Faculty of Engineering & Technology

Electrical & Computer Engineering Department

ENCS 3130: Linux Laboratory
Shell Project Report

Prepared by: Alaa M. Ahmed
ID: 1183339

And by: Anan Zein
ID: 1193078

Instructor Name: Dr. Mohammad Jubran

Assistant: Eng. Shahed Muhareb

Section: 3

Date: Thursday, 08/08/2021

Abstract

As the linux based systems are expanding world wide, the need to master shell scripting over linux based shells became important. A shell is a command-line interpreter with typical operations performed by shell scripts such as file manipulation, program execution, and printing text. In the shell script we can run several required constructs, or commands, that tell the shell environment what to do and when to do it. In this project, we took a deep dive in the shell scripting language and wrote a program that evaluates some statistics over a data set of csv format and reformat this data set to clear it from missing values or from errors.

Keywords: linux, shell, shell scripting, command

Table of Content

1. Introduction

Linux is a Unix-like Operating System. Its kernel was developed by Linus Torvalds in 1991. Since it is open-source it is freely available to download on the internet. It has many variants such as Ubuntu, Kali Linux, Fedora, Debian and many more. It is GNU licensed which means that it is free to purchase and redistribute, we even receive its source code and we can customize it to the way we want, we can alter the source code, add new features and create something new.

Shell is an interface between the operating system and the user. Whenever a user logs in or opens a terminal a new shell is launched by the operating system. It interprets the scripts which are basically programs used to create new and customized Linux commands.

There are two types of shells a) Command Line Shell b) Graphical Shell. In a Command- Line Shell, a terminal is available which is used to type commands for interacting with the system. In a Graphical Shell, windows are available which can be clicked, resized, expanded or minimized, etc. Most Linux variants come with both a graphical shell as well as Command Line Shell but Command-Line Shell remains more popular.

A shell program, sometimes referred to as a shell script, is simply a program constructed of shell commands. Shell programs are interpreted each time they are run. This means each command is processed (i.e. executed) by the shell a single line at a time. This is different from languages such as C or C++, which are translated in their entirety by a compiler program into a binary image. A shell program may be simple and consist of just a few shell commands, or it may be very complex and consist of thousands of shell commands. The complexity of the shell program is in the hand of the programmer. [1]

Due to its large applications in science, data manipulation, networking and in many other fields, it is important to understand the structure of the shell scripting to create a useful shell program. In this project, a shell script over a native linux environment was used to build a program that manipulates a dynamic-sized data set in (.csv) format by executing some commands that do some basic statistics and file correction.

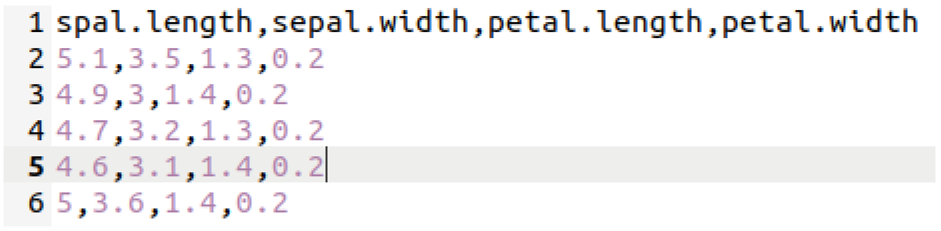
Program Structure

The structure of the program is simple and basic, it uses multiple commands from the native shell environment to manipulate the data file to work out some useful statistics about it. At first, the user is prompted to enter the data file name which he wants to process, the program reads that file name and searches for it in the local directory, if the file wasn't found, the program raises a warning and the user then is prompted to enter another file name. However, if the file exists, the program runs some commands to check the format of that file; the file is considered "correct", or in the right format, if it was a (.csv) file and it has the same number of columns in each row.

When the file is found and marked to be in the right format, the program asks the user what type of operation he wants to proceed with, the program gives the user three main options; D for finding the dimension of the program, C to run some statistics basic statistics for the data file and S to do basic file correction by substituting missing column values with the mean of that column. The user is always given the choice of existing the program by inputting E.

The program is structured to make all functions dependent on each other, that is, if you want to substitute missing values, the functions to find the dimension and the computing statistics will be executed if they weren't executed before. This led to reducing the lines of the program and thus the program needed less data and memory to operate and made it easier to understand its structure.

The input file, as mentioned before, should be in the csv format for the program to accept and process. The csv format is a well-known format for data sets, as it stores the data in a specific way. A csv file should contain a header first row which describes each column of data, it also specifies columns in each row by separating their values with a comma ",". Figure 1.1 shows an example of a csv formatted file.



```
spal.length,sepal.width,petal.length,petal.width
5.1,3.5,1.3,0.2
4.9,3,1.4,0.2
4.7,3.2,1.3,0.2
4.6,3.1,1.4,0.2
5,3.6,1.4,0.2
```

Figure 1.1: format of csv file

The program depends on parsing the data file to extract all the values from each column by using commands such as **awk**, **cut** or **sed**, which can be used with their different options to split data at a specific delimiter. For storing data, the program doesn't use any types of arrays or data structures since it's not an available option in native linux, however, it reads each column separately and processes it alone and shows its results to standard output, then it proceeds to other columns of data. This was found to be the best way to deal with column's data when it's not possible to save them in arrays.

