

ADDITIONAL ACADEMIC REPORT

PYTHON and C



March 15, 2018

PRIYA CHEVLI

17002336

Main Introduction

This report is going to talk about 3 main things. The first being a piece of python code to manipulate statistical data and provide a method of how the statistical test is going to be carried out. It can even provide some graphs depending on the statistical test. The second is a piece of C code that simulates water falling down and landing on different levels. The third is a short reflection on the topic of bioinformatics. This mainly explains the analysis of DNA and protein structures and how this can be used to create medication specialised for the individual.

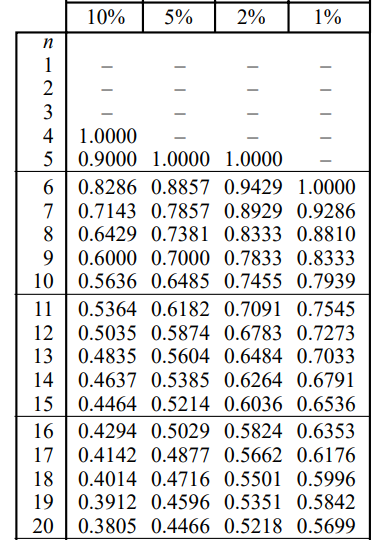
**PYTHON**

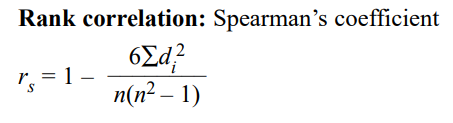
Introduction

Python is a language used to manipulate statistical data and then graph this data on graphs using a range of libraries that can be imported into python. This main use of python helped form the idea for this report along with a A-Level biology topic that had been studied. This topic required students to process, analyse and present data using different statistical tests. This was something very complicated to do by hand as it required a lot of steps. The code created in this report helps simplify this process.

There were a range of statistical tests in the biology course, each carrying out a different task depending on the type of experiment being carried out. This report focusses on four main tests. These tests are; Spearmans Rank, T-Test, Chi-Squared and Lincoln Index. The first three tests mentioned took the longest to do by hand and were the most complicated out of the bunch. These three can also be plotted on a graph. The last one is important for working out distributions hence why it was included in the project and this one cannot be plotted on a graph. To give a better insight as to why the above four tests were used the tests all four need to be explained.

1. Spearmans Rank

This tests for a significant relationship between two sets of measurements from the same sample.



***Figure1***

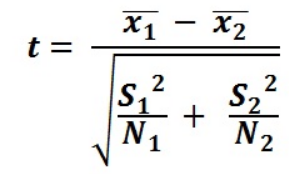
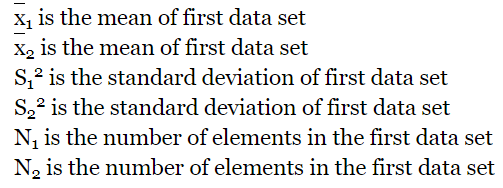
Figure1 shows the formula for this test. *d* is the difference between each ranked pair of measurements and *n* is the number of pairs of items in the sample

Figure2 is the table showing the critical values at a range of different probabilities. The code for this test will be testing the result at a probability of 0.05 (P = 0.05) which is at 5% on the table, for different numbers of paired values. The pair of measurements to use should be between 5 and 30 pairs. This is because 5 is the smallest sample size that should be used.

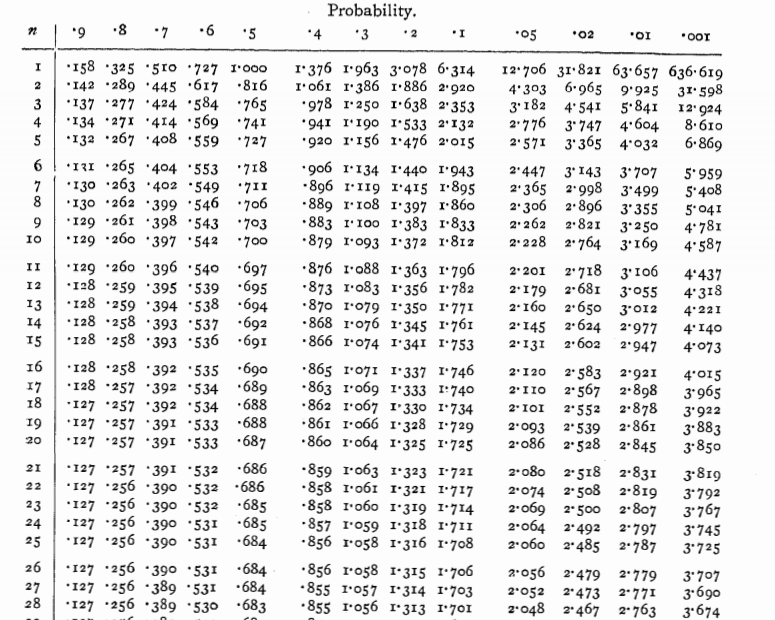
***Figure2***

1. T-test

This tests the null hypothesis for a significant difference between two sets of normally distributed data. This involves comparing the mean and standard deviation of both sets of data to determine if they are significantly different. The standard deviation is the spread of data. The formula is in figure3 with the corresponding variable name explanation in figure4. This can only work when the data is distributed and the sample size is less than 25.



***Figure3***

 ***Figure4***

This table here in figure5 shows the critical values of *t* for different degrees of freedom at a probability of 0.05 (P = 0.05). On the table the values in the .05 column are being used for the test.

The degrees of freedom value or n value looking at figure5 is worked out by doing

(n1 + n2) -2

***Figure5***

1. Chi-Squared

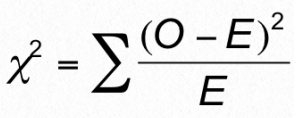
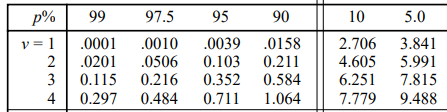
**Tests for the significant difference between observed and expected results

Figure6 shows the formula to use, the *O* represents the results observed and *E* is the expected results

***Figure6***



***Figure7***

Figure7 is a table showing the critical values of the x2 at probability of 0.05 (P = 0.05) looking at the 5.0 column for different degrees of freedom. The degree of freedom is the number of independent values or quantities which can be assigned to a statistical distribution. The degrees of freedom value is worked out by doing the number of categories-1

1. Lincoln index

This estimates the population of a species in a certain area.

Figure8 is the formula to use where S1 is the first sample, S2 is the second sample, R is the number of individuals the same from the first sample and N is the estimated population that is being calculated.

