Logo

Description automatically generated

# ANL252 Python for Data Analytics

**Tutor-Marked Assignment**

**July 2021 Presentation**

**Script:**

#python ANL252\_TMA\_W2110804\_GlenOng\_14August2021.py

#Glen TMA 08Aug2021

#IMPORTING FUNCTIONS

import math

**Description:**

This section imports the math module into the python script to enable mathematical functions can be used for the calculation of the probability density function.

**Script:**

#--------------MAIN PROGRAM--------------

#welcome greeting

print("\n")

print("----- Welcome to the statistical program -----")

print("\nINSTRUCTIONS:\n--------------------")

print("Within this program, you will need to enter the mean, variance and value of x according. Once you have entered valid values for all the variables, you may expect two output results containing the computed probability density and the computed cumulative distribution based on your inputs entered.")

print("--------------------")

#MEAN INPUT

#while-loop for input validation

while True:

#note

print("\n")

print("\*Note that value of mean can be of any numerical number between minus infinity and plus infinity.")

#question for user

mean = input("Enter the mean of the distribution: ")

#conditional statement block

#testing for blank input values and assigning 0 to the value

if mean == "":

mean = 0

#leaving loop

break

#try-except block

try:

float(mean)

except ValueError:

#print error message

print("\n")

print("Invalid value entered! Please ensure that the mean entered is as of a numerical value!")

#user to input again

continue

#if user input is not blank, remain the same value

else:

#leaving loop

break

#VARIANCE INPUT

#while-loop for input validation

while True:

#note

print("\n")

print("\*Note that value of variance can be of any numerical number larger than 0.")

#question for user

variance = input("Enter the variance of the distribution: ")

#conditional statement block

#testing for blank input values and assigning 0 to the value

if variance == "":

variance = 1

#leaving loop

break

#try-except block

try:

float(variance)

except ValueError:

#print error message

print("\n")

print("Invalid value entered! Please ensure that the variance entered is as of a numerical value!")

#user to input again

continue

#conditional statement block

#testing for values is larger than 0

if float(variance) <= 0:

#print error message

print("\n")

print("Invalid value entered! Please ensure that the value of the variance entered is more than 0!")

#user to input again

continue

#if user input is not blank, remain the same value

else:

#leaving loop

break

**Description:**

The main program contains user inputs and user validations. The main program starts with an introduction to the program and what lets the user know what to expect, followed by a while-loop containing the user input variable for mean, a conditional statement to check for blank input which will be converted to the value 0 if it is found to be true, and a try except block to validate if the mean is of numerical value.

Once the user input mean value suffices the conditions, another while-loop is executed. This while loop contains the user input variable for variance, it follows a similar logic with the mean input loop as described above. However, the blank input for variance will be converted to the value of 1 instead of 0 and followed by another conditional check to ensure the variance inputted is not 0 or less than 0.

Lastly, if a value does not meet the desired conditions, the respective while-loop is repeated. Additional print statements are created to ensure user know what was the error before re-entering another value. The respective loops will not be exited until the values suffices the conditions.