The command **awk** was the most used command in the program, since it provides many useful applications in one small command. **awk** is an excellent tool for processing rows and columns, and is easier to use **awk** than most conventional programming languages. It can be considered to be a pseudo-C interpreter, as it understands the same arithmetic operators as C. **awk** also has string manipulation functions, so it can search for particular strings and modify the output. **awk** also has associative arrays, which are incredibly useful, and is a feature most computing languages lack. Associative arrays can make a complex problem a trivial exercise.

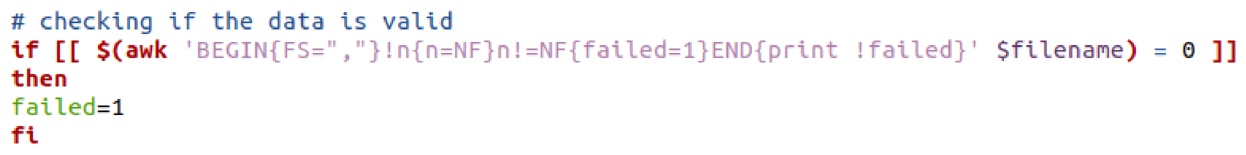
The essential organization of an **awk** program follows the form:

```
pattern { action }
```

The pattern specifies when the action is performed. Like most UNIX utilities, **awk** is line oriented. That is, the pattern specifies a test that is performed with each line read as input. If the condition is true, then the action is taken. The default pattern is something that matches every line. This is the blank or null pattern. Two other important patterns are specified by the keywords "BEGIN" and "END". As you might expect, these two words specify actions to be taken before any lines are read, and after the last line is read. The **awk** program below [2]:

```
BEGIN { print "START" }

      { print         }

END   { print "STOP"  }
```

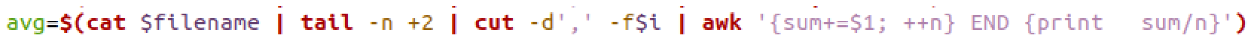
For example, the command shown in the figure 1.2 below, uses **awk** to check if the file format is valid or not. What it does is basically split the data using the Field Separator variable **FS** and then checks if the number of separated values (rows) are the same number in each column, to achieve that it uses the variable Number of Fields **NF**.



```
# checking if the data is valid
if [[ $(awk 'BEGIN{FS=","}!n{n=NF}n!=NF{failed=1}END{print !failed}' $filename) = 0 ]]
then
failed=1
fi
```

Figure 1.2: awk use in file validation

Another use of the **awk** is as shown in the figure 1.3 below, which finds the mean value of a specific column by finding the sum of all rows in that specific column and then dividing them by the Number of Fields.



```
avg=$(cat $filename | tail -n +2 | cut -d',' -f$i | awk '{sum+=$1; ++n} END {print  sum/n}')
```

Figure 1.3: awk use in computer statistics

This long introduction about the **awk** command is to elaborate the main purposes of the command and how it was used in different useful aspects in the program.

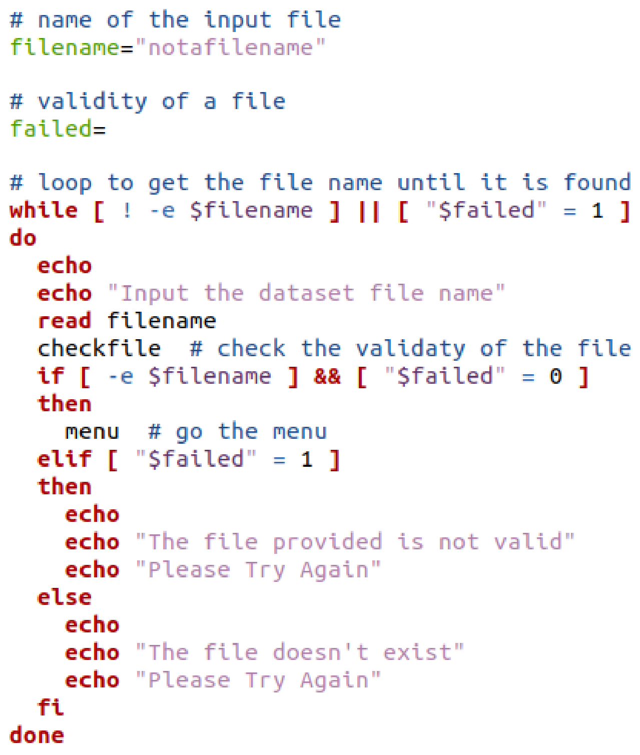
Other useful commands also were used such as **sed**, **tail** and **cut** with their different options. The command **sed** was used to find the number of columns in the data file as an easier way than **awk** by using the **-n** option related to it. The command **tail** was used heavily to eliminate the header row from the data to make the calculations easier by using the **-n** option related to it. Finally, the **cut** command was used to parse data as an easier way from **awk** with its option **-d**.