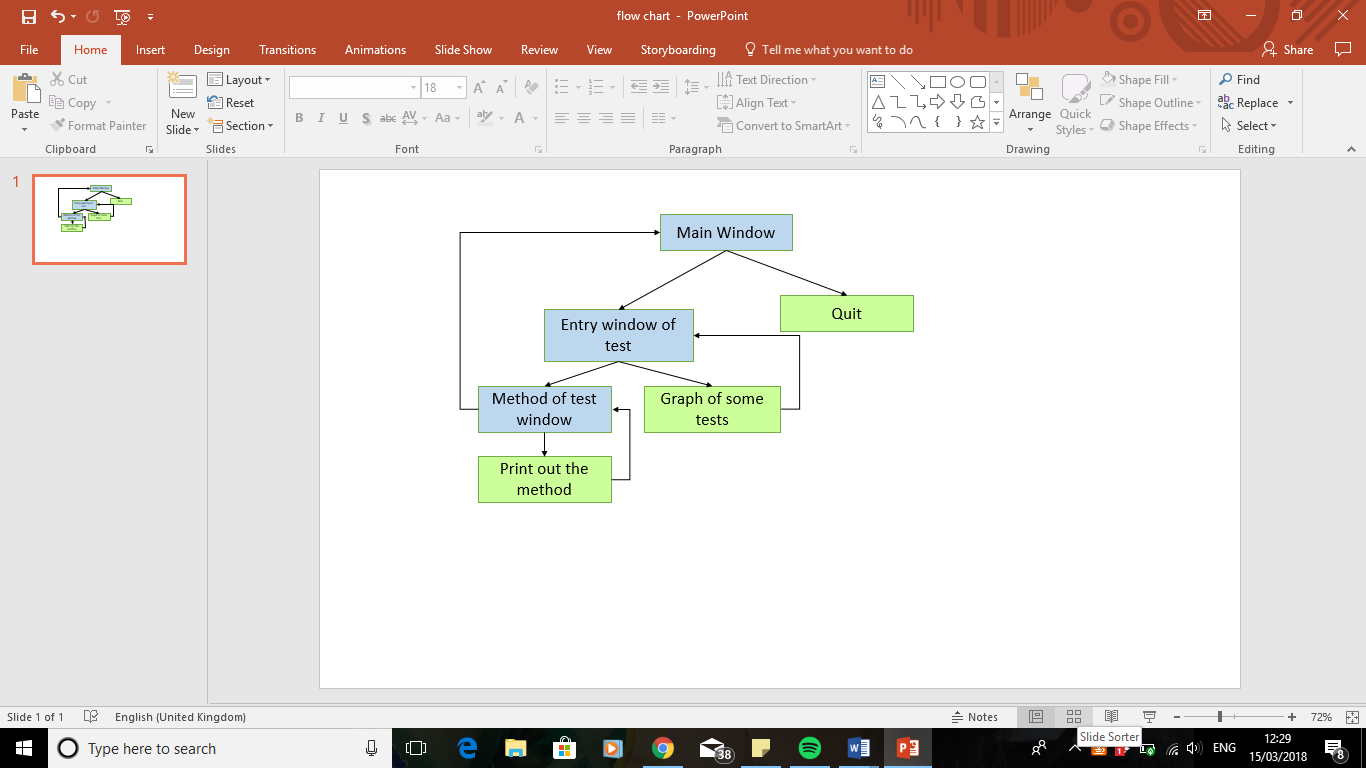
***Figure8***

Design

The project is to be displayed to the user using a user interface. This is going to be done using tkinter which is a python library that can be imported to python 3.6. The most up to date version of python is being used so it can work best with this library.

The user interface is going to have separate windows for each of the different statistical tests. Each test will have 2 windows, one for the entry of data and another for showing the method of how the statistic test was done. The graph of some of these tests is also shown on a new window but it isn’t created manually, it is done automatically.

The window colour was going to be white to keep things neutral but a light pink colour called “thistle1” is being used instead so the windows don’t look plain and boring. This has been kept consistent in all the windows. The window sizes all vary depending on how much data is to be entered and how much data is going to be displayed. The windows that will have a lot of information have scrollbars on them to prevent the data from overlapping or get missed off the window. Each of the windows have different uses which can be seen from figure9. Depending on what the window is going to do will depend on which widget is being displayed. The different types of widgets that can be used are: radiobuttons, buttons, labels and entryboxes. Graphs can also be created and the graphs create their own window to store everything on. These graphs just need to be labelled and provided the right information.



***Figure9, showing how the windows and functions link together. The windows are in blue and the functions are in green. The functions are the extra things the code can do***

Test data was also found and used in the design phase for each of the statistical tests. This is to make sure that the code was producing the right results. (CGP,2015)

Implementation

The way this project has been implemented is by splitting each key event into its own method so each method is only needed when it gets to that key stage. These methods are placed into one big class. The implementation of this was a very lengthy process but many of the ideas in each of the different tests could be repeated. For example, the way the method for each test is shown are all done in the same way, just with different variable names. This is the same with the printout methods. In this section each key feature will be explained.

* Main Window

When the code is run this is the first thing that can be seen. It is the first window to appear with the available statistical tests that the user can do. It is the main window that everything else navigates to at the end. Below is the code for this.

###This function is used to create the first main window that the user sees and can interact with

def mainWindow(self):

#'main' is a window created from the root that has been set up

#we can destroy the main window without the program crashing

#global makes this varaible known everywhere and not just in this function

global main

main = Toplevel(root)

main.focus\_force()

#This line below sets the background colour of the window

main.configure(background = "thistle1")

#This line below sets the size of the window

main.geometry("300x300+200+200")

#These two lines set a variable that will help with changing which window is displayed when a button is clicked

#This can be done using the goDo function

global v

v = IntVar()

#These are all widgets that can be displayed on the window

#This below is a Label that is acting like a heading

mainLabel = tkinter.Label(main, text = "Statistical Analysis", background = "thistle1")

#These Radiobuttons below give the user a choice as to which ststistical test they want to carry out

csRadioButton = Radiobutton(main, text = "Chi-Squared", variable = v, value = 1, bg = "thistle1")

srRadioButton = Radiobutton(main, text = "Spearmans Rank", variable = v, value = 2, bg = "thistle1")

ttRadioButton = Radiobutton(main, text = "T-test", variable = v, value = 3, bg = "thistle1")

liRadioButton = Radiobutton(main, text = "Lincoln Index", variable = v, value = 4, bg = "thistle1")

These variables are very useful when trying to manoeuvre through different windows. These variables will be used in the goDo method shown below.

#These two lines of code are buttons

#The user can quit the program or they can continue with the statistical test they want to do

goButton = tkinter.Button(main, text = "go", command = self.goDo)

quitButton = tkinter.Button(main, text = "quit", command = root.destroy)

#This next block of code places the widgets on the window by using pack()

#These are all packed in the order to be placed on the window

mainLabel.pack()

csRadioButton.pack()

srRadioButton.pack()

ttRadioButton.pack()

liRadioButton.pack()

goButton.pack()

quitButton.pack()

The method below is used by the main window so navigate to the new windows when needed

###This function is used to navigate through the 4 statistic test windows

def goDo(self):

if v.get() == 1:

The v.get() function gets the value from the corresponding radio button in the mainWindow method and then checks to see which IF statement is true. If an IF statement is true, this method opens the new window for the required statistical test. An IF statement was the most suitable selection statement to use as it checks for all known possibilities.

#Go to chiSquaredWindow

self.chiSquaredWindow()

if v.get() == 2:

#Go to spearmansRankWindow

self.spearmansRankWindow()

if v.get() == 3:

#Go to tTestWindow

self.tTestWindow()

if v.get() == 4:

#Go to lincolnIndexWindow

self.lincolnIndexWindow()

* Spearmans Rank

This is one of the tests that can be done. It has two windows which are the entry window and the method window. This tests also shows a graph of the results and can printout the method. These 4 features will be explained below.

1. Entry Window

This is the window that the user is taken to after they have chosen to do a spearmans rank test. Here the user can enter all the data required to do the test. Everything is labelled so the user will know where to enter the information.