**Output:**

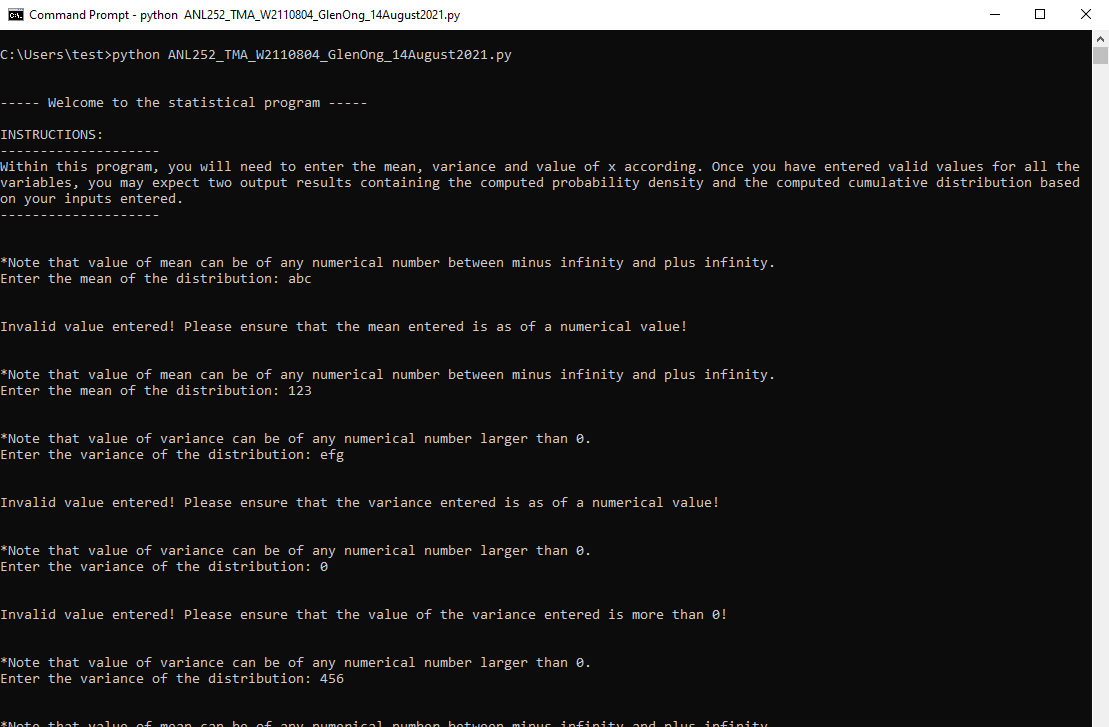


Figure b.1 sample output results of the mean and variance user input and user input validation

**Script:**

#VALUE OF RANDOM VARIABLE INPUT

#while-loop for input validation

while True:

#note

print("\n")

print("\*Note that value of mean can be of any numerical number between minus infinity and plus infinity.")

x = input("Enter the value of the random variable (X): ")

#try-except block

try:

float(x)

except ValueError:

#print error message

print("\n")

print("Invalid value entered! Please ensure that the value of the random variable entered is as of a numerical value!")

#user to input again

continue

#if value type is of a numerical value

else:

#leaving the loop

break

**Description:**

Unlike the mean user input while-loop block and variance user input while-loop block, the “x” user input does not contain any conditional statements. It only validates the user input for numerical values through a try-except nested in a while-loop.

However, similar logic still applies for the user input variable for x while-loop, whereby it will only exit the while-loop once the input value suffices the condition of numerical values.

**Output:**

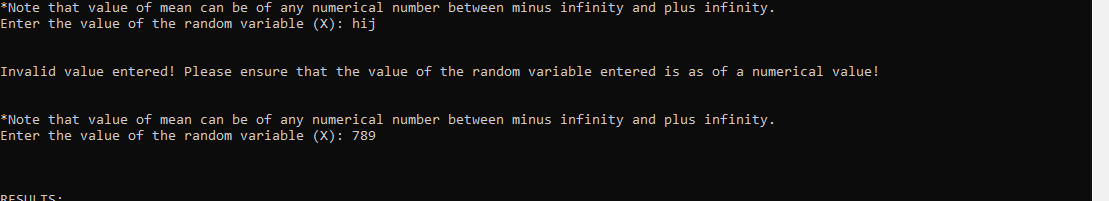


Figure c.1 sample output results of the x user input and user input validation

**Script:**

#--------------MATHEMETICAL FORMULA--------------

#probability density function (pdf) of the normal distribution

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

#probability density function calculation

pdf = (1 / math.sqrt(2 \* math.pi \* float(variance))) \* (math.exp((-(math.pow((float(x) - float(mean)),2))) / (2 \* float(variance))))

**Description:**

The mathematical formula is stored within a define function for the calculation of the probability density function. This mathematical function is converted from the formula provided by the question and leverage on the math module imported at the start of the script. This would enable the use of “math.sqrt”, “math.pi”, “math.exp”, and “math.pow” which translate the square root, pi, exponential, and power respectively.

