**Communications and Information Engineering Program**

Probability and Stochastic Processes (CIE 327)

Omar Wael Abdelmohsen Ayyad (202000806)

Project link:

<https://github.com/Omar3yyad25/Letters-Probabilty-Project>

**1. Introduction**

First of all, this project is a feature of a data analysis procedure based on a probability research. The method for assessing data loaded in the form of files is critical since the results are used in various research, including scientific and literary objectives. This project is simply a basic code for analyzing a file that you can add through the place designated for it in the GUI, and it will display the necessary statistics and graphs.

Graphical user interface

Description automatically generated**2. Code Structure**

The Cod is divided into two section which are the GUI as the following figure, and the implementation code which also consisted of 4 main function related to the 4 button provided in the GUI, the functions as following:

* **analysis() :** it is the main function for recalling rest of calculations functions and set the values which would be shown on the GUI in the specified fields.
* **clear() :** it is the function requested to clear all the variables values to start new analysis and set all the text boxes in the GUI equals to zero

my\_text\_box.delete("1.0","end")  
mean.set(0)  
var.set(0)  
ske.set(0)  
kurt.set(0)  
most\_letter.set(0)  
most\_letterOc.set(0)

* **show\_pmf() :** it is the function required to plot our PMF graph, which takes the values of probabilities for each number and letter and start plotting it using embedded library called **matplot.**

names = list(original\_dic.keys())  
values = list(prob\_dic.values())  
plt.bar(range(len(prob\_dic)), values, tick\_label=names)

* **show\_cdf() :** it is the function required to plot our CDF graph, using **numpy** library and function called **cumsun()** which record the CDF values and plot as the previous PMF.

**3. Managing Data**

As the data in the file provided not filtered as wanted, there was a mission to filter the text by using embedded function in python called **isalnum()** which make your text only have letters and numbers.



Unfortunately, the file may have another strange letters like **á , é** which cannot be caught by the previous function. Otherwise, the following solution helped us to solve the problem by making constant dictionary and the **if condition** in the analysis to deal with the only items’ key provided in it. 

The concept of dictionary is as follows, **dictionary\_name = {‘key’ : value}.**

Then as having two dictionaries called **original\_dic** and **prob\_dic** this allowed us to have different ways to get your calculation

* **original\_dic** : it is to record the number of occurrence of each letter as the value of each letter and number by the following code which return another unfiltered dictionary and start filter it again by the if condition mentioned above ,

def word\_freq(string):  
 text = ''.join(e for e in string if e.isalnum())  
 d = Counter(''.join(text))  
 sorted\_keys = sorted(dict(d).keys())  
 sorted\_dict = {key:dict(d)[key] for key in sorted\_keys}  
 return (sorted\_dict)

This dictionary allowed us to calculate the most repeated letter and its number of occurrence and set it on the GUI interface by using **max()** function provided from **operator** library and its code as follows,

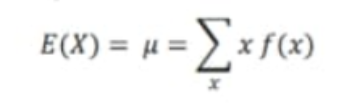
most\_letter.set(max(original\_dic.items(), key=operator.itemgetter(1))[0])  
most\_letterOc.set(max(original\_dic.items(), key=operator.itemgetter(1))[1])

**prob\_dic**: it is required to store the probability for each key as it is have the same keys in the another dictionary. Knowing that the probability of the letter or the number is : # of occurrence / sum of letters and number. After that, the result of the equation will be stored as the value of the dictionary,

sumLetters = sum(original\_dic.values())  
for n in original\_dic:  
 prob\_dic[n] = original\_dic[n]/sumLetters

**4. Probability**

As requested, the mean, variance, skewness, and kurtosis, there are 4 function with following names **calc\_mean() , calc\_var(), calc\_kurt(), calc\_ske()** to define each objective and return by the value.

General case: using a variable called counter to define the index the used item in the dictionary.