The program, as mentioned above, uses functions to make everything more clean and understandable instead of writing everything together which will make the program complex. It also initializes some global variables to make them accessible over all functions.

2. Code and Execution

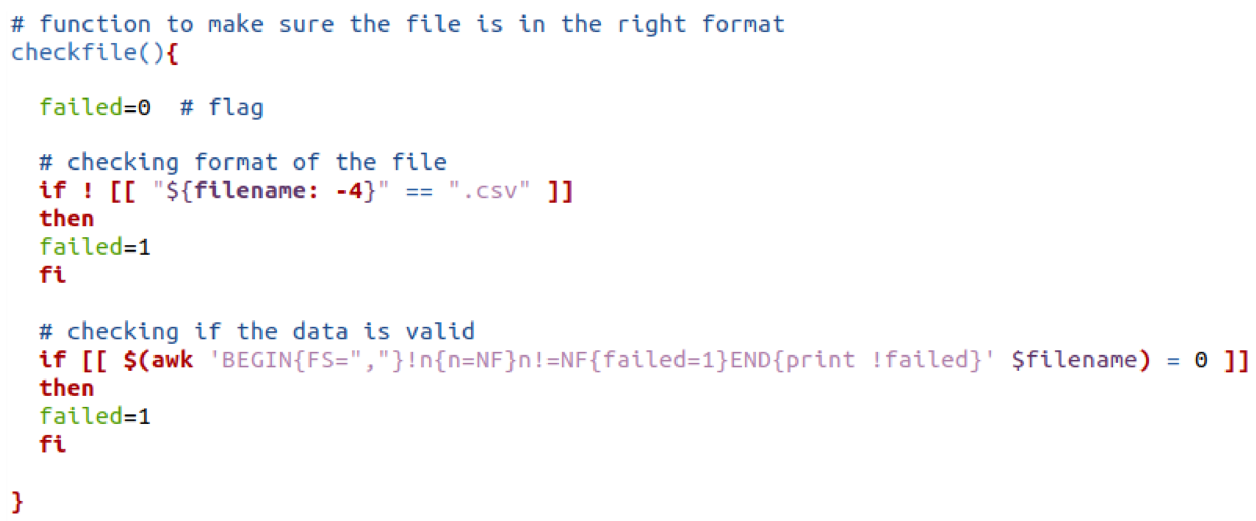
The code for the whole program can be found in the Appendix of this report.

The code was built to be as simple and clean as possible, to make it more understandable and easier to debug and fix. At first, the program prompts the user to enter the data input file name and checks if the file is valid or not; if it's not, the user must provide another filename for data input. Below are some figures showing the code and the execution of the file input portion.



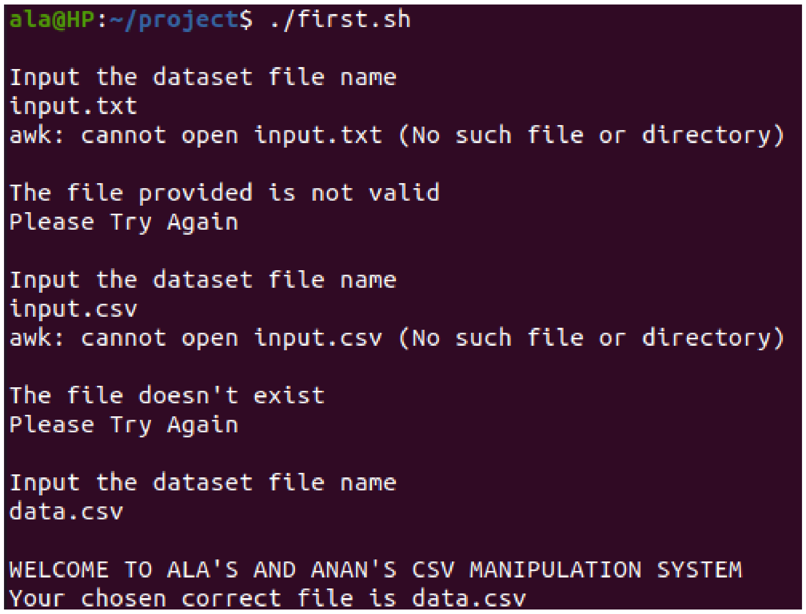
```
# name of the input file
filename="notafilename"

# validity of a file
failed=

# loop to get the file name until it is found
while [ ! -e $filename ] || [ "$failed" = 1 ]
do
  echo
  echo "Input the dataset file name"
  read filename
  checkfile  # check the validaty of the file
  if [ -e $filename ] && [ "$failed" = 0 ]
  then
    menu  # go the menu
  elif [ "$failed" = 1 ]
  then
    echo
    echo "The file provided is not valid"
    echo "Please Try Again"
  else
    echo
    echo "The file doesn't exist"
    echo "Please Try Again"
  fi
done
```