The entire formula is stored under the “pdf” variable which also suggest that the respective value would be stored within this variable.

**Script:**

#printing based on user input values and outcome for probability density function

print("\n\n")

print("RESULTS:\n--------------------")

print(f"The probability density fx(x) based on your mean({mean}), variance({variance}) and value of x({x}) is {pdf}.")

#conditional statements to determine which decimal places it should be rounded off to

#pdf values that are able to round of to 4 decimal places will be rounded off the 4 decimal places

if round(pdf, 4) > 0:

print(f"The rounded of value to four decimal places is {pdf:.4f}(4 decimal places).")

if round(pdf, 2) > 0:

print(f"Additionally, the rounded of value to two decimal places is {pdf:.2f}(2 decimal places).")

print("--------------------")

else:

print("The value is too small and thus insignificant to round off.")

print("--------------------")

**Description:**

The script then prints the respective user inputted values of mean, variance and x and the corresponding pdf value derived from the variables. Additionally, conditional statements are set up to check the value decimal places to decide if the values are suitable to be rounded into 4 decimal places, 2 decimal places and if the value should not be rounded off due to a small derived value.

The condition contains an if-else statement with a nested if-else statement. If we were to observe a derived value of 0.0032, it will not meet the nested if-else statement for the transformation to 2 decimal places and would remain at 4 decimal places. Additionally, if a derived value is 0.00003, it will not meet the first condition of 4 decimal places and thus not round the derived value as it would be too insignificant.

**Output:**

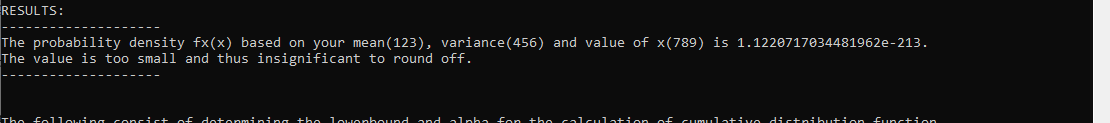


Figure e.1 sample formatted printing output based on the user-inputs

**Script:**

#cumulative distribution function (cdf) of the normal distribution

def function\_cdf(x, variance, mean):

#cumulative distribution function calculation (prerequisite)

#additional information for user

print("\n")

print("The following consist of determining the lowerbound and alpha for the calculation of cumulative distribution function")

#user input validation

#defining variables

#taking upperbound from previously defined x value

upperbound = int(x)

#creation cdf variables for looping

cdf = 0

t = x

#defining presets for mean and variance

preset\_mean1 = 0

preset\_variance1 = 1

#defining presets for the list upperbounds

CI\_50 = 0

CI\_95 = 1.64

CI\_98 = 1.960

#cdf dictionary

dictionary\_cdf = {}

#defining presets for lowerbound and upperbound

preset\_lowerbound1 = -5

preset\_upperbound1 = 5

preset\_lowerbound2 = -2

preset\_upperbound2 = 2

#stepsize

stepsize1 = 0.01

stepsize2 = 0.1

#LOWERBOUND

while True:

#User to input values for cdf specific variable

#note

print("\n")

print(f"\*Note that value of lowerbound(X) can be of any numerical number between minus infinity and {x} as previously defined.")

#question for user

lowerbound = input("Enter the lowerbound(X) of the integral: ")

#try-except block

try:

float(lowerbound)

except ValueError:

#print error message

print("\n")

print("Invalid value entered! Please ensure that the lowerbound entered is as of a numerical value!")

