STATISTICS PROJECT – 1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import expon
import warnings
warnings.filterwarnings("ignore")
```

1)Implement the counting measure in Python. Test that it satisfies additivity on the disjoint sets {"a", "b", "c"}, {"d", "e", "f"}. Implementation of the counting measure checks that the input type is correct and raises an error otherwise.

```
set_1 = {"a", "b", "c"}
set_2 = {"d", "e", "f"}
set_3 = ("g")

# A function for counting measure with a check
def counting(input_type):
    if type(input_type) is set:
        sum = 0
        for x in input_type:
            sum += 1
        return sum
    else:
        print("This is not a set !! Please give a correct input..")

print("The counting measure of set_1 is :", counting(set_1))
print("The counting measure of set_2 is :", counting(set_2))
print("The additivity of the both disjoint set is :",
counting(set_1.union(set_2)))
print(counting(set_3))
```

```
The counting measure of set_1 is : 3
The counting measure of set_2 is : 3
The additivity of the both disjoint set is : 6
This is not a set !! Please give a correct input..
None
```

2)Create a Python class which implements intervals. Use this new datatype to write a function which implements the length measure on intervals. Test it on the interval [1,3.5] Implementation of the length measure checks whether the input type is correct or raises an error otherwise.

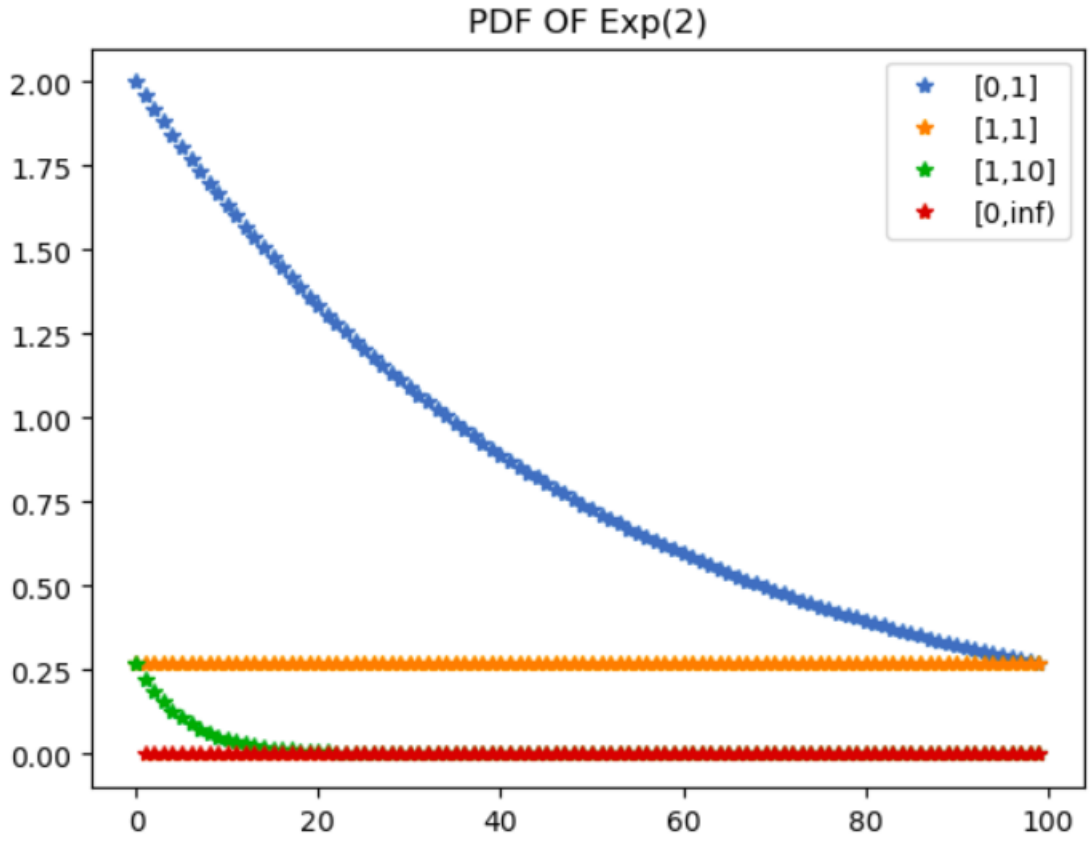
```
# A Class for Intervals
class cls_Interval:
    # A function with a check of input type
    def __init__(self,first,last,closed):
        if(type(first) == int or type(first) == float) and type(last)
== int or type(last) == float and closed in ["both", "left", "right",
"neither"]:
            self.interval = pd.Interval(left = first, right = last,
closed = closed)
        else:
            print("Your Input value is Incorrect !! Please enter the
correct value")