Figure 2.1.A: Code to read input file name from user



```
# function to make sure the file is in the right format
checkfile(){

  failed=0  # flag

  # checking format of the file
  if ! [[ "${filename: -4}" == ".csv" ]]
  then
  failed=1
  fi

  # checking if the data is valid
  if [[ $(awk 'BEGIN{FS=","}!n{n=NF}n!=NF{failed=1}END{print !failed}' $filename) = 0 ]]
  then
  failed=1
  fi

}
```

Figure 2.1.B: Code to check the validity of the input file



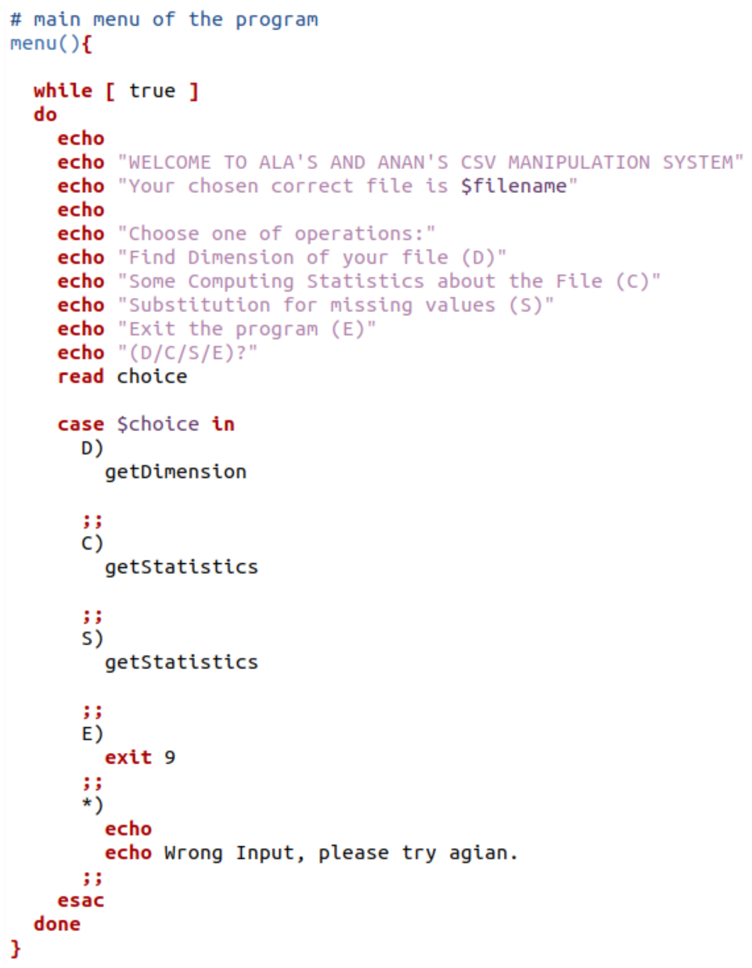
```
ala@HP:~/project$ ./first.sh

Input the dataset file name
input.txt
awk: cannot open input.txt (No such file or directory)

The file provided is not valid
Please Try Again

Input the dataset file name
input.csv
awk: cannot open input.csv (No such file or directory)

The file doesn't exist
Please Try Again

Input the dataset file name
data.csv

WELCOME TO ALA'S AND ANAN'S CSV MANIPULATION SYSTEM
Your chosen correct file is data.csv
```

Figure 2.1.C: Sequential test cases to get the right file name from user

In this way, the program makes sure from the user that it is dealing with the right input file and not a corrupted file or a non-supported format file. As shown in figures 2.1.A and 2.1.B, this was done by a while loop that executes as long as the file is not found (done by using the **-e** option) or found to be corrupted or supported (done by executing the **checkfile** function and changing the value of the flag **failed**).

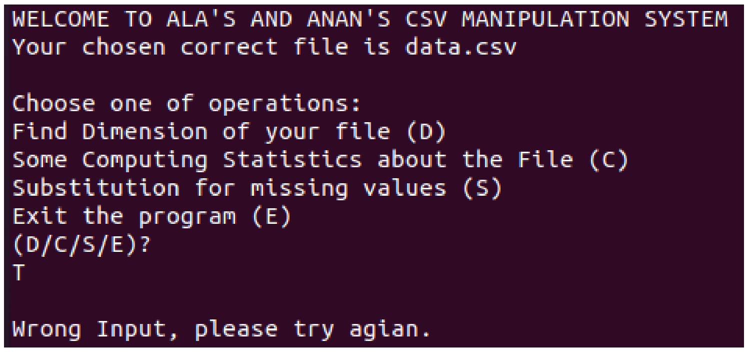
In the **checkfile** function, the program runs the comparison operation **if ! [["${filename: -4}" == ".csv"]]** to check if the last 4 characters in the file name are ".csv" and it runs the **awk** command **if [[$(awk 'BEGIN{FS=","}!n{n=NF}n!=NF{failed=1}END{print !failed}' $filename) = 0]]** to check if the number of columns are the same for all rows.