#user to input again

continue

#exiting loop

if float(lowerbound) <= float(x):

lowerbound = float(lowerbound)

break

else:

print("\n")

print(f"Invalid value entered! Please ensure that the lowerbound entered is less than or equal to {x}!")

continue

#preserving raw lowerbound input

lowerbound\_1 = lowerbound

lowerbound\_2 = lowerbound

#ALPHA

while True:

#User to input values for cdf specific variable

#note

print("\n")

print(f"\*Note that value of alpha can only be significance level can only be 0.10, 0.05 and 0.01.")

#question for user

alpha = input("Enter the significance level: ")

#try-except block

try:

float(alpha)

except ValueError:

#print error message

print("\n")

print("Invalid value entered! Please ensure that the significance level entered is as of a numerical value!")

#user to input again

continue

#exiting loop

if alpha == "0.10" or alpha == "0.1" or alpha == "0.05" or alpha == "0.01":

alpha = float(alpha)

break

else:

print("\n")

print(f"Invalid value entered! Please ensure that the significance level entered is 0.10, 0.05 or 0.01!")

continue

#printing the assumptions made to conduct the calculation

print("\n\n")

print("RESULTS:\n--------------------")

print(f"With a lower-bound of {lowerbound} with a significance level of {alpha}.")

#for cdf to loop through all the steps sizes

#while-loop for the calculation of cdf

while lowerbound\_1 <= upperbound:

#calculation of probability density function (prerequisite) for individual lowerbound values

pdf = (1 / math.sqrt(2 \* math.pi \* float(variance))) \* (math.exp((-(math.pow((float(lowerbound\_1) - float(mean)),2))) / (2 \* float(variance))))

#summing the individual values to cdf

cdf = cdf + (pdf \* alpha)

lowerbound\_1 = round(lowerbound\_1,2) + float(stepsize1)

#printing based on user input values and outcome for cumulative distribution function

print(f"With a significance level of {alpha}, the cumulative distribution Fx(x) based on your mean({mean}), variance({variance}) and value of lowerbound(x = {lowerbound}) is {cdf:.4f}.")

#resetting cdf value to 0 for next loop

cdf = 0

#Glen edited on the 12Aug2021

#While-loop to get cdf values for specific k values

while lowerbound\_2 <= CI\_98:

#calculation of probability density function (prerequisite) for individual lowerbound values

pdf = (1 / math.sqrt(2 \* math.pi \* float(preset\_variance1))) \* (math.exp((-(math.pow((float(lowerbound\_2) - float(preset\_mean1)),2))) / (2 \* float(preset\_variance1))))

#summing the individual values to cdf

cdf = cdf + (pdf \* alpha)

lowerbound\_2 = round(lowerbound\_2,2) + float(stepsize1)

#printing based on user input values and outcome for cumulative distribution function

if float(lowerbound\_2) == CI\_50:

print("\n")

print(f"With a significance level of {alpha}, the cumulative distribution Fx(x) based on your mean({preset\_mean1}), variance({preset\_variance1}) and value of upperbound(k = {lowerbound\_2}) is {cdf:.4f}(4 decimal places).")

elif float(lowerbound\_2) == CI\_95:

print("\n")

print(f"With a significance level of {alpha}, the cumulative distribution Fx(x) based on your mean({preset\_mean1}), variance({preset\_variance1}) and value of upperbound(k = {lowerbound\_2}) is {cdf:.4f}(4 decimal places).")

elif float(lowerbound\_2) == CI\_98:

print("\n")

print(f"With a significance level of {alpha}, the cumulative distribution Fx(x) based on your mean({preset\_mean1}), variance({preset\_variance1}) and value of upperbound(k = {lowerbound\_2}) is {cdf:.4f}(4 decimal places).")

