

Total Score:

Question

a)

b)

c)

d)

e)

f)

g)

h)

**BUS252**

**Python for Data Analytics**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Tutor-Marked Assignment**

**July 2021 Presentation** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Name: Chia Wei Hao, Terry**

**Student ID: H1870852**

**Submission Date: 15 August 2021**

1. import math
2. while True:

User\_mean = input("Enter a numeric data value for mean between minus infinity and plus infinity ")

try:

if User\_mean.isspace() == True or User\_mean == "":

User\_mean = 0

else:

User\_mean = float(User\_mean)

except ValueError:

print("Please enter a numeric value for mean ")

False

else:

break

while True:

User\_var = input("Enter a numeric data value for variance that is greater than zero ")

try:

if User\_var.isspace() == True or User\_var == "":

User\_var = 1

else:

User\_var = float(User\_var)

except ValueError:

print("Please enter a numeric value for variance ")

False

else:

if User\_var <= 0:

print ("Please enter value greater than zero ")

else:

break

#print (User\_mean,User\_var)

Explanation:

**Program logic for mean/variance:**

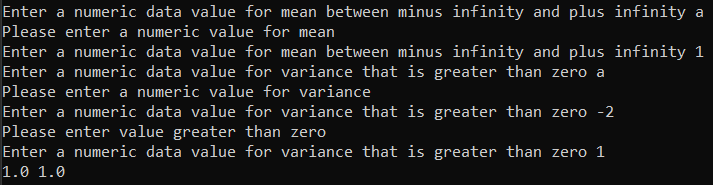
Using “while True”, the program will check if the value entered by the users agrees with the condition in the following logic:

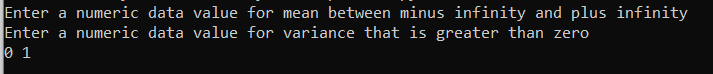
1. If the entered data value for mean is blank or space, the program will default the data value to 0 instead. Additionally, the entered data value should be numeric.
2. If the entered data value for variance is blank or space, the program will default the data value to 1 instead. Additionally, the entered data value should be numeric.

By using float instead of integer command, the program will allow users to enter data values with decimal points instead of just integers. This will allow for more flexibility and is more applicable in real-life scenarios where not all data values are whole numbers.

If the above logic returns false, the program will prompt users to re-enter the data value again as shown in the output screenshot if “a” is entered.

**Additional program logic for variance**Similarly as the program logic for mean, the logic for variance includes an additional check to ensure that the data value entered is greater than zero. If not, the program will prompt users to re-enter the data value as shown in the output screen below.

**Output (for 1b):** 

**Output for blank input(1b):**  


The above output shows the different scenarios that the control mechanism will set in to check on the input data values by the user.   
A print command (last line in output screen) is used to check the inputs entered by the user to ensure that the program records the inputs correctly.

1. while True:

User\_x = input("Enter a numeric data value for X between minus infinity and plus infinity ")

try:

User\_x = float(User\_x)

except ValueError:

print("Please enter numeric value for X ")

False

else:

break

#print (User\_x)

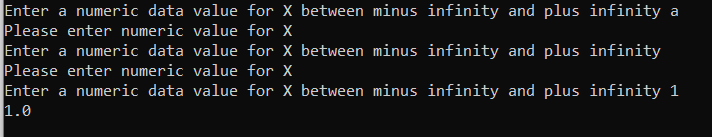
Explanation:

Using “while True”, the program will check if the value entered by the users agrees with the condition in the following logic:

1. The entered data value should be a numeric data value
2. The entered data value is not blank or space.

If the above logic returns false, the program will prompt users to re-enter the data value again as shown in the output screenshot if “a” or “(blank)” is entered.

Output:



The above output shows the different scenarios that the control mechanism will set in to check on the input data values by the user.   
A print command is used to check the inputs entered by the user to ensure that the program records the inputs correctly.

1. from math import pi

from math import exp

def calculate\_pdf(x, mean, variance):

for the formula as compared to declaring the pi value

first\_part = (1.0 / (math.sqrt(variance \* (2.0 \* math.pi))))

exponential = math.exp(-1.0 \* (x - mean)\*\*2 / (2.0 \* variance))

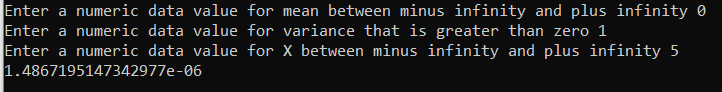
return first\_part \* exponential  
#print(calculate\_pdf(User\_x, User\_mean, User\_var))

results = calculate\_pdf(User\_x, User\_mean, User\_var)

The Pi and exponential function is imported from the math function as declared at the start.   
The user defined function is set as calculate\_pdf with the set variables declared.   
The Probability Density Function (PDF) formula is broken into 2 parts (first\_part and exponential) to minimise confusion and to ensure that the calculation is done correctly in the defined function.   
Additionally, instead of declaring and hard coding the Pi value to be a set value or coding the exponential calculation, the math.pi was used to get a more accurate result and the math.exp was used to calculate the exponential portion of the formula. By doing so, this should eliminate any potential typos or erroneous program logic.

The result of the calculation is then declared as results.

Output:



A print command is used to display the output calculated by the program.

1. print(

"The PDF result is ", results, "\nusing:"

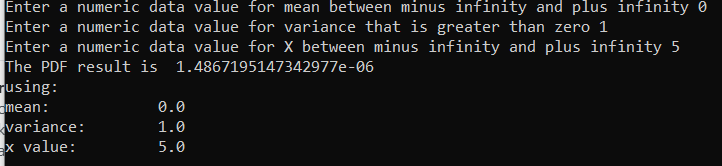
f"\n{'mean:':<10}{User\_mean:>10}",

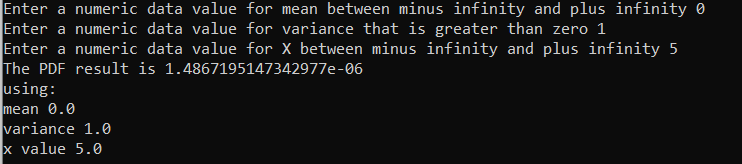
f"\n{'variance:':<10}{User\_var:>10}",

f"\n{'x value:':<10}{User\_x:>10}")

#print("The PDF result is {} \nusing: \nmean {} \nvariance {} \nx value {}" .format(results,User\_mean,User\_var,User\_x))

The print function is used to display the results as shown in the output screenshot below. A standardised format is obtained by declaring that there are fixed width (fixed character lengths) for each of the column. In this case, the left columns (mean,variance and x value) are set to be 10 characters long and the right columns; the user inputs, (User\_mean, User\_var, User\_x) are set to be 10 characters long as well. The display results is more effective in conveying the same data to the users due to the simpler and organized format of the display.   
When comparing the to Output using .format function, the earlier method is more effective and thus, is chosen as the output for this program instead.

Output: 

Output using .format:  


(this print format is not used and is greyed out using # as shown in the codes above.)

1. def float\_range(start, end = None, inc = None):

if end == None:

end = start + 0.0

start = 0.0

if inc == None:

inc = 1.0

range\_list = []

while 1:

Step\_range = start + len(range\_list) \* inc

if inc > 0 and Step\_range >= end:

break

elif inc < 0 and Step\_range <= end:

break

range\_list.append(Step\_range)

return range\_list

cdf\_result = 0

sum = 0

alpha = 0.001

a = -1000

for xStep in float\_range(a, User\_x ,alpha):

cdf\_result = calculate\_pdf(xStep,User\_mean,User\_var)

sum = sum+cdf\_result

finalResult = sum \* alpha

print("The CDF value for your inputs are")

print(finalResult)

cdf\_k0 = 0

sumk0 = 0

for xStep in float\_range(a, 0 ,alpha):

cdf\_k0 = calculate\_pdf(xStep,0,1)

sumk0 = sumk0+cdf\_k0

Resultk0 = sumk0 \* alpha

cdf\_k164 = 0

sumk164 = 0

for xStep in float\_range(a, 1.64 ,alpha):

cdf\_k164 = calculate\_pdf(xStep,0,1)

sumk164 = sumk164+cdf\_k164

Resultk164 = sumk164 \* alpha

cdf\_k196 = 0

sum3k196= 0

for xStep in float\_range(a, 1.96 ,alpha):

cdf\_k196 = calculate\_pdf(xStep,0,1)

sum3k196 = sum3k196+cdf\_k196

Resultk196 = sum3k196 \* alpha

print("when k = 0, CDF is")

print(Resultk0)

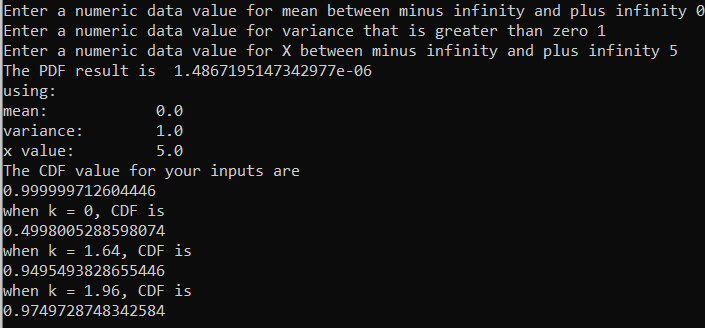
print("when k = 1.64, CDF is")

print(Resultk164)

print("when k = 1.96, CDF is")

print(Resultk196)

Output:



1. When using the range function, the function by default does not allow for float values. However, the data entered by the users in part b and c, can include decimal values, hence, the declared variable is declared as float to account for inputs with decimal places.  
   As such, a user defined range function to account for the float values is used to approach this calculation. The user defined range function will list out the individual values between a to k with the respective increment based on alpha. This list is then used to calculate the individual PDF (Probability Density function) in the range list. Subsequently, the results is then added together and multiplied by the alpha to obtain the CDF (Cumulative Distribution Function) . The k values are substituted into the calculations to account for checks when k values are defaulted to 0,1.64,1.96 as per the question requirements.
2. x = {}

for key in float\_range(-5,5,0.1):

valueForKey = calculate\_pdf(key, 0,1 )

finalKey = float("{0:.2f}".format(key))

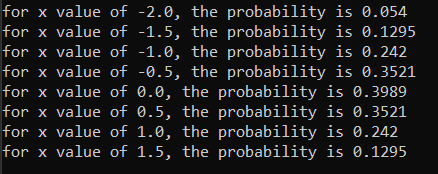
finalvaluekey = float("{0:.4f}".format(valueForKey))

x[finalKey] = finalvaluekey

for printing in float\_range(-2,2,0.5):

print("for x value of {}, the probability is {}" .format(printing,x[printing]))

Explanation:  
The dictionary is declared as x to create a dictionary to capture the results for the distribution based on the provided range of -5 to 5. The default for mean and variance is set to 0 and 1 respectively for the calculation of each key as specified by the question.   
  
The result for each key is then declared as a float to account for the decimal places.  
As the result displays results in intervals of 0.5 from -2 to 2, these results are shortened to 4 decimal places to communicate the results more effectively across to the users.   
A formatted printing is also used to display the results accordingly.

Output:  




# References

*recipes/66472-frange-a-range-function-with-float-increment.* (2001, Aug 7). Retrieved from code.activestate.com: https://code.activestate.com/recipes/66472-frange-a-range-function-with-float-increments/