The user is then given the main menu of the program in the **menu** function, the menu is in an infinite loop that ends by the user when writing **E**, in this loop the user is given multiple options and prompted to choose one of them, if the user wrote wrong value, the program will raise a warning and prompt the user to enter another input. All shown in the figure below.



```
# main menu of the program
menu(){

  while [ true ]
  do
    echo
    echo "WELCOME TO ALA'S AND ANAN'S CSV MANIPULATION SYSTEM"
    echo "Your chosen correct file is $filename"
    echo
    echo "Choose one of operations:"
    echo "Find Dimension of your file (D)"
    echo "Some Computing Statistics about the File (C)"
    echo "Substitution for missing values (S)"
    echo "Exit the program (E)"
    echo "(D/C/S/E)?"
    read choice

    case $choice in
      D)
        getDimension

      ;;
      C)
        getStatistics

      ;;
      S)
        getStatistics

      ;;
      E)
        exit 9
      ;;
      *)
        echo
        echo Wrong Input, please try agian.
      ;;
    esac
  done
}
```

Figure 2.2.A: Code to display the main menu of the program

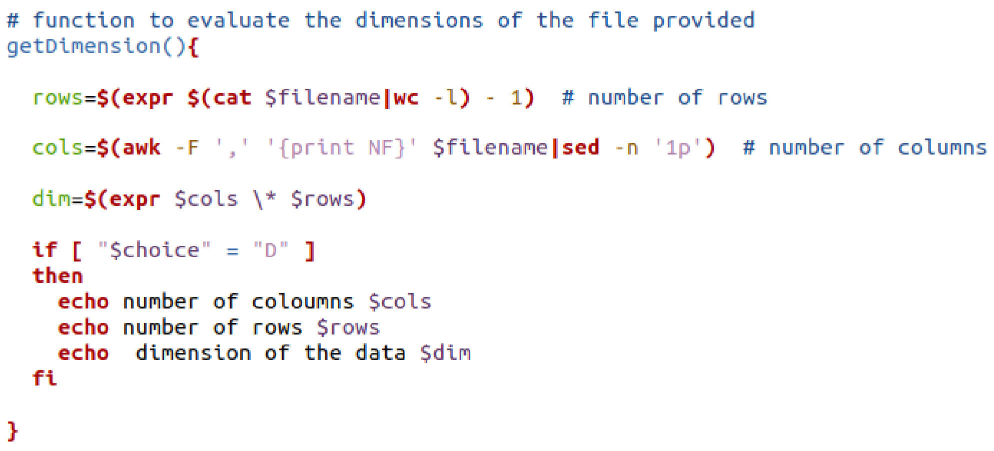


```
WELCOME TO ALA'S AND ANAN'S CSV MANIPULATION SYSTEM
Your chosen correct file is data.csv

Choose one of operations:
Find Dimension of your file (D)
Some Computing Statistics about the File (C)
Substitution for missing values (S)
Exit the program (E)
(D/C/S/E)?
T

Wrong Input, please try agian.
```

Figure 2.2.B: Stdout to show how the menu appears to the user

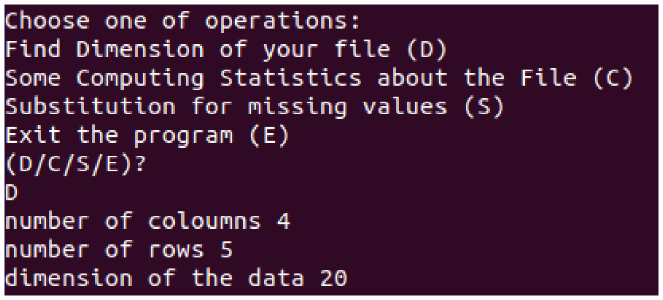
The user is prompted to enter one of the four choices: D, C, S or E. Then the program reads the user input and decides using the **case** command which function it should forward the user to.

If the user input was D, the program executes the function **getDimension** which finds the number of rows and columns of the program. This function is essential as it calculates useful information from the data file that can be used in all next functions. Below are figures.



```
# function to evaluate the dimensions of the file provided
getDimension(){

  rows=$(expr $(cat $filename|wc -l) - 1)  # number of rows

  cols=$(awk -F ',' '{print NF}' $filename|sed -n '1p')  # number of columns

  dim=$(expr $cols \* $rows)

  if [ "$choice" = "D" ]
  then
    echo number of coloumns $cols
    echo number of rows $rows
    echo  dimension of the data $dim
  fi

}
```

