CS 108 - Bash Grader

Siddhant Mulkikar

April 28, 2024

Contents

1 Objective

The objective of this CS108 Project was to create a csv file manager and interpreter with a command line interface.

2 Introduction

Bash Grader is a command line interface for instructors to grade students and manage their marks. It is an incredibly useful tool with its many functionalities and features.

3 List of Functionalities

3.1 Basic functionalities

- Upload a marklist in the form of a csv file.
- Total the marks of each student over all exams
- Combine the marks of all exams into a single csv file
- Update the marks of a student in a particular exam or add a student's record to an exam
- Implement a working git system with init, add, commit and checkout functionalities

3.2 Customisations

- A staging area with add and remove funtionalities.
- A diff patch system for the checkout functionality
- Commit history log
- Graphs for a particular exam or all exams
- Stats for a particular exam or all exams
- Report card generation for all students with grade calculation for all students, based on manual rubrics from the instructor/user
- Ranked marklists for a particular exam or all exams
- Shows git head

4 Usage

4.1 Upload

Run the following command to upload a csv file in your current directory.

```
bash submission.sh upload <filename>
```

4.2 Total

Run the following command to total the marks of each student over all exams into a total column in `main.csv`.

```
bash submission.sh total
```

4.3 Combine

The following command combines all the csv files in the current directory into `main.csv`.

```
bash submission.sh combine
```

4.4 Update

Run the following command to update the marks of a particular student or add a student's record to an exam.

```
bash submission.sh update
```

4.5 Rank

Run the following command to get the ranked marklist for a particular exam or `main.csv`.

```
bash submission.sh rank <exam-name>
```

4.6 Git Init

Run the following command to initialize a remote git repository.

```
bash submission.sh git_init <remote-repo-path>
```

4.7 Git Add

Run the following command to add files from the current directory to the staging area.

```
bash submission.sh git_add
```

4.8 Git Remove

Run the following command to remove files from the staging area.

```
bash submission.sh git_remove
```

4.9 Git Commit

Run the following command to commit the files in the staging area to the remote repository.

```
bash submission.sh git_commit -m "commit message"
```

4.10 Git Checkout

Run the following command to checkout a particular commit from the remote repository.

```
bash submission.sh git_checkout <commit-id>
bash submission.sh git_checkout -m <commit-message>
```

4.11 Git Log

Run the following command to get the commit history log.

```
bash submission.sh git_log
```

4.12 Git Head

Run the following command to get the head of the git repository, i.e. the current version of directory.

```
bash submission.sh git_head
```

4.13 Graphs

Run the following command to get the graphs for a particular exam or all exams.

```
bash submission.sh graphs
```

4.14 Stats

Run the following command to get the stats for a particular exam or `main.csv`.

```
bash submission.sh stats
```

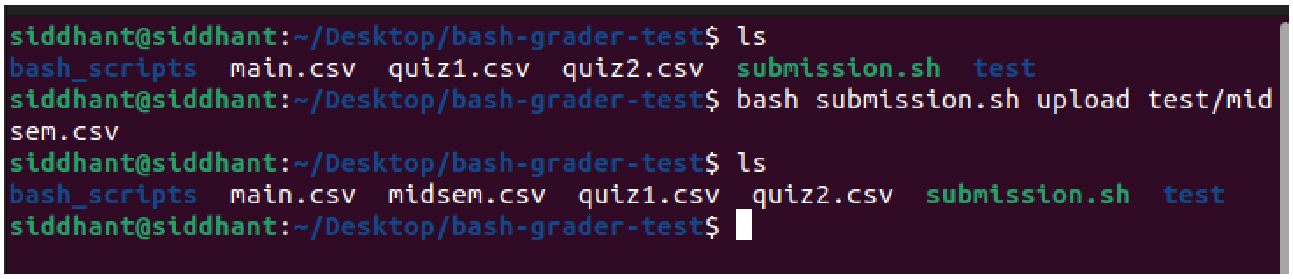
4.15 Report Card

Run the following command to get the report card/s for a particular student or all students.

```
bash submission.sh report_card
```