**Output:**

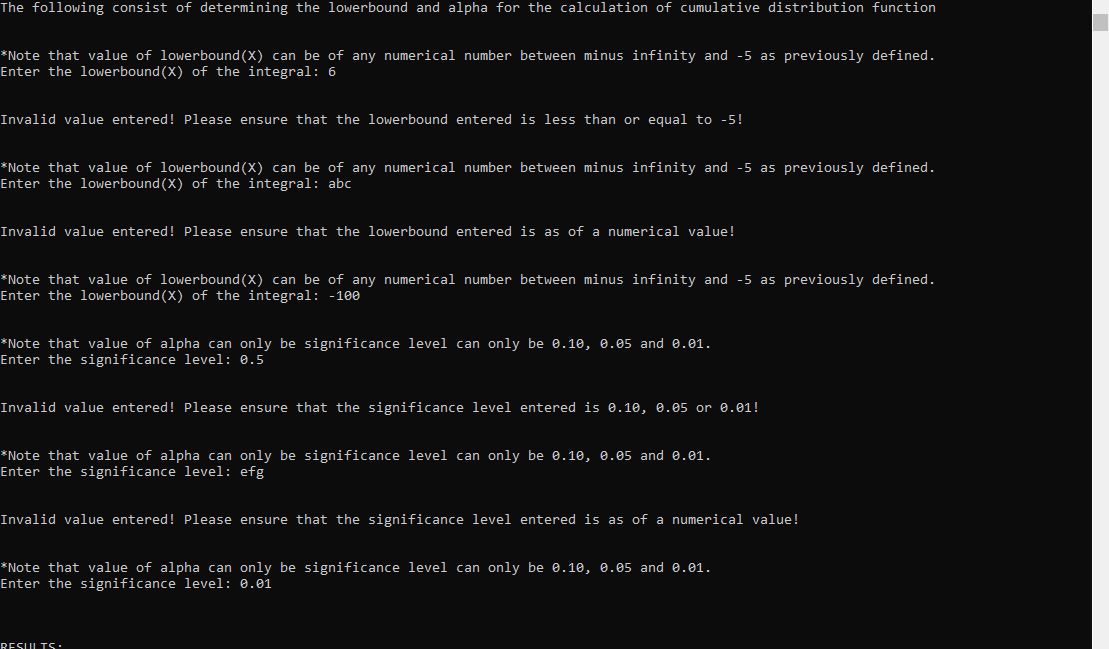


Figure f.1 sample output results of the lower bound(x) and significance level(a) user input and user input validation

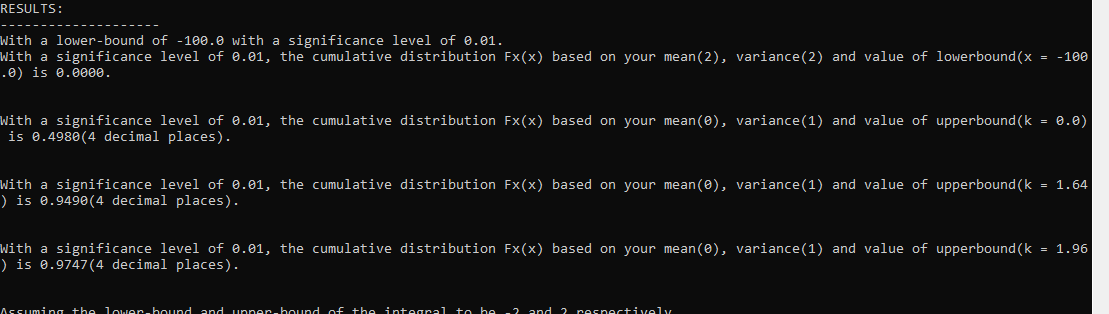


Figure f.2 sample formatted printing output based on the user-inputs

Following the creation of the user-defined function for cumulative distribution function (cdf), the program defines the prerequisite variables needed for cdf such as the cdf variable with a value of 0. The program then requires the user input for lower bound and significance level (alpha) value with conditions in place to ensure both values entered by the user are numeric. This is achieved through a nested try-except block within a while-loop. Additionally, within each while-loop block of the lower bound input and alpha input, specific conditional statements are created to further enforce the desired variables for the user-inputs.