Figure 2.3.A: Code to find the dimensions of the data file

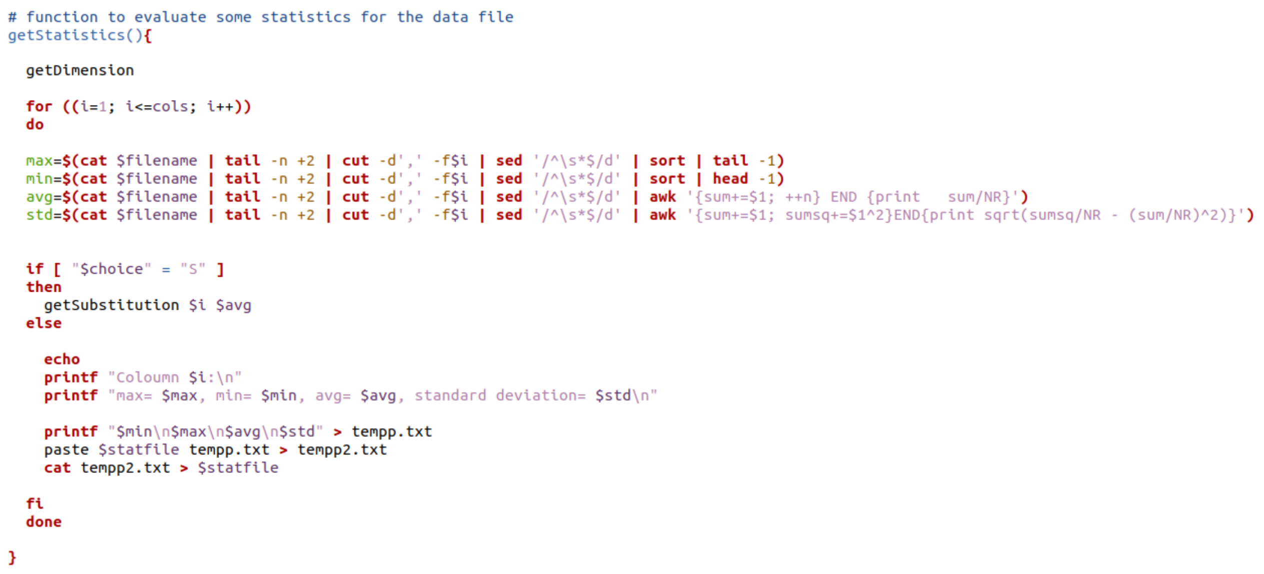


```
Choose one of operations:
Find Dimension of your file (D)
Some Computing Statistics about the File (C)
Substitution for missing values (S)
Exit the program (E)
(D/C/S/E)?
D
number of coloumns 4
number of rows 5
dimension of the data 20
```

Figure 2.3.B: Output of the ***getDimension*** *function*

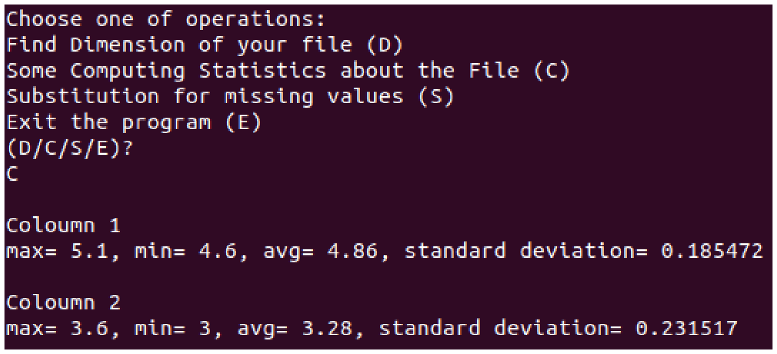
The rows are found by the command **wc** with its option **-l** which returns the number of lines in the file. The columns are found by using the **NF** variable in the **awk** command. The dimension is found by executing an expression of the rows*columns using **expr** command.

If the user input was C, the program runs the function **getStatistics** which does some basic statistics on the data as the max, min, average and standard deviation values in each column.



```
# function to evaluate some statistics for the data file
getStatistics(){

  getDimension

  for ((i=1; i<=cols; i++))
  do

  max=$(cat $filename | tail -n +2 | cut -d',' -f$i | sed '/^\s*$/d' | sort | tail -1)
  min=$(cat $filename | tail -n +2 | cut -d',' -f$i | sed '/^\s*$/d' | sort | head -1)
  avg=$(cat $filename | tail -n +2 | cut -d',' -f$i | sed '/^\s*$/d' | awk '{sum+=$1; ++n} END {print   sum/NR}')
  std=$(cat $filename | tail -n +2 | cut -d',' -f$i | sed '/^\s*$/d' | awk '{sum+=$1; sumsq+=$1^2}END{print sqrt(sumsq/NR - (sum/NR)^2)}')


  if [ "$choice" = "S" ]
  then
    getSubstitution $i $avg
  else

    echo
    printf "Coloumn $i:\n"
    printf "max= $max, min= $min, avg= $avg, standard deviation= $std\n"

    printf "$min\n$max\n$avg\n$std" > tempp.txt
    paste $statfile tempp.txt > tempp2.txt
    cat tempp2.txt > $statfile

  fi
  done

}
```