    # A function to calculate the length measure
    def length_measure(self):
            return self.interval.right - self.interval.left


interval_1 = cls_Interval(1,3.5,'both')
interval_2 = cls_Interval('three',1,'both')
print("The length of interval [1,3.5] is", interval_1.length_measure())
```

```
Your Input value is Incorrect !! Please enter the correct value
The length of interval [1,3.5] is 2.5
```

3)Import scipy.stats in order to access the scipy.stats.expon distribution. This implements the exponential distribution Exp(λ. Using the cdf method of scipy.stats.expon define a function called expon_measure which will take as input an interval (defined in the previous question) and will return its probability mass under the probability measure Exp (2) (i.e. λ = 2). Test your function by computing the probability measure of the following intervals: (a) [0, 1] (b) [1, 1] (c) [1, 10] (d) [0, ∞) Plot the pdf of Exp (2) on comment on whether your answers seem to make sense visually.

```
# Calculation of exponential measure using cdf by taking the cdf
difference right and left intervals.
def exponent(interval):
    cdf = expon.cdf(interval.interval.right, 0, 1/2) -
expon.cdf(interval.interval.left, 0, 1/2)
    return cdf

value_1 = cls_Interval(0, 1, 'both')
value_2 = cls_Interval(1, 1, 'both')
value_3 = cls_Interval(1, 10, 'both')
value_4 = cls_Interval(0, float('inf'), 'left')

print("probability mass of a : " ,exponent(value_1))
print("probability mass of b : " ,exponent(value_2))
print("probability mass of c : " ,exponent(value_3))
print("probability mass of d : " ,exponent(value_4))

# Creating values as list to calculate its probability density function
interval_value_1 = np.linspace(0, 1, num =100)
interval_value_2 = np.linspace(1, 1, num =100)
interval_value_3 = np.linspace(1, 10, num =100)
interval_value_4 = np.linspace(0, float('inf'), num =100)

val_1 = expon.pdf(interval_value_1, 0, 0.5)
val_2 = expon.pdf(interval_value_2, 0, 0.5)
val_3 = expon.pdf(interval_value_3, 0, 0.5)
val_4 = expon.pdf(interval_value_4, 0, 0.5)

# Plotting graph for pdf of Exp(2)
plt.plot(val_1, "*", label = "[0,1]")
plt.legend(loc = "upper right")
plt.plot(val_2, "*", label = "[1,1]")
plt.legend(loc = "upper right")
plt.plot(val_3, "*", label = "[1,10]")
plt.legend(loc = "upper right")
plt.plot(val_4, "*", label = "[0,inf)")
plt.legend(loc = "upper right")

plt.title("PDF OF Exp(2)")

plt.show()
```

```
probability mass of a :  0.8646647167633873
probability mass of b :  0.0
probability mass of c :  0.13533528117545912
probability mass of d :  1.0
```



Justification: Yes, The Graph makes sense visually because the graph shows how the probability measure is rolled out across the Interval. We know that the probability measure will not go beyond 1, as we can see here too, as the interval values get bigger the probability measure starts decreasing. Intervals [1,1] and [0,inf) have a straight line because [1,1] has a probability measure of 0 because it is a point, and in the case of [0,inf) the probability measure sums up to 1.

4)Using the pdf method of scipy.stats.expon, define a function called expon_pdf which will take one argument x and return the pdf of the probability measure Exp (2) evaluated at x. Import the integration routine quad from scipy.integrate, and read the documentation https://docs.scipy. org/doc/scipy/reference/generated/scipy.integrate.quad.html to see how it works. Use quad to compute and print the following integrals (a) R 1 0 expon_pdf(x) dx (b) R 1 1 expon_pdf(x) dx (c) R 10 1 expon_pdf(x) dx (d) R ∞ 0 expon_pdf(x) dx Compare your answers with those of the previous question. What do you see? Why is this the case?

```
from scipy import integrate

# A function to calculate pdf value
pdf = lambda i : expon.pdf([i], 0, 1/2)

# A function to Integrate the pdf function with its intervals
def integral_function(low_limit, up_limit, function):
    final = integrate.quad(function, low_limit, up_limit)
    return final[0]

print("Integral a : ", integral_function(0, 1, pdf))
print("Integral b : ", integral_function(1, 1, pdf))
print("Integral c : ", integral_function(1, 10, pdf))
print("Integral d : ", integral_function(0, float('inf'), pdf))
```

```
Integral a :  0.8646647167633873
Integral b :  0.0
Integral c :  0.13533528117545912
Integral d :  0.9999999999999999
```

Justification: so, To Conclude we are getting similar and the same answer as the 3rd Question. This is the expected answer because we are calculating the cdf in both questions but in the different way, So In the previous question we have calculated the cdf by taking the difference of cdf of upper and the lower bound. But in the case of this question we have taken cdf by integrating the interval values.