Hacettepe University
Department of Computer Engineering

BBM 105 Assignment 2 Report

Umut Bayındır - 2210356147

I.Table of Contents

1.Introduction

In this assignment, we were tasked with making a clinical decision support system(CDSS), which is basically a health information technology that provides clinicians, staff, patients or the other individuals with knowledge and person-specific information, to help health and health care. In this specific instance, our task was to make a simple CDSS on the field of oncology that helps the doctors with keeping a list of potential cancer patients and their information which they can access at any time and add or remove patients. The other goals were making a function calculating the actual probability of the patient having cancer and another function recommending or not recommending the treatment procedure.

2.OS Functions

When we get to the specifics of the assignment, we were tasked with reading inputs from a text file named doctors_aid_inputs.txt, processing the commands in python language and then outputting the results to another text file, doctors_aid_outputs.txt.

2.1.fread()

I started by coding fread() and fwrite() functions.. I put f at the start for ‘function’ to distinguish them from read() and write() methods also used in the code for fread() and fwrite() respectively. To read and write a text file on python os library is required, so I used import command to access it. Then with the aid of a py file sent by the bbm103 teachers to showcase the read() and write() methods, I wrote the both functions. Starting with ‘current_dir_path = os.getcwd()’, which allows the python script to access files in the same directory. Then I created three lists in a row, namely commands patientinfo and output, which will play key parts later. Then I started writing fread(). I used ‘reading_file_name = “doctors_aid_inputs.txt”’ and ‘reading_file_path = os.path.join(current_dir_path, reading file_name)’ to look for doctors_aid_inputs.txt and search the directory for it respectively. Then I wrote with ‘open(reading_file_path, “r”) as i:’ to start reading the file. Then I made a while cycle and used readline() to scan all the lines one by one and ‘commands.append(line)’ to carry the lines to the aforementioned commands list.

```
1   import os
2   current_dir_path = os.getcwd()
3   commands=[]
4   patientinfo=[]
5   output=[]
6   def fread():
7       reading_file_name = "doctors_aid_inputs.txt"
8       reading_file_path = os.path.join(current_dir_path, reading_file_name)
9       with open(reading_file_path, "r") as i:
10                  count = 0
11                  while True:
12                      count += 1
13                      line = i.readline()
14                      if not line:
15                          break
16                      commands.append(line)
17                  i.close()
```

2.2.fwrite()

Next step was the fwrite(). It is basically very similar to fread(), but basically all reading_ in items were replaced with writing_, as well as “r” was shifted with “w”. Instead of a while cycle, I used a for cycle to transfer the items in output list to stroutput string to write the contents of the list.

```
18  def fwrite():
19      writing_file_name = "doctors_aid_outputs.txt"
20      writing_file_path = os.path.join(current_dir_path, writing_file_name)
21      with open(writing_file_path,"w") as o:
22          stroutput=""
23          for x in output:
24              stroutput += x
25          o.write(stroutput)
26          o.close()
```

3.Executing Commands

3.1.Scrapping the Commands

In this step, I started by calling the fread() function and then created a chain to scrap the commands to words by using a series of temporary lists, strings and for cycles to turn the lists to strings to be used with split() method, which turned the product to a list, which needed to be turned to string to progress and so on. In first step, I used splitlines() to get rid off the “\n”s(line symbols) at the end,which prevented a patient’s last information mixing with another patient’s first information. I originally used split(“\n”), but then later switched it to splitlines(), since it practically did the same. Next step was split(“,”), which divided each information. This step was primarily used when adding a patient to the list and was skipped in other commands, since other commands lacked commas. Last step was split(“ “), which divided the commands and called patient’s name.

```
147  fread()
148  for i in range(len(commands)):
149      strstep1=""
150      strstep2=""
151      strstep3=""
152      step1=commands[i]
153      for x in step1:
154          strstep1+=x
155      step2=strstep1.splitlines()
156      for x in step2:
157          strstep2+=x
158      step3=strstep2.split(",")
159      for x in step3:
160          strstep3+=x
161      data=strstep3.split(" ")
```

```
probability Hayriye
recommendation Ateş
create Toprak, 0.98, Prostate Cancer, 21/100000, Hormonotherapy, 0.20
```