5 Working

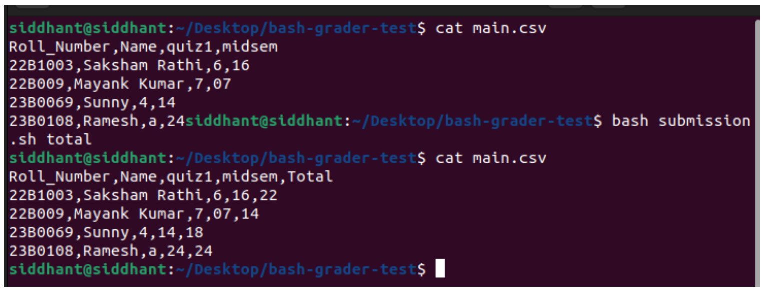
5.1 Upload



```
siddhant@siddhant:~/Desktop/bash-grader-test$ ls
bash_scripts  main.csv  quiz1.csv  quiz2.csv  submission.sh  test
siddhant@siddhant:~/Desktop/bash-grader-test$ bash submission.sh upload test/mid
sem.csv
siddhant@siddhant:~/Desktop/bash-grader-test$ ls
bash_scripts  main.csv  midsem.csv  quiz1.csv  quiz2.csv  submission.sh  test
siddhant@siddhant:~/Desktop/bash-grader-test$ 
```

In the above example, the file `quiz1.csv` is uploaded i.e. copied into the current directory from the filepath given as argument by the user. The script checks if the file exists and if it is a csv file. If it is, only then is the file copied into the current directory.
Script files : upload.sh

5.2 Total



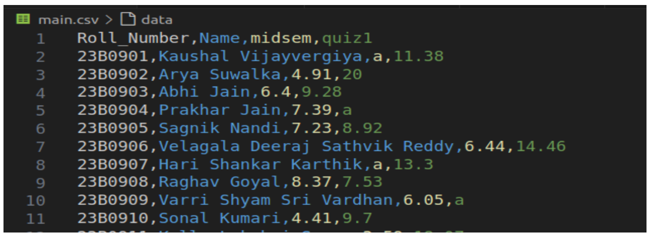
```
siddhant@siddhant:~/Desktop/bash-grader-test$ cat main.csv
Roll_Number,Name,quiz1,midsem
22B1003,Saksham Rathi,6,16
22B009,Mayank Kumar,7,07
23B0069,Sunny,4,14
23B0108,Ramesh,a,24siddhant@siddhant:~/Desktop/bash-grader-test$ bash submission
.sh total
siddhant@siddhant:~/Desktop/bash-grader-test$ cat main.csv
Roll_Number,Name,quiz1,midsem,Total
22B1003,Saksham Rathi,6,16,22
22B009,Mayank Kumar,7,07,14
23B0069,Sunny,4,14,18
23B0108,Ramesh,a,24,24
siddhant@siddhant:~/Desktop/bash-grader-test$ 
```

In the above example, the script totals the marks of each student over all exams and adds a new column `Total` to the `main.csv` file. If the marks column has `a` as its entry, the script skips that row while totaling.
Script files : total.awk

5.3 Combine

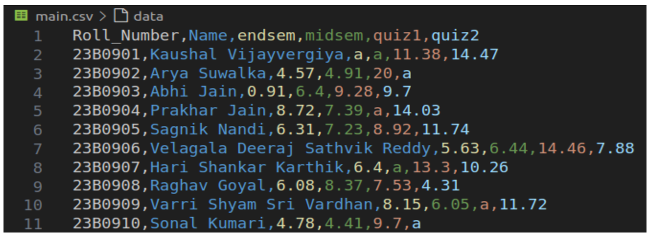
In the below example, the script combines all the csv files in the current directory into a single `main.csv` file. The script checks if the file is a csv file and if it is not the `main.csv` file.
`main.csv` is constructed from scratch in the following steps:

