explicitly mentioned in this report.

10minbasics (2015). *What is Hortonworks Sandbox | 10 Min Basics - Sandbox.* [online] 10 Min Basics. Available at: http://10minbasics.com/what-is-hortonworks-sandbox/#:~:text=Sandbox%20is%20a%20virtual%20machine [Accessed 3 Apr. 2021].

Wang, H. (2021). *Coursework Description CS3800*.