example of commands from doctors_aid_inputs.txt

3.2.Calling the Functions

This part was critical for functions to work. First of all, I turned the commands in the scrapped data to their own string group called funct, shortened version of function, to avoid confusion with the built-in class. Then, I made an if chain to separate each type of command to their respective functions. It's worth noting that miswritten commands will not work. 'data.pop(0)' removes the commands from the data and 'funct=""' makes sure commands don't repeat in the for cycle. 'data.clear()' removes the data to prevent it from clogging the process in future operations. fwrite() after the for cycle marks the end of the script.

```
    funct=data[0]
    if funct == "create":
        funct=""
        data.pop(0)
        create()
    if funct == "remove":
        funct=""
        data.pop(0)
        remove()
    if funct == "list":
        funct=""
        data.pop(0)
        flist()
    if funct == "probability":
        funct=""
        data.pop(0)
        probability()
    if funct == "recommendation":
        data.pop(0)
        funct=""
        recommendation()
    data.clear()
fwrite()
```

4.Command Functions

I wrote the command functions along with earlier OS functions to make the checking process efficient. All command functions behave similarly. They first check if the list is empty and then check if the name from the inputs match any of the names recorded before. They take the names from patientinfo by spliting the desired list element to its words and then taking the first word, which is the patient name. This part took me a lot of time and originally didn't work for many attempts. I originally used 'if name in patientname:' first and then I tried using 'if patienname.find(name)!=-1:', thinking there might have been a problem about the use, but then my friends told me it didn't work due to the entire function being part of the for loop and should be divided to two parts with one being in for loop and the other outside of it. So I wrote 'isFound=True' after the if part and then added 'if isFound is true:' which ended up working.

```
[['Hayriye', '0.999', 'Breast', 'Cancer', '50/100000', 'Surgery', '0.40'], ['Deniz', '0.9999', 'Lung', 'Cancer', '40/100000', 'Radiotherapy', '0.50'], ['Toprak', '0.98', 'Prostate', 'Cance
r', '21/100000', 'Hormonotherapy', '0.20'], ['Hypatia', '0.9975', 'Stomach', 'Cancer', '15/100000', 'Immunotherapy', '0.04'], ['Pakiz', '0.9997', 'Colon', 'Cancer', '14/100000', 'Targeted'
, 'Therapy', '0.30'], ['Su', '0.98', 'Breast', 'Cancer', '50/100000', 'Chemotherapy', '0.20']]
```

This is how patientinfo looks in python. It basically contains all the information related to a patient in a single element. Functions take one element at a time and split the element to its words.

4.1.create()

create() is probably the second most important function after fread(), since it allows to add patient information to the patientinfo list, which is the key to the other command functions. This function checks if the entered name is saved before and adds the name and other information if the name isn't found. It's designed to not allow duplication of names, even if other information is different.

```
27  def create():
28      name=data[0]
29      patientname=""
30      if len(patientinfo)==0:
31          patientinfo.append(data.copy())
32          output.append("Patient {} is recorded.\n".format(name))
33      else:
34          for i in range(len(patientinfo)):
35              patientname=patientinfo[i][0]
36              if name in patientname:
37                  isFound=True
38                  break
39              else:
40                  isFound=False
41                  continue
42          if isFound is True:
43              output.append("Patient {} cannot be recorded due to duplication.\n".format(name))
44          else:
45                  patientinfo.append(data.copy())
46                  output.append("Patient {} is recorded.\n".format(name))
```