###This is the spearmans rank main window

def spearmansRankWindow(self):

#The main window is destroyed

main.destroy()

global srwindow

global srindependentvarEntry

global srdependentvarEntry

global xtotalEntry

global xvalEntry

global backButton2

global srcanvas

global srframe

#Here are all the features of the window

srwindow = Toplevel(root)

srwindow.focus\_force()

srwindow.configure(background = "thistle1")

srwindow.geometry("475x500+100+100")

#This frame is created so the scrollbar can be added to it further down

frame = Frame(srwindow,relief=GROOVE,width=100,height=100)

frame.place(x=10, y=10)

srcanvas = Canvas(frame)

#the background here changes the colour of the frame

srframe = Frame(srcanvas, bg = "thistle1")

scrollbar = tkinter.Scrollbar(frame, orient="vertical", command = srcanvas.yview)

srcanvas.configure(yscrollcommand=scrollbar.set)

#Here the scrollbar is added to the frame

scrollbar.pack(side="right", fill="y")

srcanvas.pack(side="left")

srcanvas.create\_window((0,0),window=srframe,anchor='nw')

srframe.bind("<Configure>",self.srScrollfunction)

#The next lines of code add the widgets to the window

srLabel = tkinter.Label(srframe, text = "SPEARMANS RANK", background = "thistle1")

srindependentvarLabel = tkinter.Label(srframe, text = "What was the independent variable in the investigation?", bg = "thistle1")

srindependentvarEntry = tkinter.Entry(srframe,width = 30)

srdependentvarLabel = tkinter.Label(srframe, text = "What was the dependent variable in the investigation?", bg = "thistle1")

srdependentvarEntry = tkinter.Entry(srframe,width = 30)

xtotalLabel = tkinter.Label(srframe, text = "How many values for the dependent and independent variables?", background = "thistle1")

xtotalEntry = tkinter.Entry(srframe)

xtotalGoButton = tkinter.Button(srframe, text = "Enter", command = self.xvarentries)

backButton2 = tkinter.Button(srframe, text = "back to main", command = self.toMainFromsrwindow)

When the enter button is clicked the xvarentries method is called.

#The widgets are then packed

srLabel.pack()

srindependentvarLabel.pack()

srindependentvarEntry.pack()

When placing items on the window, the place it is put can be specified. The back button here has been placed at the bottom. This was the best place to put this bottom as it is easy to see and isn’t in the way of all the other information being displayed on the window.

srdependentvarLabel.pack()

srdependentvarEntry.pack()

xtotalLabel.pack()

xtotalEntry.pack()

xtotalGoButton.pack()

backButton2.pack(side = BOTTOM)

The window for the entry of data uses a scrollbar. The scrollbar is needed as the user could be entering a large data set. All of this won’t fit on the window so having the scrollbar ensures that all the data can be entered and all the information can be seen to the user. This is especially important when showing the result to the user at the bottom of the window. If the result cannot be seen, it would be a problem. The scrollbar is used with chi-squared and the t-test too. The scrollbar cannot be added to the method above so it needs its own method which can be seen below. The scrollbar methods for the other two tests aren’t going to be included as they are the same in principle but with different variable names.

###This gives the Spearmans Rank window a scrollbar

def srScrollfunction(event):

#the colour here makes the whole bg coloured

srcanvas.configure(scrollregion= srcanvas.bbox("all"),width=430,height=470, bg = "thistle1")

When the enter button is clicked the below method is called. This method checks the data entered is valid and then carries on getting more of the data from the user. In this method the x variables for the Spearmans Rank test are entered. These are the independent variable values.

###This function gets the spearmans rank x variable values from the user

def xvarentries(self):

global xvalEntry

global xval

global xcount

global xtotal

#exception handling total number of variables in both variables window

#Tries to get an integer from the user, if not an error is shown

Here the number of independent variables entered is checked to see if it is valid or not.

try:

xtot = xtotalEntry.get()

xtotal = int(xtot)

except:

messagebox.showinfo("ERROR", "You haven't entered an integer (a whole number)")

xvalLabel = tkinter.Label(srframe, text = "Give the independent variable values below", background = "thistle1")

xvalLabel.pack()

A FOR loop is used here as we don’t know how many values are to be entered values there are

xcount = 0

xval = []

for xcount in range (0,xtotal):

#This is where the user enters the independent variable values

xvalEntry = tkinter.Entry(srframe)

xval.append(xvalEntry)

xvalEntry.pack()

#Here the yvarentries() function is called

self.yvarentries()

In this method the y variables for the Spearmans Rank test are entered. The y variables are the dependent variables

###This function gets the spearmans rank y variables from the user

def yvarentries(self):

global yvalEntry

global yval

yvalLabel = tkinter.Label(srframe, text = "Give the dependent variable values below", background = "thistle1")

yvalLabel.pack()

ycount = 0

yval = []

for ycount in range (0,xtotal):

#This is where the user enters the dependent variable values

yvalEntry = tkinter.Entry(srframe)

yval.append(yvalEntry)

yvalEntry.pack()

calculateButton = tkinter.Button(srframe, text = "Calculate", command = self.spearmansrank)

calculateButton.pack()

When the calculate button is pressed the spearmansrank method is called. Here the result for the test is calculated.

1. Method and Method Window

This is how the test is calculated to find the result. This result along with the hypothesis and the critical value is shown to the user on the entry window. The method can then be shown to the user in a new window.

The way the result was found is shown below. This is called when the calculate button is pressed.

###This is the spearmans rank working out window

def spearmansrank(self,\*args):

global copyofx

global copyofy

global xfinalRank

global yfinalRank

global D

global D2

global SD2

global n

global r

Checks to see if the names of the independent and dependent variables are of the correct data type otherwise the hypothesis once created wouldn’t make sense.

global srans

global srinull

global srdnull

global nop

#null hypothesis

#exception handling independent and dependent variables

try:

#Tries to get strings from the user, if not an error would be shown

sri = srindependentvarEntry.get()

srinull = str(sri)

srd = srdependentvarEntry.get()

srdnull = str(srd)

except:

messagebox.showinfo("ERROR", "You haven't entered the correct data type (need to enter text)")

#get x entries into a list

xvalues = []

for xvalEntry in xval:

#exception handling x entries

#tries to get a float from the user, if not an error would be shown

try:

p = xvalEntry.get()

a = float(p)

except:

messagebox.showinfo("ERROR", "You haven't entered a number (can be a decimal)")

xvalues.append(a)

The values entered for the independent variable are added to the xvalues list. Using a list will help rank the numbers in ascending order.

#This for loop convert items in list to float

for i in range (0, len(xvalues)):

xvalues[i] = float(xvalues[i])

#This block of code gets the y entries into a list

yvalues = []

for yvalEntry in yval:

#exception handling y entries

#It tries to get a float, if not an error would be shown

try:

q = yvalEntry.get()

b = float(q)

except:

messagebox.showinfo("ERROR", "You haven't entered a number(can be a decimal)")

The values entered for the dependent variable are added to the yvalues list.

yvalues.append(b)

#This converts the items in a list to float

for n in range (0, len(yvalues)):

yvalues[n] = float(yvalues[n])

This variable stores the result of the test. The letter r is used to store this value as it is the same as in the formula

#r is the spearmans rank variable

r = 0

#the ranking of x variable list

A copy of the original variable list is made so the original list can be used to show the method to the user if they wish to see it

#need to make a copy of list x to do this

copyofx = xvalues.copy()

#need a list to store the values visited

xvisitedValues = []

#need a list to store the positions

xpositions = []

The ranking of the values is done for the independent variables now

# need to sort the list

xsorted = xvalues.sort()

#enumerate through the list

for e, j in enumerate(xvalues):

#if the value from the list isn't in the list xvisitedValues

if j not in xvisitedValues:

#assign the value in the list to the variable name value

xvalue = j

#num is the number of times the no is repeated in the list

xnum = xvalues.count(j)

#this resets the posadd so new values can be added to it

xposadd = 0

#now iterate again through the list

xiterate = 0

for xiterate in range(0, len(xvalues)):

#if the value is the same as the value in the position checking

if j == xvalues[xiterate]:

xgetpos = xiterate

#add the position of the (value + 1) to list visitedValues

#cannot start at 0 so must add 1

xposadd = xposadd + (xgetpos + 1)

xvisitedValues.append(xvalues[xgetpos])

for add in range(0, xnum):

xpositions.append(xposadd/xnum)

#need an empty list to store the positions in the correct order

xfinalRank = []

#make the list have 0's in each position of the length of copyofx list

for item in copyofx:

xfinalRank.append(0)

#make another copy of list x

xcopy2 = xvalues.copy()

#want to go through each item in list copyofx

for e, j in enumerate(copyofx):

#find the item j in t

# .index gets the first occurence of the j item we are looking for

xpos = xcopy2.index(j)