1. First, an array of all unique roll numbers and names is created.
2. Then, based on the exam files, a header is created with the roll numbers and names.
3. A mesh of `a`'s is created with the dimensions of the header.
4. Then the script iterates over all exam files and fills in the marks of each student in the mesh, whenever they are found. This ensures that anyone who is not present in a particular exam file is marked as `a`(absent) in `main.csv` for that particular exam.



```
Roll_Number,Name,midsem,quiz1
23B0901,Kaushal Vijayvergiya,a,11.38
23B0902,Arya Suwalka,4.91,20
23B0903,Abhi Jain,6.4,9.28
23B0904,Prakhar Jain,7.39,a
23B0905,Sagnik Nandi,7.23,8.92
23B0906,Velagala Deeraj Sathvik Reddy,6.44,14.46
23B0907,Hari Shankar Karthik,a,13.3
23B0908,Raghav Goyal,8.37,7.53
23B0909,Varri Shyam Sri Vardhan,6.05,a
23B0910,Sonal Kumari,4.41,9.7
```

(a) Before combine

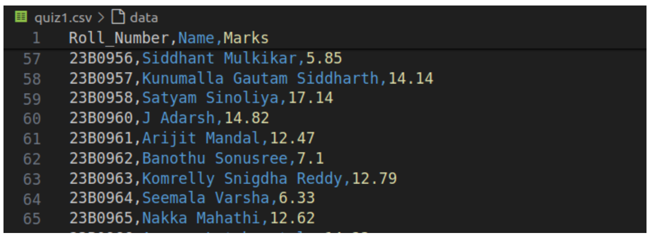


```
Roll_Number,Name,endsem,midsem,quiz1,quiz2
23B0901,Kaushal Vijayvergiya,a,a,11.38,14.47
23B0902,Arya Suwalka,4.57,4.91,20,a
23B0903,Abhi Jain,0.91,6.4,9.28,9.7
23B0904,Prakhar Jain,8.72,7.39,a,14.03
23B0905,Sagnik Nandi,6.31,7.23,8.92,11.74
23B0906,Velagala Deeraj Sathvik Reddy,5.63,6.44,14.46,7.88
23B0907,Hari Shankar Karthik,6.4,a,13.3,10.26
23B0908,Raghav Goyal,6.08,8.37,7.53,4.31
23B0909,Varri Shyam Sri Vardhan,8.15,6.05,a,11.72
23B0910,Sonal Kumari,4.78,4.41,9.7,a
```

(b) After combine

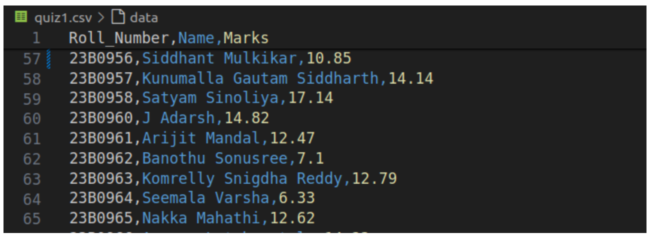
Script files : combine.sh, combine.awk

5.4 Update



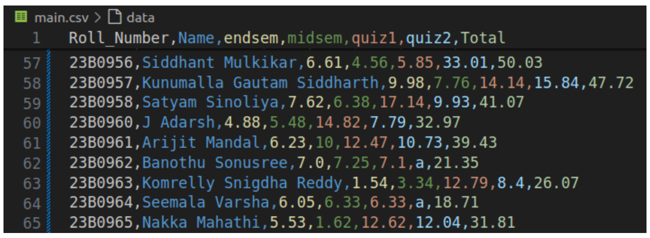
```
Roll_Number,Name,Marks
23B0956,Siddhant Mulkikar,5.85
23B0957,Kunumalla Gautam Siddharth,14.14
23B0958,Satyam Sinoliya,17.14
23B0960,J Adarsh,14.82
23B0961,Arijit Mandal,12.47
23B0962,Banothu Sonusree,7.1
23B0963,Komrelly Snigdha Reddy,12.79
23B0964,Seemala Varsha,6.33
23B0965,Nakka Mahathi,12.62
```