**calc\_mean()**: as we know that the equation of the mean is

the implementation of the code depended on the equation and used an array called **probability\_mean** and get the sum of it as the end.

counter = 0  
for i in prob\_dic:  
 x= counter \* prob\_dic[i]  
 counter = counter+1  
 probabilty\_mean.append(x)  
mean\_1 = sum(probabilty\_mean)  
return mean\_1

**calc\_var():** this function is used to calculate the variance and we know its formula as following

getting the excepted if X² and stored its value in array called **probability\_mean2** and get its sum and minus the mean squared from it as following

counter = 0  
for j in prob\_dic:  
 y= pow(counter,2)\*prob\_dic[j]  
 probabilty\_mean2.append(y)  
 counter = counter +1  
var\_1 = sum(probabilty\_mean2) - pow(sum(probabilty\_mean),2)  
return var\_1

Text

Description automatically generated**calc\_kurt():** as the previous, getting the excepted of X power 4 and store it in **probability\_mean4** and get the kurtosis, but with different handling as the dominator should not equals to zero,

counter = 0  
for j in prob\_dic:  
 y= pow(counter,4)\*prob\_dic[j]  
 probabilty\_mean4.append(y)  
 counter = counter +1  
  
dom = pow(math.sqrt(sum(probabilty\_mean2) - pow(sum(probabilty\_mean),2)),4)  
if (dom != 0):  
 kurt\_1 = (sum(probabilty\_mean4)-4\*(sum(probabilty\_mean))\*(sum(probabilty\_mean3))+6\*(pow(sum(probabilty\_mean),2))\*((sum(probabilty\_mean2) - pow(sum(probabilty\_mean),2)))+3\*(pow(sum(probabilty\_mean),4)))/pow(math.sqrt(sum(probabilty\_mean2) - pow(sum(probabilty\_mean),2)),4)  
 return kurt\_1  
else:  
 return ("None")

Text

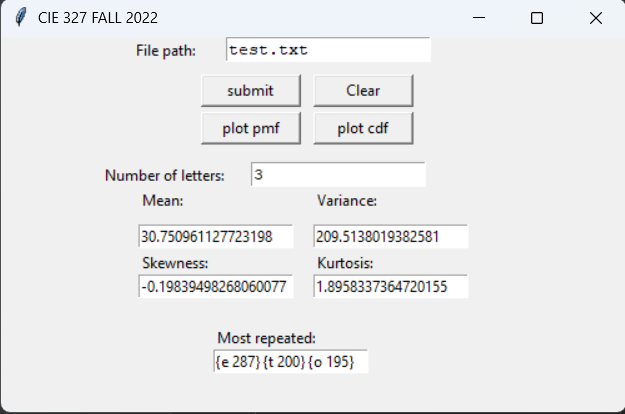
Description automatically generated with medium confidence**calc\_ske():** as the previous, getting the excepted of X power 4 and store it in **probability\_mean3** and get the skewness,

counter = 0  
for j in prob\_dic:  
 y= pow(counter,3)\*prob\_dic[j]  
 probabilty\_mean3.append(y)  
 counter = counter +1  
  
dom = pow(math.sqrt(sum(probabilty\_mean2) - pow(sum(probabilty\_mean),2)),3)

if (dom != 0):  
 ske\_1 = (sum(probabilty\_mean3) - 3\*sum(probabilty\_mean)\*(sum(probabilty\_mean2) - pow(sum(probabilty\_mean),2))-pow(sum(probabilty\_mean),3))/pow(math.sqrt(sum(probabilty\_mean2) - pow(sum(probabilty\_mean),2)),3)  
 return ske\_1  
else:  
 return ("None")

**5. Testing and Results**

* Test 1:

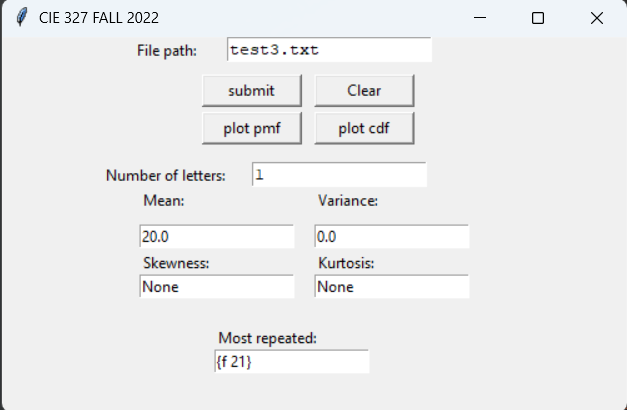
Importing the sample file provided on with documents the GUI gave us the following information,

Chart, histogram

Description automatically generatedChart, histogram

Description automatically generated

* Graphical user interface, text, application

  Description automatically generatedTest 2:
* Chart, bar chart

  Description automatically generatedChart

  Description automatically generated with medium confidenceImporting new file have only 1 letter by typed more than one time. The result as expected that the variance = 0 , skewness = none , and kurtosis = none