#find the position of the occurence in the positions list and store in finalRank

xfinalRank[e] = xpositions[xpos]

#need to add a zero to the item in that position in list xcopy2 so it doesnt get found again

xcopy2[xpos] = 0

#the ranking of y variable list

#need to make a copy of list y to do this

copyofy = yvalues.copy()

#need a list to store the values visited

yvisitedValues = []

#need a list to store the positions

ypositions = []

The ranking of the values is done for the dependent variables now

# need to sort the list

ysorted = yvalues.sort()

#enumerate through the list

for e, j in enumerate(yvalues):

#if the value from the list isn't in the list visitedValues

if j not in yvisitedValues:

#assign the value in the list to the variable name value

yvalue = j

#num is the number of times the no is repeated in the list

ynum = yvalues.count(j)

#this resets the posadd so new values can be added to it

yposadd = 0

#now iterate again through the list

yiterate = 0

for yiterate in range(0, len(yvalues)):

#if the value is the same as the value in the position checking

if j == yvalues[yiterate]:

ygetpos = yiterate

#add the position of the (value + 1) to list visitedValues

#cannot start at 0 so must add 1

yposadd = yposadd + (ygetpos + 1)

yvisitedValues.append(yvalues[ygetpos])

for add in range(0, ynum):

ypositions.append(yposadd/ynum)

#need an empty list to store the positions in the correct order

yfinalRank = []

#make the list have 0's in each position of the length of c list

for item in copyofy:

yfinalRank.append(0)

#make another copy of list a

ycopy2 = yvalues.copy()

#want to go through each item in list copyofy

for e, j in enumerate(copyofy):

#find the item j in t

# .index gets the first occurence of the j item we are looking for

ypos = ycopy2.index(j)

#find the position of the occurence in the positions list and store in finalRank

yfinalRank[e] = ypositions[ypos]

#need to add a zero to the item in that position in list t so it doesnt get found again

ycopy2[ypos] = 0

#subracting values from two lists

D = []

#t value will help iterate through each item in the lists

t = 0

for t in range (0, len(xfinalRank)):

d = xfinalRank[t] - yfinalRank[t]

t = t + 1

D.append(d)

#now need to square all the values in the D list

#add the squared values to the list D2

D2 = []

e = 0

for e in range (0, len(D)):

d2 = D[e]\*D[e]

e = e + 1

D2.append(d2)

#need to find the sum of the values in the D2 list and multiply it by 6

SD2 = (sum(D2))\*6

#need to find (n^3)-n

n = ((xtotal)\*(xtotal)\*(xtotal))-(xtotal)

#finally use all these values to get the spearmans rank value

r = 1 - (SD2/n)

The final value has been rounded to three decimal places

r = str(round(r,3))

#number of pairs for the critical value

nop = xtotal

#check if the hypothesis is significant or not at a probability of 0.5 (P=0.05)

The critical values from the table have been added to this list so it’s easier to get the right value

#starts at a minimum of 5 pairs

srcriticalValList = [1,0.886,0.786,0.738,0.700,0.648,0.618,0.587,0.560,0.538,0.521,0.505,0.485,0.472,0.460,0.447]

This IF then works out which statement is displayed

for n in range(0,len(srcriticalValList)):

srcriticalVal = srcriticalValList[nop-5]

#This then works out if the null hypothesis is significant or not?

if float(r) >= float(srcriticalVal):

srans = "The spearmans rank value of " + str(r) + " is greater than the critical value of " + str(srcriticalVal)+ "\n We reject null hypothesis at a 5% significance level, \n as there is only a 5% probability that the results are due to chance"

else:

srans = "The spearmans rank value of " + str(r) + " is less than the critical value of " + str(srcriticalVal)+ "\n We accept the null hypothesis at a 5% significance level, \n as there is a 95% probability that the results are due to chance"

#Here all the final information is displayed on the window

spearmansRankLabel = tkinter.Label(srframe, text = "The spearmans rank value is: " + r , background = "thistle1")

srnullHypothesisLabel = tkinter.Label(srframe, text = "The null hypothesis is:", background = "thistle1")

srnullHypothesisDataLabel = tkinter.Label(srframe, text = "There is no significant correlation between " + str(srinull) + " and " + str(srdnull), bg = "thistle1")

srcriticalValCompLabel = tkinter.Label(srframe, text = srans, bg = "thistle1")

#Two buttons are being displayed, one shows the method, the other shows a graph

methodButton = tkinter.Button(srframe, text ="Click here to see the full method", command = self.srMethodWindow)

graphButton = tkinter.Button(srframe, text="Click to see graph", command = self.srGraph)

spearmansRankLabel.pack()

srnullHypothesisLabel.pack()

srnullHypothesisDataLabel.pack()

srcriticalValCompLabel.pack()

methodButton.pack()

graphButton.pack()

The method window is then shown when the method button is clicked on the spearmans rank entry window. The method window is created in the srMethodWindow function which is shown below. The way this works is by using a lot of labels, each containing either an explanation of what the data underneath it shows or how the value was found, or it shows the data. This is all done using labels which are packed underneath each other. The data however could only be added to the label if it was a string so the data had to be converted into a string in the labels.

###This function contains the method for working out the spearmans rank

#each step is shown

def srMethodWindow(self):

#Here the srwindow is destroyed as this new one is created

srwindow.destroy()

global srmethwindow

#The features of the new window created are mentioned here

srmethwindow = Toplevel(root)

srmethwindow.focus\_force()

srmethwindow.configure(background = "thistle1")

srmethwindow.geometry("700x500+100+100")

methodLabel = tkinter.Label(srmethwindow, text = "The worked method", bg = "thistle1")

#var 1 original list

var1ListLabel = tkinter.Label(srmethwindow, text = "The first variable value list: " ,bg = "thistle1")

var1ListDataLabel = tkinter.Label(srmethwindow, text = str(copyofx), bg = "thistle1")

#var 1 ranked list

var1RankLabel = tkinter.Label(srmethwindow, text = "The first variable value list ranked in ascending order: " ,bg = "thistle1")

var1RankDataLabel = tkinter.Label(srmethwindow, text = str(xfinalRank),bg = "thistle1")

#var 2 original list

var2ListLabel = tkinter.Label(srmethwindow, text = "The second variable value list: " ,bg = "thistle1")

var2ListDataLabel = tkinter.Label(srmethwindow, text = str(copyofy), bg = "thistle1"

Here this list called copyofy is converted into a string using str()

#var 2 ranked list

var2RankLabel = tkinter.Label(srmethwindow, text = "The second variable value list are ranked in ascending order: " ,bg = "thistle1")

var2RankDataLabel = tkinter.Label(srmethwindow, text = str(yfinalRank), bg = "thistle1")

#differences in rank

rankDiffLabel = tkinter.Label(srmethwindow, text = "Find the difference between each item in the ranked lists: " , bg = "thistle1")

rankDiffDataLabel = tkinter.Label(srmethwindow, text = str(D), bg = "thistle1")

#square each ranked value

squaredRankLabel = tkinter.Label(srmethwindow, text = "The above list items are now squared to give this: " , bg = "thistle1")

squaredRankDataLabel = tkinter.Label(srmethwindow, text = str(D2), bg = "thistle1")

#sum of the values

sumRankValsLabel = tkinter.Label(srmethwindow, text = "The sum of the values given above are: " ,bg = "thistle1")

sumRankValsDataLabel = tkinter.Label(srmethwindow, text = str(SD2/6), bg = "thistle1")

#sum \*6

sum6Label = tkinter.Label(srmethwindow, text = "The above value is now multiplied by 6: " ,bg = "thistle1")

sum6DataLabel = tkinter.Label(srmethwindow, text = str(SD2), bg = "thistle1")

#(n^3)-1

nvaluesLabel = tkinter.Label(srmethwindow, text = "(n^3)-n is now worked out (n is no of items): " ,bg = "thistle1")

nvaluesDataLabel = tkinter.Label(srmethwindow, text = str(n), bg = "thistle1")