(a) quiz1.csv before update



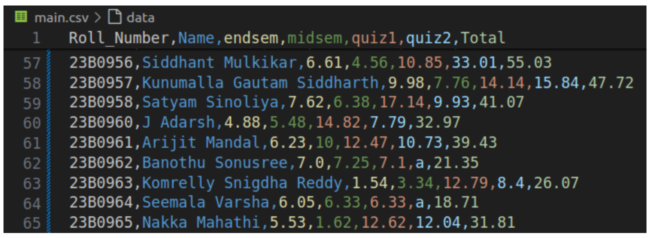
```
Roll_Number,Name,Marks
23B0956,Siddhant Mulkikar,10.85
23B0957,Kunumalla Gautam Siddharth,14.14
23B0958,Satyam Sinoliya,17.14
23B0960,J Adarsh,14.82
23B0961,Arijit Mandal,12.47
23B0962,Banothu Sonusree,7.1
23B0963,Komrelly Snigdha Reddy,12.79
23B0964,Seemala Varsha,6.33
23B0965,Nakka Mahathi,12.62
```

(b) quiz2.csv after update



```
Roll_Number,Name,endsem,midsem,quiz1,quiz2,Total
23B0956,Siddhant Mulkikar,6.61,4.56,5.85,33.01,50.03
23B0957,Kunumalla Gautam Siddharth,9.98,7.76,14.14,15.84,47.72
23B0958,Satyam Sinoliya,7.62,6.38,17.14,9.93,41.07
23B0960,J Adarsh,4.88,5.48,14.82,7.79,32.97
23B0961,Arijit Mandal,6.23,10,12.47,10.73,39.43
23B0962,Banothu Sonusree,7.0,7.25,7.1,a,21.35
23B0963,Komrelly Snigdha Reddy,1.54,3.34,12.79,8.4,26.07
23B0964,Seemala Varsha,6.05,6.33,6.33,a,18.71
23B0965,Nakka Mahathi,5.53,1.62,12.62,12.04,31.81
```

(a) main.csv before update



```
Roll_Number,Name,endsem,midsem,quiz1,quiz2,Total
23B0956,Siddhant Mulkikar,6.61,4.56,10.85,33.01,55.03
23B0957,Kunumalla Gautam Siddharth,9.98,7.76,14.14,15.84,47.72
23B0958,Satyam Sinoliya,7.62,6.38,17.14,9.93,41.07
23B0960,J Adarsh,4.88,5.48,14.82,7.79,32.97
23B0961,Arijit Mandal,6.23,10,12.47,10.73,39.43
23B0962,Banothu Sonusree,7.0,7.25,7.1,a,21.35
23B0963,Komrelly Snigdha Reddy,1.54,3.34,12.79,8.4,26.07
23B0964,Seemala Varsha,6.05,6.33,6.33,a,18.71
23B0965,Nakka Mahathi,5.53,1.62,12.62,12.04,31.81
```

(b) main.csv after update

In the above example, the script updates the marks of **23B0956** in `quiz1.csv` and `main.csv`.
The script first asks the user for the roll number and name of the student whose marks are to be updated. It then checks if the student's name matches with the name paired up with the roll number given by the user. If the names do not match, the script confirms with the user with a prompt for the correct name.
It then checks if the student is present in the `main.csv` file and if the exam file exists. If the student is present in the `main.csv` file, the script updates the marks of the student in `main.csv` by calling `update_main.awk` and the exam file by calling `update_exam.awk`.
If the student's record does not exist in the `main.csv` file, the script prompts the user to confirm if they want to add the student's record to an exam file and `main.csv`. If the user confirms, the script adds the student's record to the exam file and `main.csv`.
It also asks the user if they want to update the marks of the student in some other exam and repeats the same process. If the user does not want to update the marks of the student in any other exam, the script checks if

`main.csv` had been totaled before. If it was, then the script calls `total.awk` to correct the total, as the marks in `main.csv` have been updated.

Script files : update.sh, update_exam.awk, update_main.awk, total.awk

5.5 Rank

This commands takes the examname as a command line argument and ranks the students based on the marks in that exam. If the user enters `main` as the examname, the script ranks the students based on the total marks in `main.csv`. The ranked marklists are sorted according to rank and are stored in the folder `ranked_marklists` in the current directory.