4.2.remove()

remove() deletes the information entered before if it's saved under the entered patient name. It is practically coded opposite to create(), as it doesn't work if the entered name isn't found, but works if the name is found.

```
47  def remove():
48      name=data[0]
49      patientname=""
50      if len(patientinfo)==0:
51          output.append("Patient {} cannot be removed due to absence.\n".format(name))
52      else:
53          for i in range(len(patientinfo)):
54              patientname=patientinfo[i][0]
55              if name in patientname:
56                  isFound=True
57                  break
58              else:
59                  isFound=False
60                  continue
61          if isFound is True:
62                  patientinfo.pop(i)
63                  output.append("Patient {} is removed.\n".format(name))
64          else:
65                  output.append("Patient {} cannot be removed due to absence.\n".format(name))
```

4.3.flist()

flist() basically shows the information collected under the patientinfo list in an easy to read way. ‘f’ at the start stands for function, similar to fread(), fwrite() and funct, it’s written in this way to prevent the interpreter from confusing the command with the list class. This function works by taking six of the words in a patientinfo element instead of just the name and then write all the elements one line for one element at a time. It turns diagnosis accuracy and treatment risk to percentage. They are needed to be turned into floats first, since patientinfo kept them as strings. I cut corners by not taking the word ‘cancer’ from patientinfo and instead wrote it in output.append. When I originally wrote the function, I didn’t notice that one of the treatment types, ‘targeted therapy’ was two words. This caused an issue by the word ‘therapy’ replacing the treatment risk. I solved this issue by adding an if condition changing the searched risk element to the next element if the treatment type said ‘targeted’. I also wrote to change the treatment name to ‘targeted therapy’ in that specific situation. flist() was the easiest function to make overall.

```
66  def flist():
67      output.append("Patient Diagnosis\tDisease \t\tDisease \tTreatment\t\tTreatment\n")
68      output.append("Name\tAccuracy\tName\t\t\tIncidence\tName\t\t\tRisk\n")
69      output.append("-------------------------------------------------------------------------\n")
70      for i in range(len(patientinfo)):
71          patientname=patientinfo[i][0]
72          accuracy=patientinfo[i][1]
73          acc=float(accuracy)
74          oacc=acc*100
75          cancer=patientinfo[i][2]
76          incidence=patientinfo[i][4]
77          treatment=patientinfo[i][5]
78          if treatment=="Targeted":
79              risk=patientinfo[i][7]
80              treatment="Targeted Therapy"
81              frisk=float(risk)
82              ofrisk=frisk*100
83              output.append("{}\t{}%\t\t{} Cancer\t\t{}\t{}\t{}%\n".format(patientname,oacc,cancer,incidence,treatment,ofrisk))
84          else:
85              risk=patientinfo[i][6]
86              frisk=float(risk)
87              ofrisk=frisk*100
88              output.append("{}\t{}%\t\t{} Cancer\t\t{}\t{} \t\t{}%\n".format(patientname,oacc,cancer,incidence,treatment,ofrisk))
```

```
Patient Diagnosis       Disease                 Disease         Treatment               Treatment
Name    Accuracy        Name                    Incidence       Name                    Risk
-------------------------------------------------------------------------
Hayriye 99.9%           Breast Cancer           50/100000       Surgery                 40.0%
Deniz   99.99%          Lung Cancer             40/100000       Radiotherapy            50.0%
Ateş    99.0%           Thyroid Cancer          16/100000       Chemotherapy            2.0%
Toprak  98.0%           Prostate Cancer         21/100000       Hormonotherapy          20.0%
Hypatia 99.75%          Stomach Cancer          15/100000       Immunotherapy           4.0%
Pakiz   99.97%          Colon Cancer            14/100000       Targeted Therapy        30.0%
```

an example list from doctors_aid_outputs.txt

4.4.probability()