#final equation

srFinalAnsLabel = tkinter.Label(srmethwindow, text = " Do 1-(" + str(SD2) + "/" + str(n) +") to get the final answer of: " , bg = "thistle1")

srFinalAnsDataLabel = tkinter.Label(srmethwindow, text = str(r), bg = "thistle1")

#This button is used to print out a hard copy of the method

printButton = tkinter.Button(srmethwindow, text = "Print the method", command = self.srprint)

backButton = tkinter.Button(srmethwindow, text = "back to main", command = self.toMainFromsrmethwindow)

methodLabel.pack()

var1ListLabel.pack()

var1ListDataLabel.pack()

var1RankLabel.pack()

var1RankDataLabel.pack()

var2ListLabel.pack()

var2ListDataLabel.pack()

var2RankLabel.pack()

var2RankDataLabel.pack()

rankDiffLabel.pack()

rankDiffDataLabel.pack()

squaredRankLabel.pack()

squaredRankDataLabel.pack()

sumRankValsLabel.pack()

sumRankValsDataLabel.pack()

sum6Label.pack()

sum6DataLabel.pack()

nvaluesLabel.pack()

nvaluesDataLabel.pack()

srFinalAnsLabel.pack()

srFinalAnsDataLabel.pack()

printButton.pack()

backButton.pack(side = BOTTOM)

The rest of the methods are done in the similar way when the equivalent button is pressed. The other method windows aren’t going to be shown in this section because they will be the same as this one but with the corresponding variable names and the right steps to show the user how the result was found. In principle the way the method is shown is the same as this above.

1. The Graph

This is a window that is created automatically when the graph button is clicked on the spearmans rank entry window. This graph shows a visual representation of the test being done and why the result found is significant or not. The code for this is below.

###This function is to draw the spearmans rank graph

def srGraph(self):

#95% conidence level or P=0.05

srcriticalValList = [1,0.886,0.786,0.738,0.700,0.648,0.618,0.587,0.560,0.538,0.521,0.505,0.485,0.472,0.460,0.447]

xPoints = [5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

#plot the r value

plt.scatter(nop,np.absolute(r))

plt.axis([0,nop+1,0.0,1.0])

#plot the 95% confidence level

plt.plot(xPoints,srcriticalValList)

#give the graph some axies titles

plt.xlabel("Degree of Freedom")

plt.ylabel("Spearmans Rank Correlation Coefficient")

plt.title("The Spermans Rank value and degrees of freedom at P=0.05")

plt.show()

1. Printout the method

This is a function that has been added. It can print out the method so a hard copy can be taken of the data entered in. The code for this is below.

###This will help further how the user can see the worked method, by enabling it to be printed out, a hard copy can be given to students or the teaacher can use the hard copy

def srprint(self):

#try and open up a spearmans rank file to store the data

Here a file to write to is trying to be opened. This is so the data can be stored here and then printed out using a printer.

try:

spearmansRankFile = open("spearmansRank.txt","r")

#if the file cannot be found make one

except:

spearmansRankFile = open("spearmansRank.txt","a")

#add the required data to the file

spearmansRankFile = open("spearmansRank.txt","w")

spearmansRankFile.write("The Worked Method" + "\n\n")

spearmansRankFile.write("The first variable value list:" + "\n")

spearmansRankFile.write(str(copyofx) + "\n\n")

spearmansRankFile.write("The first variable value list ranked in ascending order:" + "\n")

spearmansRankFile.write(str(xfinalRank) + "\n\n")

spearmansRankFile.write("The second variable value list:" + "\n")

spearmansRankFile.write(str(copyofy) + "\n\n")

spearmansRankFile.write("The second variable value list ranked in ascending order:" + "\n")

spearmansRankFile.write(str(yfinalRank) + "\n\n")

spearmansRankFile.write("Find the difference between each item in the ranked lists:" + "\n")

spearmansRankFile.write(str(D) + "\n\n")

spearmansRankFile.write("The above items are now squared to give this:" + "\n")

spearmansRankFile.write(str(D2) + "\n\n")

spearmansRankFile.write("The sum of the values given above are:" + "\n")

spearmansRankFile.write(str(SD2/6) + "\n\n")

spearmansRankFile.write("The above value is now multiplied by 6:" + "\n")

spearmansRankFile.write(str(SD2) + "\n\n")

spearmansRankFile.write("(n^3)-n is now worked out (n is the number of items):" + "\n")

spearmansRankFile.write(str(n) + "\n\n")

spearmansRankFile.write("Do 1-" + str(SD2) + "/" + str(n) + " to get the final answer of:" + "\n")

spearmansRankFile.write(str(r) + "\n\n")

spearmansRankFile.write("The number of pairs of data:" + str(nop) + "\n")

spearmansRankFile.write("The null hypothesis is:" + "\n")

spearmansRankFile.write("There is no significant correlation between " + str(srinull) + " and " + str(srdnull) + "\n\n")

spearmansRankFile.write(str(srans) + "\n")

This line prints the file created

os.startfile("spearmansRank.txt","print")

spearmansRankFile.close()

As can be seen from the code above, a lot of this code is like the method window but instead of labels, the lines are written to the file one line at a time and then printed out at the end.

* T-test

This is another test that can be done. It has 2 main windows, one to enter all the data in and the other to show the method. It also can print out a hard copy of the method and draw a graph. Below the entry window, the method and the graph code will be explained as showing the method to the user and printing out a hardcopy of the method has already been explained so it doesn’t need to be repeated.

1. Entry Window

This is the window that the user gets taken to when they want to do this statistical test.

###The t-Test main window

def tTestWindow(self):

main.destroy()

global ttwindow

global ttindependentvarEntry

global ttdependentvarEntry

global totalsEntry

global ttframe

global ttcanvas

ttwindow = Toplevel(root)

ttwindow.focus\_force()

ttwindow.geometry("500x500+100+100")

ttwindow.configure(background = "thistle1")

frame = Frame(ttwindow,relief=GROOVE,width=100,height=100,bd=1, bg="thistle1")

frame.place(x=10,y=10)

ttcanvas=Canvas(frame)

ttframe=Frame(ttcanvas,bg = "thistle1")

scrollbar = Scrollbar(frame, orient="vertical", command=ttcanvas.yview)

ttcanvas.configure(yscrollcommand=scrollbar.set)

#add the scrollbar

A scrollbar is added to this entry window as well as a lot of data could be entered and it may not fit on the window.

scrollbar.pack(side="right", fill="y")

ttcanvas.pack(side="left")

ttcanvas.create\_window((0,0),window=ttframe,anchor="nw")

ttframe.bind("<Configure>",self.ttScrollfunction)

ttLabel = tkinter.Label(ttframe, text = "T-TEST", background = "thistle1")

ttindependentvarLabel = tkinter.Label(ttframe, text = "What are the independent variables in the investigation?", bg = "thistle1")

ttindependentvarEntry = tkinter.Entry(ttframe,width = 30)

ttdependentvarLabel = tkinter.Label(ttframe, text = "What is the dependent variable in the investigation?", bg = "thistle1")

ttdependentvarEntry = tkinter.Entry(ttframe,width = 30)

totalsLabel = tkinter.Label(ttframe, text = "How many values for variable 1 and 2?", background = "thistle1")

totalsEntry = tkinter.Entry(ttframe)

totalsGoButton = tkinter.Button(ttframe, text = "Enter", command = self.xvalsentries)

backButton4 = tkinter.Button(ttframe, text = "back to main", command = self.toMainFromttwindow)

ttLabel.pack()

ttindependentvarLabel.pack()

ttindependentvarEntry.pack()

ttdependentvarLabel.pack()

ttdependentvarEntry.pack()

totalsLabel.pack()

totalsEntry.pack()

totalsGoButton.pack()

backButton4.pack(side = BOTTOM)

When the enter button is clicked the xvalsentries function is called to get the independent variable data. This is shown below.