Figure 2.4.A: Code to find some basic statistics on each column of data



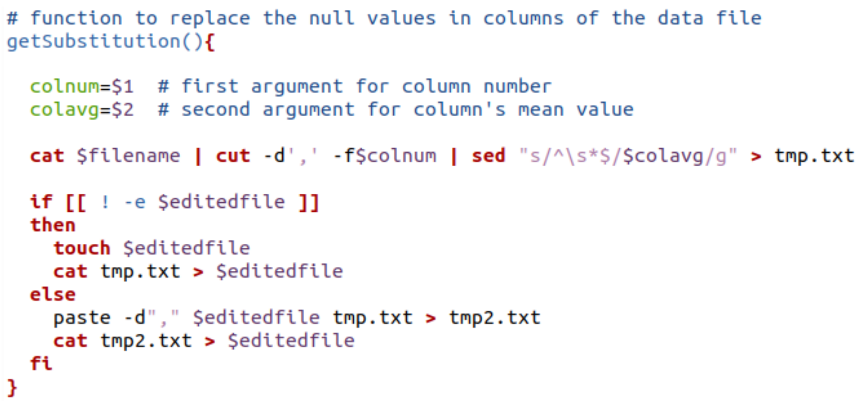
```
Choose one of operations:
Find Dimension of your file (D)
Some Computing Statistics about the File (C)
Substitution for missing values (S)
Exit the program (E)
(D/C/S/E)?
C

Coloumn 1
max= 5.1, min= 4.6, avg= 4.86, standard deviation= 0.185472

Coloumn 2
max= 3.6, min= 3, avg= 3.28, standard deviation= 0.231517
```

*Figure 2.4.B: Sample data output from the **getStatistics** function*

It's noticeable that the **getStatistics** function first calls the **getDimension** function to make sure that the number of columns and rows are already known before starting to evaluate statistics.

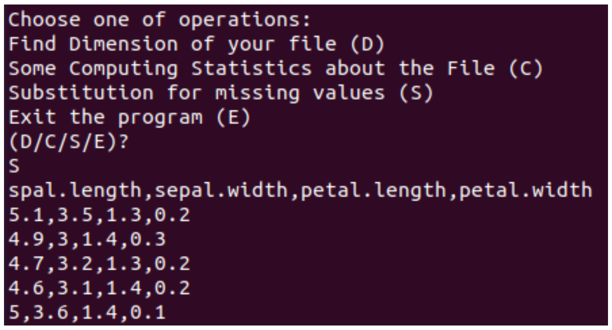
After delimiting each column, **tail -n +2** eliminates the header file and then the **sed** command shown removes any unwanted white spaces from the column's data. The max and min values are found by the **sort** command. The average is found by a simple **awk** command that takes the sum of all values in a column and divides them by the number of fields. The standard deviation is a more complex mathematical operation but it has also been executed using the **awk** command.

If the user chooses the input S it will run the **getSubstitution** function, which searches for the missing values in each column of data and replaces it with the average value of that function, this function takes two arguments; the column number and the average value of that column.



```
# function to replace the null values in columns of the data file
getSubstitution(){

  colnum=$1  # first argument for column number
  colavg=$2  # second argument for column's mean value

  cat $filename | cut -d',' -f$colnum | sed "s/^\s*$/$colavg/g" > tmp.txt

  if [[ ! -e $editedfile ]]
  then
    touch $editedfile
    cat tmp.txt > $editedfile
  else
    paste -d"," $editedfile tmp.txt > tmp2.txt
    cat tmp2.txt > $editedfile
  fi
}
```

Figure 2.5.A: Code to substitute missing values in the file



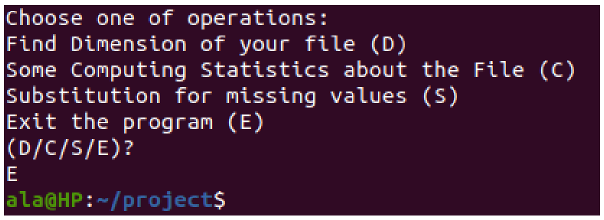
```
Choose one of operations:
Find Dimension of your file (D)
Some Computing Statistics about the File (C)
Substitution for missing values (S)
Exit the program (E)
(D/C/S/E)?
S
spal.length,sepal.width,petal.length,petal.width
5.1,3.5,1.3,0.2
4.9,3,1.4,0.3
4.7,3.2,1.3,0.2
4.6,3.1,1.4,0.2
5,3.6,1.4,0.1
```

*Figure 2.5.A: Sample data output of the **getSubsitution** function*

The missing values in the columns was replaced using the command **sed "s/^\s*$/$colavg/g"** which took every column and inspected the existence of white spaces in it to replace it with that column's average value. The **getSubsitution** function is executed with the **getStatistics** function to get the average of each column with its index. **getSubsitution** takes two arguments, the column index and its average value.

All the output data from the **getSubsitution** and **getStatistics** functions are saved into new output files to make it easily accessible for later. The new files are **edited_$filename** which saves the data after substituting missing values and **stat_$filename** which saves the statistics related to data.

The program ends only if the user worte E as input, this will execute the **exit** command, which will prompt the program to enter the exit status of its process.



```
Choose one of operations:
Find Dimension of your file (D)
Some Computing Statistics about the File (C)
Substitution for missing values (S)
Exit the program (E)
(D/C/S/E)?
E
ala@HP:~/project$
```

Figure 2.6: Exiting the program

3. Conclusion

In this project, we implemented a data manipulation system to evaluate statistics and provide file correction for any data set in the csv format. The project was implemented over shell scripting language in native linux environment. In this report we explained the structure of the program first theoretically then by code and by running test cases over it.

