**ANL252**

**Python for Data Analytics**

**Tutor-Marked Assignment (TMA01)**

**July 2021 Presentation**

|  |  |
| --- | --- |
| Name | Cheong Jia En, Sandras |
| PI No. | N1981099 |
| Class | T09 |
| Submission Date | 13 August 2021 |

(a)

**Code**

# Importing math package

import math

(b)

**Code**

# Message for user to read before keying their inputs

Message = [

"Mean can be any value between minus infinity and positive infinity.",

"Variance must be larger than 0.",

"Press Enter to have default Mean = 0 and Variance = 1."

]

for i in Message: # For loop using the list of messages to user

print(i)

# User to provide values for Mean and Variance of distribution

# Error handling

while True:

mean = input("\nPlease enter Mean of distribution: ") # User input

if len(mean) == 0: # If user press enter without keying any value, the length of user input is 0,

mean = 0 # Mean will then be set to 0 as default

print("Your Mean is 0.")

break

else:

try:

mean = float(mean) # If user entered a value, the input should be a number (float)

except ValueError:

print("Please key in a numerical value.") # Error message to user if input is a string

else:

print(f"Your Mean is {mean}.")

break

while True:

variance = input("\nPlease enter Variance of distribution: ")

if len(variance) == 0: # If user press enter without keying any value, the length of user input is 0,

variance = 1 # Variance will then be set to 1 as default

print("Your Variance is 1.")

break

else:

try:

variance = float(variance) # If user entered a value, the input should be a number (float)

except ValueError:

print("Please key in a numerical value.") # Error message to user if input is a string

else:

if variance <= 0: # If user input is negative or equal to 0

print("Your Variance must be larger than 0. Please try again.") # Error message to user

else:

print(f"Your Variance is {variance}.")

break

**Output**

When user press Enter to set Mean and Variance to 0 and 1 respectively, the output is as follows:

Graphical user interface, text, application

Description automatically generated

When user entered a non-numeric input, an error message is thrown, and user is asked to re-enter Mean again. Likewise for Variance. However, if user entered value of 0 or negative values for Variance, another error message will be thrown and the user is required to re-enter Variance again.

Text, letter

Description automatically generated

(c)

**Code**

# Message for user to read before entering X value

print(f"\nX can be any value between minus infinity and plus infinity.")

# Error handling

valid = True

while True:

try:

X = float(input("\nPlease enter X value: "))

except ValueError:

print(f"Please key in a numerical value.") # Error message when user input is a string

else:

print(f"X is {X}.")

break

**Output**

If user failed to enter a value and press Enter key instead, an error message is thrown. The same message is thrown if user enters a non-numerical value.

Text

Description automatically generated

(d & e)

**Code**

import math

def pdf\_formula(X, mean, variance): # Defining probability density function (pdf)

return 1/math.sqrt(2\*math.pi\*variance)\*math.exp(-(X-mean)\*\*2/(2\*variance)) # pdf formula using user inputs in (b) and (c)

fx = pdf\_formula(X, mean, variance)

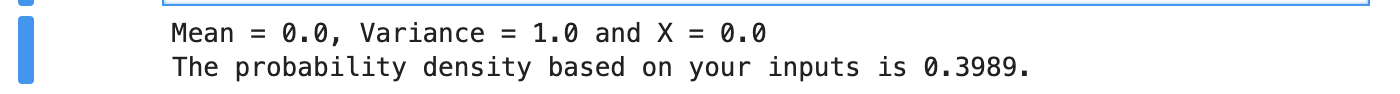
#Formatted printing

print(f"\nMean = {mean}, Variance = {variance} and X = {X}")

print(f"The probability density based on your inputs is {fx:.4f}.")

**Output**

A user-defined function for probability density function (pdf) is created using the pdf formula given. The corresponding probability density fx(x) is computed based on user’s Mean, Variance and X inputs.



(f)

**Code**

alpha = 0.01 # step width

a = -100 # number close to negative infinity

# k refers to the X value that user input in (c), k = X

# small x refers to normally distributed random variable x

def xprob(X, alpha, a): # Defining cumulative distribution function (cdf)

cdf = 0

i = 0

terms = (X-a)/alpha # No. of terms between a and X

while i<=terms:

pdf = pdf\_formula(a+i\*alpha, mean, variance) # pdf of each fx(x) from -100 to 0

cdf = cdf + pdf # Summation of fx(x) from -100 to 0

i = i + 1 # Prevent infinite loop

cdf = cdf\*alpha # Weighted sum of fx(x)

return cdf

cdf = xprob(X, alpha, a)

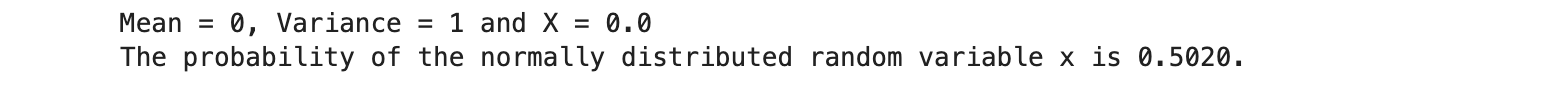
#Formatted printing

print(f"\nMean = {mean}, Variance = {variance} and X = {X}")

print(f"The probability of the normally distributed random variable x is {cdf:.4f}.")

***\*Note: k = X value that user input in (c)***

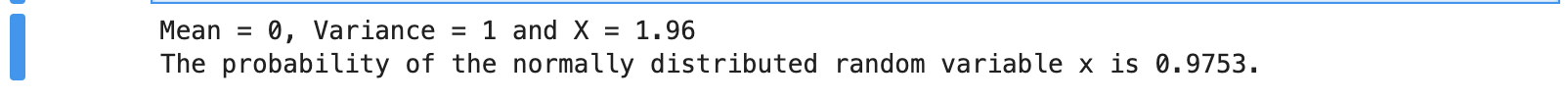
**Output (k = 0)**



**Output (k = 1.64)**



**Output (k = 1.96)**



(g)

Firstly, I defined what the variables represent. I let alpha represent the range of each step, and a represent the number close to negative infinity, which are 0.01 and -100 respectively. I replaced k in the formula with X as it is the user input in (c). Subsequently, I constructed a new user-defined function called the “xprob” using the formula of cumulative distribution function (cdf).

To determine the number of terms between a and X, I created the variable “terms” and its formula which gives us a total of 10000 terms. A while loop is used where the condition is to repeat and execute the instructions as long as “i” is less than or equal to 10000, which explains why I set “i” equal to 0 initially to include fx(0). The instructions are to compute pdf of each term i.e., fx(x) from -100 to 0, then proceed to sum up all the density functions of x. Likewise, I have assigned variable “cdf” a value of 0. Then cdf can be approximated by weighting the summation i.e., multiplying by alpha 0.01. Finally, “cdf” will return the probability of the normally distributed random variable x which is computed using the formula. (199 words)

(h)

**Code**

# define dictionary

xkeys = {}

print(f"\nProbabilities for the corresponding x are as shown: \n")

for x in range(-50, 51, 1): # Range does not allow float so multiply the range and step by 10

x = x/10 # Divide by 10 to achieve the desired keys in float type

xkeys[x] = xprob(x, alpha, a) # Call cdf user defined function from part(f)

valid = False

for x in xkeys.keys():

if -2 <= x <= 2: # If x key is within the range(-2, 2)

if (x % 0.5 == 0): # No remainder to ensure equal 0.5 intervals

valid = True

print(f"{x}: {xkeys[x]:.4f}")

**Output**

**Text

Description automatically generated**