Script files : rank.sh

5.6 Git Init

This command initializes a remote git repository at the path given by the user as a command line argument. It also checks if the path given is a valid path. If a remote repository is already initialized for the current directory, the script prompts the user to confirm if they want to reinitialize the repository. If the user confirms, the script reinitializes a fresh remote repository at the path provided by the user.

It also creates a hidden folder `.gitrepo` in the current directory. This folder contains a hidden file `.gitreponame` which stores the name of the remote repository and `.git_log` which records the commit history.

Apart from initialization, the script also copies all the csv files in the current directory at the time of initialization to a hidden folder `.ogfiles` in the remote repository. The `.ogfiles` folder contains the first version of every file which was ever committed. This serves as a refernce point for all files, for my **diff patch** system for `git\_checkout` to work.

Script files : git_init.sh

5.7 Git Add

This command first checks if a remote repository has been initialized for the current directory. If it has not, the script prompts the user to initialize a remote repository first.
This command then prompts the user to enter all the files, which they want to add to the staging area, separated by spaces. It then copies all the files to the staging area in the remote repository. If any particular file doesnt exist in the current directory, it gives an error message.
Script files : git_add.sh

5.8 Git Remove

This command first checks if a remote repository has been initialized for the current directory. If it has not, the script prompts the user to initialize a remote repository first.
It then prompts the user to enter all the files, which they want to remove from the staging area, separated by spaces. It then deletes all the files from the staging area in the remote repository. If any particular file doesnt exist in the staging area, it gives an error message.

Script files : git_remove.sh

5.9 Git Commit

The command first checks if a remote repository has been initialized for the current directory. If it has not, the script prompts the user to initialize a remote repository first.
It then checks what all files in the staging area are modified with respect to the **first version of the file** in the `.ogfiles` folder. It also checks for any new files which were created and committed for the first time. It then updates the `.git_log` file with these modified and added files, along with the commit message and the files which were modified or added in this commit, which is uniqely identified by the commit message, given by the user and the 16 character hash commit id, generated randomly using the `uuidgen` command. A folder with the commit-id as its name is created in the commits folder of the remote repository.

If any file in the staging area is not present in the `.ogfiles` folder, it adds it to the `.ogfiles` folder. It then copies the diff of the file from the `.ogfiles` folder and the file in the staging area into the commit-id folder.

It also gets the list of all csv files in the current repository at the time of the commit and stores their name in a hidden `.files` file in the commit-id folder. This provides a snapshot of all the files in the current repository at the time of the commit, which is useful at the time of `git\_checkout`.The stage is deleted after the commit is made.

Script files : git_commit.sh

5.10 Git Checkout

The command first checks if a remote repository has been initialized for the current directory. If it has not, the script prompts the user to initialize a remote repository first.
The user can checkout a particular commit by entering the commit-id (even entering the first few characters of the commit-id works too, if they are unique and not shared with any other commit-id) or the commit message. The script then parses and checks for matches to the commit-id or the commit message in the `.git_log` file. If it finds a match, it goes to the commit-id folder.// There, it gets the snapshot of the current repository at the time of the commit from the `.files` file. It also gets the list of patch files in the commit-id folder.

It deletes all the .csv files from the current directory and then iterates over all files in the snapshot. If the patch file exists for a particular file, it applies `patch -b` command to this patch file and the corresponding file in the `.ogfiles` folder, which gives back the version of the file at that commit as `<filename>.csv` and also the original file in the `.ogfiles` as `<filename>.csv.orig`. The .csv file is then copied from the `.ogfiles` folder to the current directory and the .csv.orig file is renamed to .csv file, hence restoring the original file in the `.ogfiles` folder.

Script files : git_checkout.sh

5.11 Git Log

The command first checks if the remote repository has been initialized for the current directory. If it has not, the script prompts the user to initialize a remote repository first. The script then reads the `.git_log` file and prints the commit history log in a readable format, with the commit-id, commit-message and time of commit. It also displays any new files created in the current directory, when they are committed for the first time, by the name **New files added:**, and if any files were modified from the time when the time the remote repository was initialized, it also shows them under the name **Files modified:**

While this command only displays the commit history log, it is actually the `git_commit` command which actually updates the `.git_log` file, at every commit.