###The t-Test x variable window

def xvalsentries(self):

global xvalueEntry

This is checking for an integer value, if an integer hasn’t been entered an error pops up and the user can try and enter the valid data type again.

global xvaluelist

global totals

#exception handling no of variables

try:

tots = totalsEntry.get()

totals = int(tots)

except:

messagebox.showinfo("ERROR", "You haven't entered an integer (a whole number)")

xvalueLabel = tkinter.Label(ttframe, text = "Give the first variable values below", background = "thistle1")

xvalueLabel.pack()

xcounted = 0

A list is used again here because it is easier to manipulate the data to do different things.

xvaluelist = []

for xcounted in range (0,totals):

xvalueEntry = tkinter.Entry(ttframe)

xvaluelist.append(xvalueEntry)

xvalueEntry.pack()

xvalueButton = tkinter.Button(ttframe,text = "Enter", command=self.yvalsentries)

xvalueButton.pack()

When the enter button is clicked the yvalsentries function is called so the user can enter the dependent variable values. This can be seen below.

###The t-Test y variable window

def yvalsentries(self):

global yvalueEntry

global yvaluelist

yvalueLabel = tkinter.Label(ttframe, text = "Give the second variable values below", background = "thistle1")

yvalueLabel.pack()

ycounted = 0

yvaluelist = []

for ycounted in range (0,totals):

yvalueEntry = tkinter.Entry(ttframe)

yvaluelist.append(yvalueEntry)

yvalueEntry.pack()

calculationButton = tkinter.Button(ttframe, text = "Calculate", command = self.ttest)

When the calculate button is clicked the ttest method is called and the result can be worked out

calculationButton.pack()

1. Method

This works out the T-test result when the calculate button is clicked. The code for this is below.

###The t-Test working out window

def ttest(self):

global totals

global x

global y

global sumx

global x2

global sumx2

global sumy

global y2

global sumy2

global xmean

global ymean

global xvar

global yvar

global din

global v1

global v2

global t

global ttinull

global ttdnull

global dof

global ttans

#null hypothesis

#exception handling independent and dependent variables

try:

This makes sure the independent and dependent variables are strings

tti = ttindependentvarEntry.get()

ttinull = str(tti)

ttd = ttdependentvarEntry.get()

ttdnull = str(ttd)

except:

messagebox.showinfo("ERROR", "You haven't entered the correct data type (need to enter text)")

#get x entries into a list

x = []

for xvalueEntry in xvaluelist:

#exception handling x vals

try:

r = xvalueEntry.get()

a = float(r)

except:

messagebox.showinfo("ERROR", "You haven't entered a number (can be a decimal)")

x.append(a)

#convert items in list to float

for p in range (0, len(x)):

x[p] = float(x[p])

#get the y entries into a list

y = []

for yvalueEntry in yvaluelist:

#exception handling y vals

try:

s = yvalueEntry.get()

b = float(s)

except:

messagebox.showinfo("ERROR", "You haven't entered a number (can be a decimal)")

y.append(b)

#convert items in a list to float

for q in range (0, len(y)):

y[q] = float(y[q])

#get the sum of the values in the x list

sumx = sum(x)

#find the x-squared vales with the sum

i = 0

x2 = []

Each variable in the x list is squared and added to the new list x2. A new list is used so the original list can be used in the method window to show the step by step process.

for i in range (0, len(x)):

xs = x[i] \* x[i]

i = i + 1

xs = round(xs,3)

x2.append(xs)

sumx2 = sum(x2)

#get the sum of the values in the y list

sumy = sum(y)

#find the y-squared vales with the sum

e = 0

y2 = []

for e in range (0, len(x)):

ys = y[e] \* y[e]

e = e + 1

y2.append(ys)

sumy2 = sum(y2)

#the x mean

xmean =(sumx)/totals

The mean value is needed to be found as it is used to find the final answer.

xmean = round(xmean,4)

#the y mean

ymean = (sumy)/totals

ymean = round(ymean,4)

#the xvariance

#sumx\*\*2 is sumx squared

xa = (sumx\*\*2)/totals

xvar = (sumx2 - xa) /(totals - 1)

xvar = round(xvar,4)

#the yvariance

ya = (sumy\*\*2)/totals

yvar = (sumy2 - ya) /(totals - 1)

yvar = round(yvar,4)

#the difference in means

#abs gives the +ve value

din = abs(xmean - ymean)

din = round(din,4)

#the addition of variance over total no of data items

v1 = xvar/totals

v1 = round(v1,4)

v2 = yvar/totals

v2 = round(v2,4)

#the t value

#sqrt has to be imported from the math library

t = din/ (math.sqrt(v1+v2))

t = str(round(t,3))

#once the t-value is found

#the degree of freedom needs to be calculated and then compared to table values

dof = (totals + totals)-2

dof = str(dof)

#check if the hypothesis is significant or not P=0.05

#starts at a minimum of 1

ttcriticalValList = [12.706,4.303,3.182,2.776,2.571,2.447,2.365,2.306,2.262,2.228,2.201,2.179,2.160,2.145,2.131,2.120,2.110,2.101,2.093,2.086,2.08,2.074,2.069,2.064,2.060,2.056,2.052,2.048]

for n in range(0,len(ttcriticalValList)):

g = int(dof)- 1

ttcriticalVal = ttcriticalValList[g]

#is it significant?

if float(t) >= float(ttcriticalVal):

ttans = "The T-Test value of " + str(t) + " is greater than the critical value of " + str(ttcriticalVal)+ "\n We reject null hypothesis at a 5% significance level, \n as there is a 5% probability that the results due to chance"

else:

ttans = "The T-Test value of " + str(t) + " is less than the critical value of " + str(ttcriticalVal)+ "\n We accept the null hypothesis at a 5% significance level, \n as there is a 95% probability that the results are due to chance"

ttestLabel = tkinter.Label(ttframe, text = "The T-test value is: " + t, background = "thistle1")

dofLabel = tkinter.Label(ttframe, text = "The degree of freedom is: " + dof, background = "thistle1")

srnullHypothesisLabel = tkinter.Label(ttframe, text = "The null hypothesis is:", background = "thistle1")

srnullHypothesisDataLabel = tkinter.Label(ttframe, text = "There is no significant difference between " + str(ttdnull) + " in " + str(ttinull), bg = "thistle1")

ttcriticalValCompLabel = tkinter.Label(ttframe, text = ttans, bg = "thistle1")

methodButton = tkinter.Button(ttframe, text = "Click here to see the full method", command = self.ttMethodWindow)

graphButton = tkinter.Button(ttframe, text = "Click to see graph", command = self.ttGraph)

ttestLabel.pack()

dofLabel.pack()

srnullHypothesisLabel.pack()

srnullHypothesisDataLabel.pack()

ttcriticalValCompLabel.pack()

methodButton.pack()

graphButton.pack()

1. The graph

If the graph button is clicked in the tTestWindow a graph is shown. This is to provide a visual representation of the final value. The code for this is below.

##The Ttest graph

def ttGraph(self):

#95% conidence level or P=0.05

#starts at a minimum of 1 to 28

ttcriticalValList = [12.706,4.303,3.182,2.776,2.571,2.447,2.365,2.306,2.262,2.228,2.201,2.179,2.160,2.145,2.131,2.120,2.110,2.101,2.093,2.086,2.08,2.074,2.069,2.064,2.060,2.056,2.052,2.048]

xPoints = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28]

#plot the t value

plt.scatter(float(dof),float(t))

plt.axis([0,float(dof)+1,0.0,12.706])

The titles are needed to give the graph more readability.