Leveraging on the user-inputted values of the mean and variance from the main program, and the user-inputted values from the user user-defined function for cdf and the prerequisite variables, a while-loop is used to iterate through each lower bound value less than or equal to the upper bound value. Within each iteration, the probability density function for the corresponding lower bound is calculated and multiplied by the alpha as defined in the user input before adding the multiplied value into the “cdf” variable. The lower bound value is then incremented by the step size of 0.01.

194 words

**Script:**

#Glen edited on the 14Aug2021

#while loop to store dictionary with keys between -5 to 5 if x value entered is within keys range and the respective cpf values for x

while float(preset\_lowerbound1) <= float(t) <= float(preset\_upperbound1):

#calculation of probability density function (prerequisite) for individual lowerbound values

pdf = (1 / math.sqrt(2 \* math.pi \* float(preset\_variance1))) \* (math.exp((-(math.pow((float(t) - float(preset\_mean1)),2))) / (2 \* float(preset\_variance1))))

#summing the individual values to cdf

cdf = cdf + (pdf \* alpha)

dictionary\_cdf[str(round(float(t),2))] = round(cdf,4)

t = round(float(t),2) + float(stepsize2)

#new dictionary to filter for keys between -2 and 2 and the respective cpf values for x that is stored in the previous dictionary

new\_dictionary\_cdf = {k:v for k,v in filter(lambda sub: float(sub[0]) >= preset\_lowerbound2 and float(sub[0]) <= preset\_upperbound2, dictionary\_cdf.items())}

#filtering for

filtered\_dictionary\_cdf = {k:v for (k,v) in new\_dictionary\_cdf.items() if str(abs(float(k)))[2:] == "0" or str(abs(float(k)))[2:] == "5"}

#conditional printing

#if there are values in the new dictionary print the following

if len(filtered\_dictionary\_cdf) > 0:

print("\n")

print(f"Assuming the lower-bound and upper-bound of the integral to be {preset\_lowerbound2} and {preset\_upperbound2} respectively.")

print(f"The probabilities of the normal distribution for x({x}) between {preset\_lowerbound2} and {preset\_upperbound2} are {filtered\_dictionary\_cdf}")

#if there are no values in the new dictionary print the following

elif len(filtered\_dictionary\_cdf) == 0:

print("\n")

print(f"Assuming the lower-bound and upper-bound of the integral to be {preset\_lowerbound2} and {preset\_upperbound2} respectively.")

print(f"There are no dictionary to display! The probabilities of the normal distribution for x({x}) does not fall between {preset\_lowerbound2} and {preset\_upperbound2}.")

print("--------------------")

**Description:**

Part (h) follows a similar logic as described in part (g) in the calculation of cumulative distribution function (cdf). However, within each iteration, the “x” values will be stored as keys and the respective cdf value will be stored as values into the dictionary. Once all the keys from -5 to 5 and the respective value is stored, a new dictionary is created to filter out keys from -2 and 2 and the respective values. Another dictionary is also created to filter out values containing .5 and .0 from the new dictionary to ensure a step size of 0.5 keys and the respective values are being stored in the filtered dictionary.

**Output:**

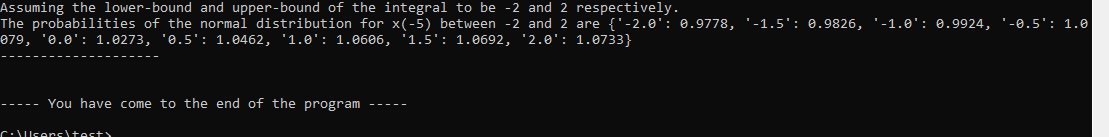


Figure h.1 sample filtered dictionary output based on the x user-input

# Appendix:

