Tugas Individu Praktikum 1

Mujadid Choirus Surya

121450015

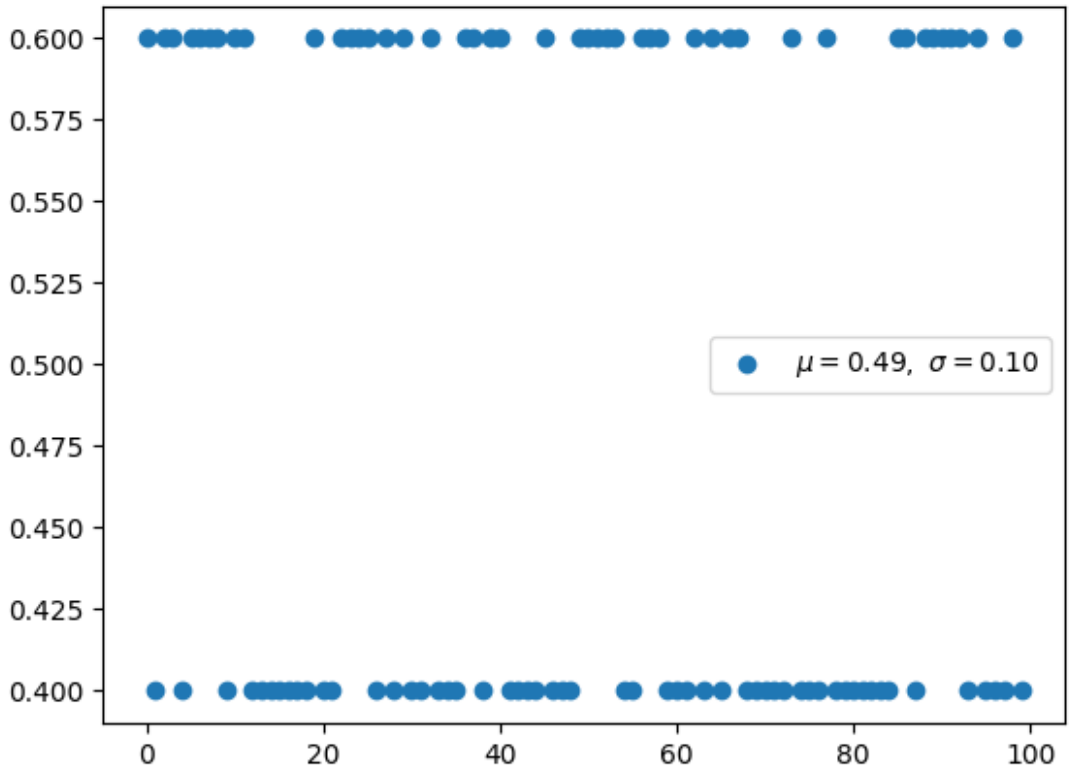
RA

Latihan A1

kumpulkan code dan grafik dalam link disediakan asisten praktikum dalam bentuk ipnyb.

```
import random
import numpy as np
from matplotlib import pyplot as plt

def bernoulli(p,k): # membuat fungsi bernoulli
  return p if k else 1-p #  return dalam fungsi bernoulli. Jika k adalah True (atau 1), maka fungsi akan mengembalikan nilai p. Jika k adalah False (atau 0), maka fungsi akan mengembalikan nilai 1-p

n_experiment = 100 # menentukan jumlah eksperimen yang akan dilakukan
p = 0.6 # deklarasi nilai probabilitas sukses dalam distribusi Bernoulli
x = np.arange(n_experiment) # deklarasi variabel x yang akan berisi array n_experiment. Akan digunakan sebagai sumbu x pada scatter plot
y = [] # Array kosong yang akan digunakan untuk menyimpan hasil dari eksperimen distribusi Bernoulli.

for _ in range(n_experiment): # loop for yang diulang sebanyak n_experiment
  pick = bernoulli(p, k=bool(random.getrandbits(1))) # menggambarkan pengambilan sampel dari distribusi Bernoulli, fungsi dipanggil dengan probabilitas p dan nilai k yang dihasilkan secara acak 0/1
  y.append(pick) # Nilai yang diambil (pick) ditambahkan ke dalam array y

u,s = np.mean(y), np.std(y) # perhitungan rata-rata (u) dan deviasi standar (s) dari array y
plt.scatter(x,y, label= r'$\mu=%.2f,\ \sigma=%.2f$' % (u,s)) # membuat scatter plot, label dari plot berisi rata-rata (u) dan deviasi standar (s)
```

```
plt.legend() # menampilkan legenda pada plot yang menunjukkan label yang telah ditentukan sebelumnya
plt.show() # menampilkan plot
```

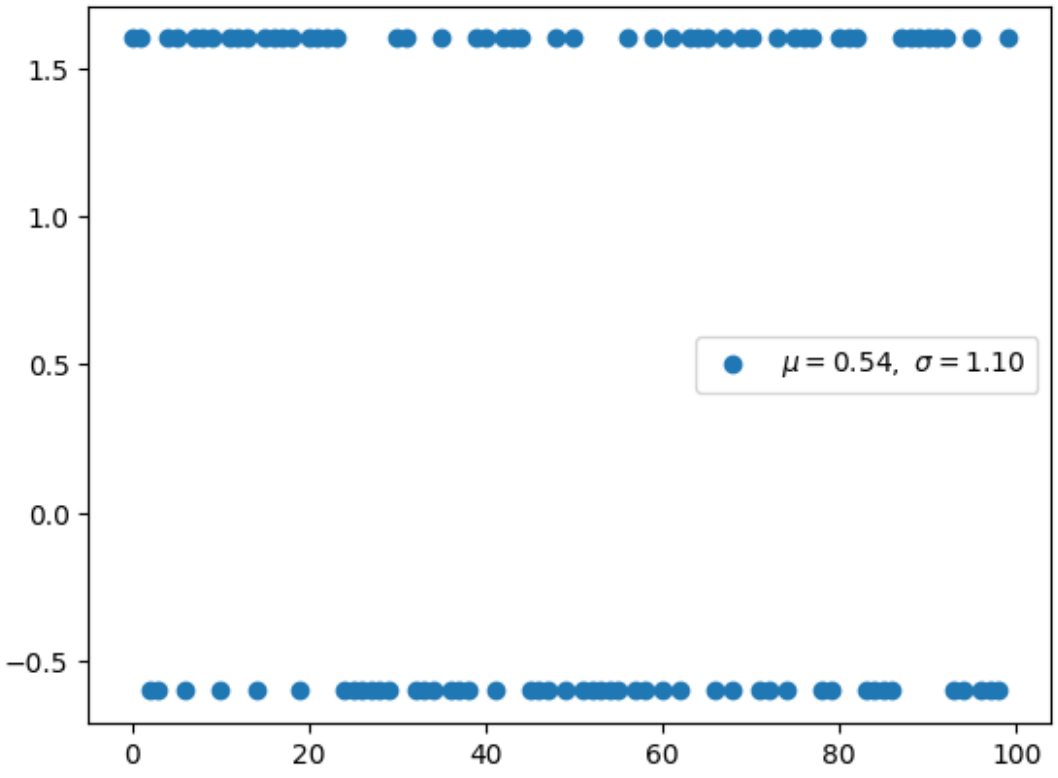


Latihan A2

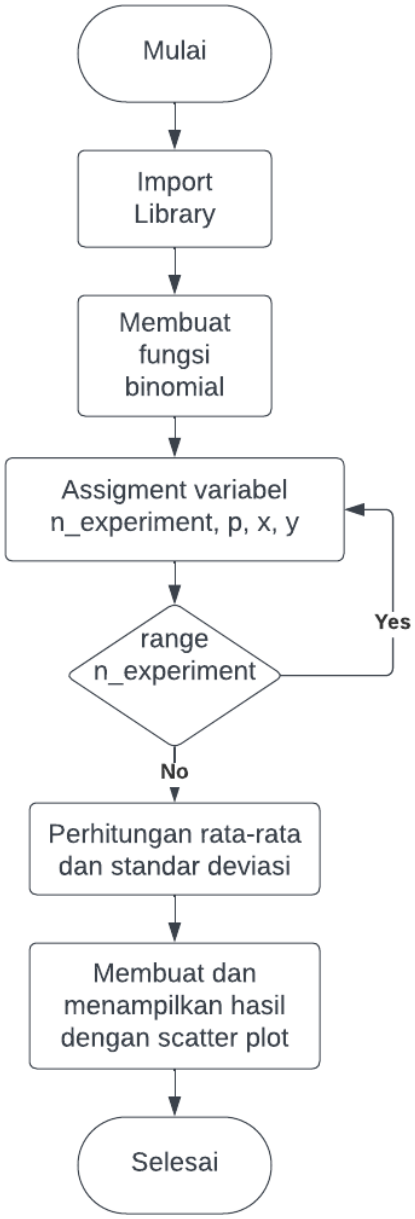
Coba jika p=1,6 berapa nilai μ dan σ , kumpilkan code dan grafik berbentuk ipnyb

```
def bernoulli(p,k): # membuat fungsi bernoulli
  return p if k else 1-p #  return dalam fungsi bernoulli. Jika k adalah True (atau 1), maka fungsi akan mengembalikan nilai p. Jika k adalah False (atau 0), maka fungsi akan mengembalikan nilai 1-p

n_experiment = 100 # menentukan jumlah eksperimen yang akan dilakukan
p = 1.6 # deklarasi nilai probabilitas sukses dalam distribusi Bernoulli
x = np.arange(n_experiment) # deklarasi variabel x yang akan berisi array n_experiment. Akan digunakan sebagai sumbu x pada scatter plot
y = [] # Array kosong yang akan digunakan untuk menyimpan hasil dari eksperimen distribusi Bernoulli.
```

```
for _ in range(n_experiment): # loop for yang diulang sebanyak n_experiment
  pick = bernoulli(p, k=bool(random.getrandbits(1))) # menggambarkan pengambilan sampel dari distribusi Bernoulli, fungsi dipanggil dengan probabilitas p dan nilai k yang dihasilkan secara acak 0/1
  y.append(pick) # Nilai yang diambil (pick) ditambahkan ke dalam array y

u,s = np.mean(y), np.std(y) # perhitungan rata-rata (u) dan deviasi standar (s) dari array y
plt.scatter(x,y, label= r'$\mu=%.2f,\ \sigma=%.2f$' % (u,s)) # membuat scatter plot, label dari plot berisi rata-rata (u) dan deviasi standar (s)
plt.legend() # menampilkan legenda pada plot yang menunjukkan label yang telah ditentukan sebelumnya
plt.show() # menampilkan plot
```