#plot the 95% confidence level

plt.plot(xPoints,ttcriticalValList)

#give the graph some axies titles

plt.xlabel("Degree of Freedom")

plt.ylabel("T-test")

plt.title("The T-test value and degrees of freedom at P=0.05")

plt.show()

* Chi-Squared

This is another test that can be done. It has 2 main windows, one to enter the data into and another to show the method. It can also print out a hard copy of the method and draw a graph. Below the entry window, the method and the graph code will be explained as showing the method to the user and printing out a hardcopy of the method has been explained previously so it doesn’t need to be repeated.

1. Entry Window

This is the window that is shown when the chi-squared test is to be done. It enables the user to enter in the required data for the test to be carried out.

###The chi-squared statistic test main window

def chiSquaredWindow(self):

#The line below destroys the main window

main.destroy()

global cswindow

global cscategoriesEntry

#These four lines of code set up the chiSquaredWindow with all its features

cswindow = Toplevel(root)

cswindow.focus\_force()

cswindow.configure(background = "thistle1")

cswindow.geometry("600x500+100+100")

#These two lines below add vertical scrollbar to the window

scrollbar = Scrollbar(cswindow)

scrollbar.pack(side=RIGHT, fill=Y)

csLabel = tkinter.Label(cswindow, text = "CHI-SQUARED", background = "thistle1")

cscategoriesLabel = tkinter.Label(cswindow, text = "How many categories of data were collected? (number of independent variables)", bg = "thistle1")

#This line below is an entry box which is another widget, this is so the user can enter in data

cscategoriesEntry = tkinter.Entry(cswindow)

cscategoryButton = tkinter.Button(cswindow, text = "Enter", command = self.cscategories)

backButton1 = tkinter.Button(cswindow, text = "back to main", command = self.toMainFromcswindow)

csLabel.pack()

cscategoriesLabel.pack()

cscategoriesEntry.pack()

cscategoryButton.pack()

backButton1.pack(side = BOTTOM)

When the enter button is pressed the cscategories method is called and the data for each category can be entered. The code for this method is below.

###This method is used to work out the number of categories of data

###It is for the user to enter in all the information required

def cscategories(self):

global category

global csindependentvarEntry

global csdependentvarEntry

global csSiteEntry

global indVarList

global siteValList

#exception handling of the number of categories

#It tries and gets the correct data type which is an integer from the user

access = False

try:

cate = cscategoriesEntry.get()

category = int(cate)

#If the correct data type isn't entered it comes up with an error box for the user to see what the error is

except:

messagebox.showinfo("ERROR", "You haven't entered an integer (whole number)")

#These two variables below are used to store the data entered in by the user

indVarList = []

siteValList = []

no = 1

#This is a for loop that repeats the entry boxes for the independet variables depending on the number of categories

for x in range (0,category):

csindependentvarLabel = tkinter.Label(cswindow, text = "What was independent variable " + str(no) + " in the investigation?", bg = "thistle1")

csindependentvarEntry = tkinter.Entry(cswindow,width = 30)

#The independent variable name(s) that was entered is(are) added to the indVarList list

indVarList.append(csindependentvarEntry)

csSiteLabel = tkinter.Label(cswindow, text = "The number found at this site?", background = "thistle1")

csSiteEntry = tkinter.Entry(cswindow)

#The value for each site entered is added to the siteValList list

siteValList.append(csSiteEntry)

no = no + 1

csindependentvarLabel.pack()

csindependentvarEntry.pack()

csSiteLabel.pack()

csSiteEntry.pack()

csdependentvarLabel = tkinter.Label(cswindow, text = "What was the dependent variable in the investigation?", bg = "thistle1")

csdependentvarEntry = tkinter.Entry(cswindow,width = 30)

entryButton = tkinter.Button(cswindow, text = "Calculate", command = self.chisquared)

csdependentvarLabel.pack()

csdependentvarEntry.pack()

entryButton.pack()

1. Method

When the calculate button is pressed the chisquared method is called to work out the result of the test. It will display the result along with the degree of freedom and the null hypothesis. It will even evaluate the null hypothesis. The code for this is below.

###This function is where the chi-squared test is worked out

def chisquared(self):

global sumSiteVal

global values

global mean

global C

global A

global sumofCList

global X

global csdnull

global theIndependentVars

global csans

global csdof

#exception handling of the dependent variable

#It tries and gets the correct data type which is a string from the user

try:

csd = csdependentvarEntry.get()

csdnull = str(csd)

#If the correct data type isn't enetered it gives the user an error

except:

messagebox.showinfo("ERROR", "You haven't entered the correct data type (need to enter text)")

independent = []

for csindependentvarEntry in indVarList:

#Checks if a value is definitely entered into the entry box and if not an error is shown

if csindependentvarEntry == "":

messagebox.showinfo("ERROR", "You havent entered any text")

#exception handling independent variable

#It tries and gets a string from the user

try:

csi = csindependentvarEntry.get()

indep = str(csi)

#If a string isn't entered, it sends an error to the user

except:

messagebox.showinfo("ERROR", "You haven't entered the correct data type (need to enter text)")

independent.append(indep)

#this helps improve the layout of the code when it is needed

theIndependentVars = (" and ".join(map(str, independent)))

values = []

for csSiteEntry in siteValList:

#exception handling site values

#An integer needs to be entered by the user otherwise an error come up

try:

v = csSiteEntry.get()

val = int(v)

except:

messagebox.showinfo("ERROR", "You haven't entered an integer (a whole number)")

values.append(int(val))

#The line below gets the sum of the values entered so the mean can be worked out

sumSiteVal = sum(values)

#The next two lines gets the mean observed data

mean = (sumSiteVal)/category

mean = round(mean,4)

#This next block of code does the (observed-mean)^2/mean at each site

C = []

Need to find the value for each category so a FOR loop is used.

for n in range(0, category):

A = ((values[n]-mean)\*(values[n]-mean))/mean

A = round(A,4)

C.append(A)

#This line now gets the sum of the values in both sites

sumofCList = sum(C)

#chi-squared value rounded to 3 decimal places and then converted into a string

#This is so it can be used in the method window that the user can see

X = str(round(sumofCList,3))

#This works out the degree of freedom which is needed for accepting or rejecting the null hypothesis

csdof = category-1

#This gets the critical value of the test being done at a probability of 0.05 (P=0.05)

cscriticalValList = [3.84,5.99,7.82]

for n in range(0,len(cscriticalValList)):

cscriticalVal = cscriticalValList[csdof-1]

#This block of code works out if the the hypothesis significant or not

if float(X) >= float(cscriticalVal):

csans = "The chi-squared value of " + str(X) + " is greater than the critical value of " + str(cscriticalVal)+ "\n so we reject the null hypothesis at a 5% significance level, \n as there is only a 5% probability that the result is due to chance"

else:

csans = "The chi-squared value of " + str(X) + " is less than the critical value of " + str(cscriticalVal) + "\n so we accept the null hypothesis at a 5% significance level, \n as there is a 95% probability that results are due to chance"

#The final result is then shown on screen

chisquaredLabel = tkinter.Label(cswindow, text = "The chi-squared value is: " + X, background = "thistle1")

#The null hypothesis is also shown on screen

csnullHypothesisLabel = tkinter.Label(cswindow, text = "The null hypothesis is:", background = "thistle1")

csnullHypothesisDataLabel = tkinter.Label(cswindow, text = "There is no significant difference in " + str(csdnull) + " in " + str(theIndependentVars), bg = "thistle1")

cscriticalValCompLabel = tkinter.Label(cswindow, text = csans, bg = "thistle1")

#The worked solution can then be seen when a button is clicked

methodButton = tkinter.Button(cswindow, text = "Click here to see the worked solution", command = self.csMethodWindow)

#The graph of this test can then be seen when a button is clicked

graphButton = tkinter.Button(cswindow, text = "Click to see the graph", command = self.csGraph)

chisquaredLabel.pack()

csnullHypothesisLabel.pack()

csnullHypothesisDataLabel.pack()

cscriticalValCompLabel.pack()

methodButton.pack()

graphButton.pack()

1. The graph

This graph is drawn if the graph button is clicked as it will show how the evaluation of the null hypothesis has been done. And the user will be able to see this.

###This function is used to draw a graph of the chi-squared test using the data provided

def csGraph(self):

#95% conidence level or P=0.05

cscriticalValList = [3.84,5.99,7.82]

xPoints = [1,2,3]

#plot the X value

plt.scatter(csdof, float(X))

plt.axis([0.9,xPoints[-1],0.0,float(X)+1])

#plot the 95% confidence level

plt.plot(xPoints,cscriticalValList)

#give the graph some axis titles

plt.xlabel("Degree of Freedom")

plt.ylabel("Chi-Sqauared")

plt.title("The Chi-Squared value and degree of freedom at P=0.05")

plt.show()

* Lincoln Index

This is the final test that can be done. It has 2 main windows, one to enter all the data in and the other to show the method. It also can print out the method. Below the entry window and the method will be explained as showing the method to the user has already been explained so it doesn’t need to be repeated.

1. Entry Window

This is the window that the user sees when the Lincoln index test has been chosen. The code for this test is below.

###This function is the lincoln index main window

def lincolnIndexWindow(self):

#Here the main window is destroyed

main.destroy()

global liwindow

global lincoln

global firsttotalEntry

global secondtotalEntry

global fromFirstEntry

var = StringVar()

#Here are the characteristics of the window

liwindow = Toplevel(root)

liwindow.focus\_force()

liwindow.configure(background = "thistle1")

liwindow.geometry("400x400+200+200")

liLabel = tkinter.Label(liwindow, text = "LINCOLN INDEX", background = "thistle1")

firsttotalLabel = tkinter.Label(liwindow, text = "How many did you catch in the first sample?", background = "thistle1")

firsttotalEntry = tkinter.Entry(liwindow)

secondtotalLabel = tkinter.Label(liwindow, text = "How many did you catch in the second sample?", background = "thistle1")

secondtotalEntry = tkinter.Entry(liwindow)

fromFirstLabel = tkinter.Label(liwindow, text = "How many from the second sample were also in the first one?", bg = "thistle1")

fromFirstEntry = tkinter.Entry(liwindow)

enterButton = tkinter.Button(liwindow, text = "Calculate", command = self.lincoln)

backButton3 = tkinter.Button(liwindow, text = "back to main", command = self.toMainFromliwindow)

liLabel.pack()

firsttotalLabel.pack()

firsttotalEntry.pack()

secondtotalLabel.pack()

secondtotalEntry.pack()

fromFirstLabel.pack()

fromFirstEntry.pack()

enterButton.pack()

backButton3.pack(side = BOTTOM)

As can be seen from the code above this is a simpler test than the other three so additional methods for the independent and dependent variables aren’t needed.

1. Method

When the calculate button is clicked the Lincoln method is called. This can be seen below.

###The lincoln index working out window

def lincoln(self):

global n

global s01

global s02

global r

global N

#exception handling first sample total

#It tries to get an integer, otherwise it shows a error message

try:

s01 = firsttotalEntry.get()

#s01 = 0

s1 = int(s01)

except:

messagebox.showinfo("ERROR", "You haven't entered an integer (a whole number)")

#exception handling second sample total

#It tries to get an integer otherwise an error is shown

try:

s02 = secondtotalEntry.get()

#s02 = 0

s2 = int(s02)

except:

messagebox.showinfo("ERROR", "You haven't entered an integer (a whole number)")

#exception handling the second ones in the first one

#It tries to get an integer or an error message is shown

try:

r = fromFirstEntry.get()

R = int(r)

except:

messagebox.showinfo("ERROR", "You haven't entered an integer (a whole number)")

if r != 0:

The result has been rounded to 4 decimal places

n =(s1)\*(s2)/(R)

n = round(n,4)

N = str(n)

else:

n = 0

N = str(n)

lincolnLabel = tkinter.Label(liwindow, text = "The Estimated Population is " + N, background = "thistle1")

methodButton = tkinter.Button(liwindow, text = "Click here to see the full method", command = self.liMethodWindow)

lincolnLabel.pack()

methodButton.pack()

* Main Program

#MAIN PROGRAM#

root = Tk()

root.iconify()

#call the mainWindow def to get the main tkinter screen

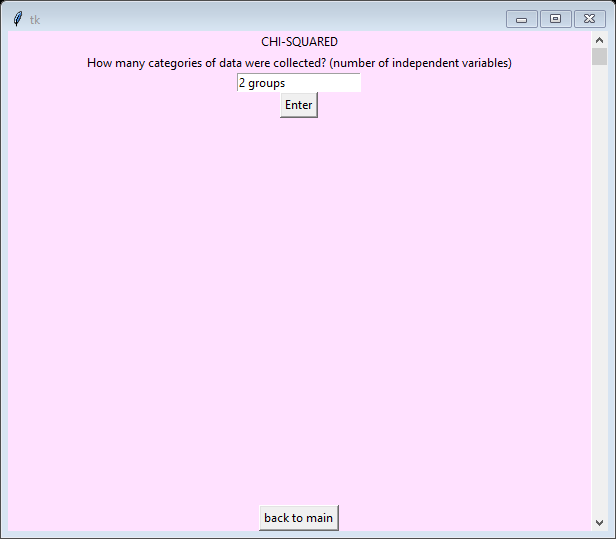
Statistics()

#add all code before this line below - should only work if this is the last thing

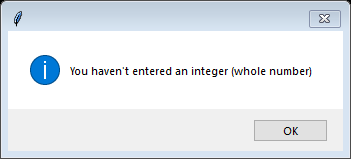
root.mainloop()

* Overview

Overall the implementation went well but some things needed to be explained a bit further. One thing that needs to be explained is exception handling. This was used throughout the code so everything the user enters can be checked. This ensures the correct data type is entered which prevents errors in the rest of the code. An example of this can be seen in figure2 and figure3.



***Figure10, showing the incorrect data type being entered into the entry box.***



***Figure11, showing the error message that the user sees when enter is clicked.***

As can be seen having these error messages is very important as they explain what the user needs to do to get rid of the error. Another thing that needs to be mentioned is that there are global variables being used. This is so the variables created in one method can be used inside other methods. Another thing that needs to be mentioned is that widgets need to be added to the window created in order that they will appear. These widgets are the labels, buttons, entry boxes and radio buttons. These widgets are added using pack() as been previously explained. The last thing that needs to be mentioned is that only one window is ever open at a time. For example, when the program is run and a test is selected the main menu window closes down automatically and the new test window opens up. This is to prevent lots of windows being open at the same time which would make it difficult to navigate fluently through the software.

Evaluation

The code for this report works through any issues that could occur like any errors with data entered or any problems with there being large data sets. The graphs are all working well and were tested separately before being added to the main program. Having each feature of the statistic test in its own method was a really good idea because it provided a very simple way to track any errors through the coding process.

To make the code more effective, having a class for each statistical test would be a good idea so then everything to do with that one test is done all together in that class. This would be something to look at doing in the future. Another thing that could be done is to add more statistical tests or include more critical values so the null hypothesis can be tested at a range of probabilities. This would broaden the projects usage making it useful to more people.

Reference list

<https://geographyfieldwork.com/SpearmansRank.htm>

18/11/2017

<http://mei.org.uk/files/pdf/formula_book_mf2.pdf>

20/11/2017

<https://digital.library.adelaide.edu.au/dspace/bitstream/2440/10701/1/stat_tab.pdf>

<http://hdl.handle.net/2440/10701> - the cite website

20/11/2017

<https://ncalculators.com/math-worksheets/how-to-calculate-t-test.htm>

10/11/2017

<https://whereistheevidence.wordpress.com/articles/statistics/chi-square/>

10/11/2017

CGP, (2015), Essential Maths Skills for A-Level Biology, CGP

1/11/2017