probability() was the hardest function to write. This was mainly because it uses a formula I wasn't familiar until the preparation for the assignment. probability() basically calculates the actual possibility of the patient suffering from cancer. This value is calculated by subtracting the diagnosis accuracy from 1, then multiplying it with the denominator of the incidence, incidence is the prevalence of that specific disease in the local area, and then adding the numerator of the incidence, after that dividing the numerator to the result up until now. I turned this formula to 'probab=(inci1/((1-acc)*100000+inci1))' as you can see below. When making the function, I wrote it the way so it turns the result to percentage and then rounds it to maximum of two decimals. Similar to flist(), the numbers used in this function are needed to be turned into floats or integers since they were kept as strings in patientinfo. I also cut corners by not using the denominator, since the incidence was based on the same area (Turkey) and used the same denominator, 100000.

```
91  def probability():
92      name=data[0]
93      patientname=""
94      incidence=""
95      cancer=""
96      if len(patientinfo)==0:
97          output.append("Probability for {} cannot be calculated due to absence.\n".format(name))
98      else:
99          for i in range(len(patientinfo)):
100             patientname=patientinfo[i][0]
101             if name in patientname:
102                 isFound=True
103                 break
104             else:
105                 isFound=False
106                 continue
107         if isFound is True:
108             accuracy=patientinfo[i][1]
109             acc=float(accuracy)
110             cancer=patientinfo[i][2]
111             cancerl=cancer.lower()
112             incidence=patientinfo[i][4]
113             inci=incidence.split("/")[0]
114             inci1=int(inci)
115             probab=(inci1/((1-acc)*100000+inci1))*100
116             oprobab=round(probab,2)
117             output.append("Patient {} has a probability of {}% of having {} cancer.\n".format(name,oprobab,cancerl))
118         else:
119             output.append("Probability for {} cannot be calculated due to absence.\n".format(name))
```

```
Patient Pakiz has a probability of 31.82% of having colon cancer.
```

an example of the use of the probability() function.

4.5.recommendation()

recommendation() is simply the continuation of the probability(). It calculates the probability and then compares it to the treatment risk. If the risk is higher than probability, it doesn't recommend the treatment. If the risk is lower, it recommends the treatment.

```
120  def recommendation():
121      name=data[0]
122      patientname=""
123      if len(patientinfo)==0:
124          output.append("Recommendation for {} cannot be calculated due to absence.\n".format(name))
125      else:
126          for i in range(len(patientinfo)):
127              patientname=patientinfo[i][0]
128              if name in patientname:
129                  isFound=True
130                  break
131              else:
132                  isFound=False
133                  continue
134          if isFound is True:
135                  accuracy=patientinfo[i][1]
136                  acc=float(accuracy)
137                  incidence=patientinfo[i][4]
138                  risk=patientinfo[i][6]
139                  r=float(risk)
140                  inci=incidence.split("/")[0]
141                  inci1=int(inci)
142                  probab=inci1/((1-acc)*100000+inci1)
143                  if r>=probab:
144                      output.append("System suggests {} NOT to have the treatment.\n".format(name))
145                  else:
146                      output.append("System suggests {} to have the treatment.\n".format(name))
147          else:
148              output.append("Recommendation for {} cannot be calculated due to absence.\n".format(name))
```

```
Patient Diagnosis       Disease                Disease         Treatment              Treatment
Name    Accuracy        Name                   Incidence       Name                   Risk
-------------------------------------------------------------------------------
Su      98.0%           Breast Cancer          50/100000       Chemotherapy           20.0%

        Patient Su has a probability of 2.44% of having breast cancer.
```

For example, let's say the values and the probability() example above belongs to Su. Chemotherapy has about 20% risk. Probability is on the other hand 2.44%, way below 20%, which doesn't seem to be worth the risk. In this situation, function would return the following:

```
System suggests Su NOT to have the treatment.
```

5. Conclusion and Grading

This assignment showcases how interpreter languages such as python can be used to make programs for vital purposes. It also taught me when to use which methods under which conditions and how to use them effectively.

Evaluation	Points	Evaluate Yourself / Guess Grading
Indented and Readable Codes	5	5
Using Meaningful Naming	5	5
Using Explanatory Comments	5	5
Efficiency (avoiding unnecessary actions)	5	2
Function Usage	25	20
Correctness	35	35
Report	20	20
There are several negative evaluations	…	…