Script files : git_log.sh

5.12 Git head

This command shows the present head of the current directory.

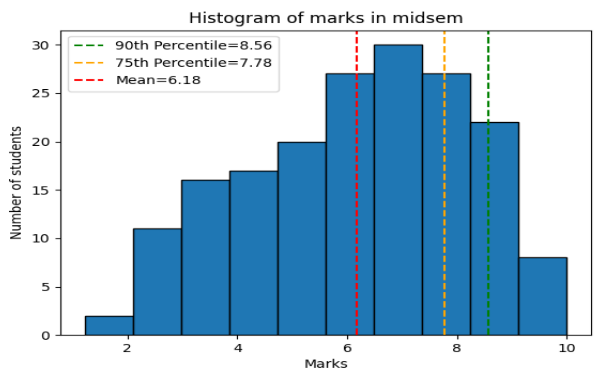
Script files : git_head.sh

5.13 Graphs

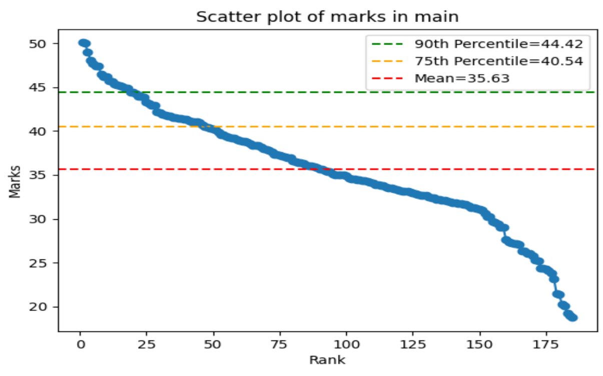
This command calls the `graphs.py` python script, which has a interface where users can choose between a histogram and a scatter plot. Then, the user can choose between a particular exam or `main` (for `main`, it considers the total marks for mark generation). It then calls the corresponding function from `graph_functions.py` to plot the graph, if the csv file of the exam exists.
The graphs also show the demarcation for 90th percentile, 75th percentile and mean marks.

Modules used for graph generation: os, csv, matplotlib

Script files: graphs.py, graph_functions.py



(a) Histogram for midsem marks



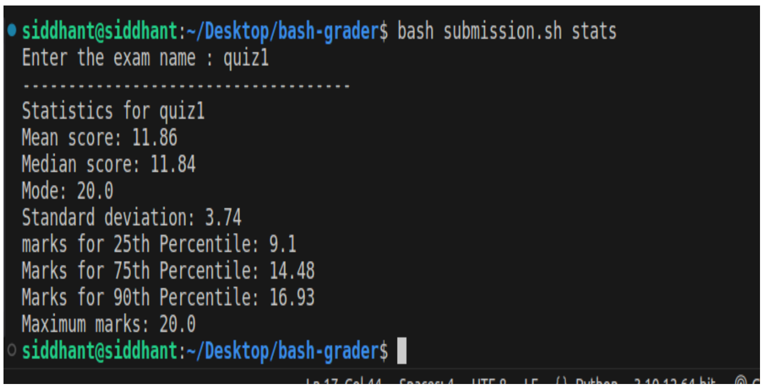
(b) Scatter plot for main.csv

5.14 Stats

This command calls the `stats.py` python script, which has a interface where users can choose between a particular exam or `main` (for `main`, it considers the total marks for mark generation). It then displays the stats for that particular exam or `main`, if the csv file for the exam exists.
The stats include the mean, median, mode, standard deviation, 25^{th} percentile, 75^{th} percentile, 90^{th} percentile and maximum marks.
All these stats are calculated by calling their respective functions from `stats_functions.py`.