References

[1] A. Kidwai et al., “A comparative study on shells in Linux: A review,” Materials Today: Proceedings, vol. 37, pp. 2612–2616, 2021, doi: 10.1016/j.matpr.2020.08.508.

[2] “The Grymoire’s tutorial on AWK,” *Grymoire.com*, 2020. https://www.grymoire.com/Unix/Awk.html (accessed Aug. 11, 2021).

Appendix

```
#Alaa Ahmed 1183339

#Anan Zein 1193078


#LINUX LABARTORY ENCS3130

#DR.MOHAMMAD JUBRAN




# function to replace the null values in columns of the data file

getSubstitution(){


      colnum=$1  # first argument for column number

      colavg=$2  # second argument for column's mean value


  cat $filename | cut -d',' -f$colnum | sed "s/^\s*$/$colavg/g" > tmp.txt


  if [[ ! -e $editedfile ]]

  then

      touch $editedfile

      cat tmp.txt > $editedfile

  else

      paste -d"," $editedfile tmp.txt > tmp2.txt

      cat tmp2.txt > $editedfile

  fi

}


# function to evaluate some statistics for the data file

getStatistics(){
      getDimension


      for ((i=1; i<=cols; i++))
      do


      max=$(cat $filename | tail -n +2 | cut -d',' -f$i | sed '/^\s*$/d' | sort
| tail -1)
      min=$(cat $filename | tail -n +2 | cut -d',' -f$i | sed '/^\s*$/d' | sort
| head -1)
      avg=$(cat $filename | tail -n +2 | cut -d',' -f$i | sed '/^\s*$/d' | awk
'{sum+=$1; ++n} END {print     sum/NR}')
      std=$(cat $filename | tail -n +2 | cut -d',' -f$i | sed '/^\s*$/d' | awk
'{sum+=$1; sumsq+=$1^2}END{print sqrt(sumsq/NR - (sum/NR)^2)}')



      if [ "$choice" = "S" ]
      then
          getSubstitution $i $avg
      else


          echo
          printf "Coloumn $i:\n"
          printf "max= $max, min= $min, avg= $avg, standard deviation=
$std\n"


          printf "$min\n$max\n$avg\n$std" > tempp.txt
          paste $statfile tempp.txt > tempp2.txt
          cat tempp2.txt > $statfile
```

```
	fi
	done

}

# function to evaluate the dimensions of the file provided
getDimension(){

	rows=$(expr $(cat $filename|wc -l) - 1)  # number of rows

	cols=$(awk  -F  ','  '{print  NF}'  $filename|sed  -n  '1p')   #  number  of
columns

	dim=$(expr $cols \* $rows)

	if [ "$choice" = "D" ]
	then
		echo number of coloumns $cols
		echo number of rows $rows
		echo  dimension of the data $dim
	fi

}

# main menu of the program
menu(){

	while [ true ]
	do
```

```
echo
echo "WELCOME TO ALA'S AND ANAN'S CSV MANIPULATION SYSTEM"
echo "Your chosen correct file is $filename"
echo
echo "Choose one of operations:"
echo "Find Dimension of your file (D)"
echo "Some Computing Statistics about the File (C)"
echo "Substitution for missing values (S)"
echo "Exit the program (E)"
echo "(D/C/S/E)?"
read choice

case $choice in
      D)
            getDimension

      ;;
      C)
            if [[ ! -e $statfile ]]
            then
                  touch $statfile
            fi
            printf "Min\nMax\nMean\nSTDEV" > $statfile

            getStatistics

            printf "\ndata is availabe at $statfile\n"
            rm tempp.txt
```

```
				rm tempp2.txt


			;;

			S)

				getStatistics

				cat $editedfile

				rm tmp.txt

				rm tmp2.txt

				printf "\ndata is available at $editedfile\n"


			;;

			E)

				exit 9

			;;

			*)

				echo

				echo Wrong Input, please try agian.

			;;

		esac

	done

}




# function to make sure the file is in the right format

checkfile(){


	failed=0  # flag
```

```
      # checking format of the file

      if ! [[ "${filename: -4}" == ".csv" ]]

      then

      failed=1

      fi


      # checking if the data is valid

      if   [[   $(awk   'BEGIN{FS=","}!n{n=NF}n!=NF{failed=1}END{print   !failed}'
$filename) = 0 ]]

      then

      failed=1

      fi


}




#

# Main

#


# dimension of the data file

rows=

cols=

dim=


# menu choice

choice=


# name of the input file
```

```
filename="notafilename"


# validity of a file

failed=


# loop to get the file name until it is found

while [ ! -e $filename ] || [ "$failed" = 1 ]

do

      echo

      echo "Input the dataset file name"

      read filename

      checkfile  # check the validaty of the file

      if [ -e $filename ] && [ "$failed" = 0 ]

      then

            editedfile="edited_$filename"

            statfile="stat_$filename"

        menu  # go the menu

      elif [ "$failed" = 1 ]

      then

            echo

            echo "The file provided is not valid"

            echo "Please Try Again"

      else

        echo

        echo "The file doesn't exist"

        echo "Please Try Again"

      fi

done
```