Modules used for stats generation: os, csv, statistics

Script files: stats.py, stats_functions.py



```
siddhant@siddhant:~/Desktop/bash-grader$ bash submission.sh stats
Enter the exam name : quiz1
-----------------------------------
Statistics for quiz1
Mean score: 11.86
Median score: 11.84
Mode: 20.0
Standard deviation: 3.74
marks for 25th Percentile: 9.1
Marks for 75th Percentile: 14.48
Marks for 90th Percentile: 16.93
Maximum marks: 20.0
siddhant@siddhant:~/Desktop/bash-grader$
```

5.15 Report Card

This command calls the `report_card.py` python script, which has a interface where users can choose between a particular student or all students. It then displays the report card for that particular student or saves the pdf of all student report cards in the `report_cards` folder in the current directory.

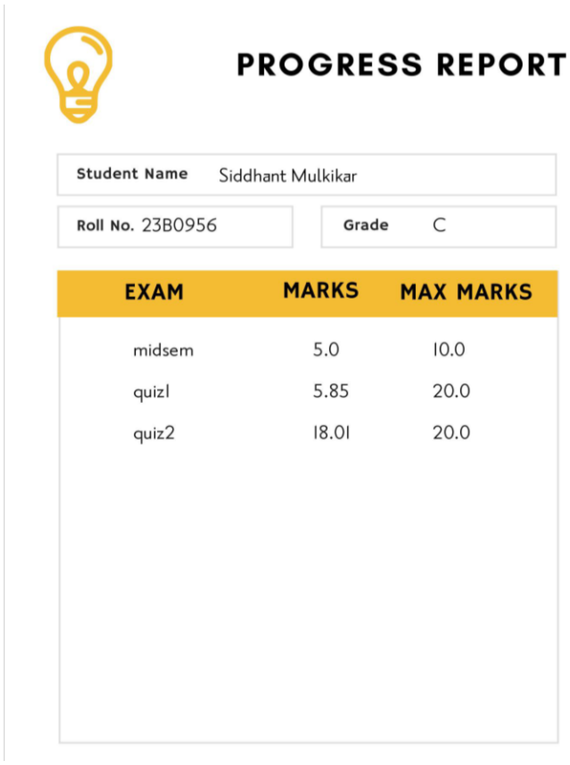
It gets the grades and percentile rubrics for the grades from the instructor by calling functions from the `grade.py` script. It then gets all the exams conducted and the maximum marks scored in each of these exams. Student marks for all students are also fetched by using the module `csv`.
It then calculates the weighted percentage of each student by using the formula:

$$W.P. = \frac{\sum_{i=1}^{n} M_i \times M_{max_i}}{\sum_{i=1}^{n} {M_{max_i}}^2} \times 100 \quad (1)$$

where M_i is the marks scored by the student in exam i, M_{max_i} is the maximum marks scored in exam i and n is the number of exams.

This is considering the maximum marks scored in each exam as the total marks for that exam. This ensures that a person with more marks in an exam with a greater weightage gets a higher weighted percentage.

The script then calculates the grade of each student by using the weighted percentage and the rubrics provided by the instructor. Then the percentiles of each student are calcualted with this weighted percentage using the `scipy` module. Then, the report card is generated from a template report card and text is put on the template using the `PIL` module and then converted to pdf form for all students, while it's only displayed for one person.



PROGRESS REPORT

Student Name	Siddhant Mulkikar
Roll No. 23B0956	**Grade** C

EXAM	MARKS	MAX MARKS
midsem	5.0	10.0
quiz1	5.85	20.0
quiz2	18.01	20.0

Modules used for report card generation: os, csv, statistics, scipy, PIL

Script files: report_card.py, grade.py

6 References

[1] Bash manual: `https://www.gnu.org/software/bash/manual/bash.html`

[2] Awk manual: `https://www.gnu.org/software/gawk/manual/gawk.html`

[3] Python manual: `https://docs.python.org/3/`

[4] Matplotlib manual: `https://matplotlib.org/`

[5] Scipy module: `https://www.scipy.org/`

[6] PIL module: `https://pillow.readthedocs.io/en/stable/`

[7] csv module: `https://docs.python.org/3/library/csv.html`

[8] os module: `https://docs.python.org/3/library/os.html`