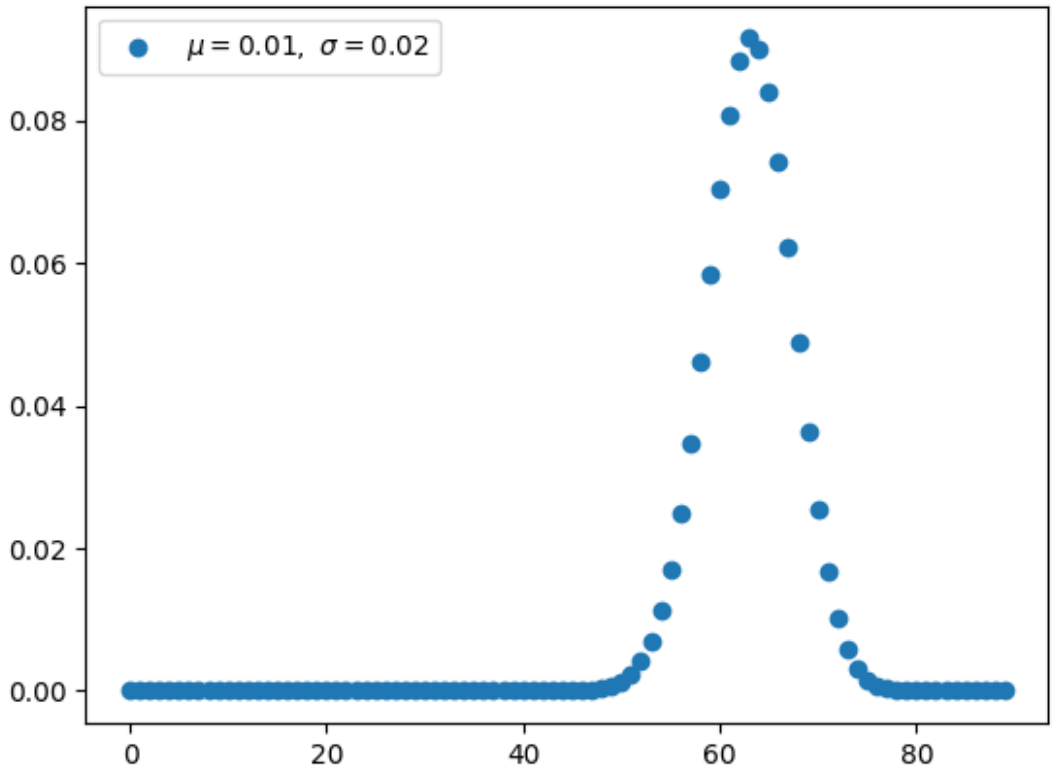
Flowchart


Mulai
Import Library
Membuat fungsi binomial
Assigment variabel n_experiment, p, x, y
range n_experiment
Yes
No
Perhitungan rata-rata dan standar deviasi
Membuat dan menampilkan hasil dengan scatter plot
Selesai

Latihan B1

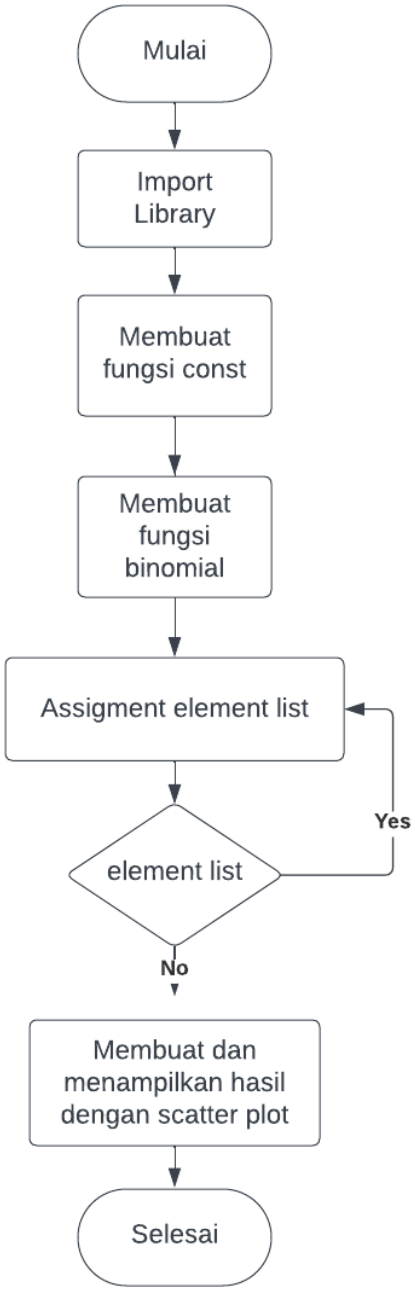
Jika matriks diganti dengan [(0.9, 10), (0.2, 70), (0.7, 90)] bagaimana grafiknya jelaskan menggunakan markdown dibawah grafik kumpulkan file ipnyb.

```
import operator as op
from functools import reduce

def const(n,r): # Membuat fungsi 'const'
  r = min(r, n-r) # Memilih nilai terkecil antara 'r' dan 'n-r'
  numer = reduce(op.mul, range(n, n-r, -1),1) # Menghitung numerator dengan mengalikan semua angka dari 'n' hingga 'n-r' dengan decrement -1
  denom = reduce(op.mul, range(1, r+1),1) # Menghitung denominasi dengan mengalikan semua angka dari 1 hingga 'r'
  return numer/denom  # Mengembalikan hasil dari pembagian numer/denom

def binomial(n,p): # Membuat fungsi 'binomial'
  q = 1-p # Menghitung q (probabilitas komplementer)
  y = [const(n,k)*(p**k)*(q**(n-k))for k in range(n)]  # Membuat list 'y' yang berisi nilai-nilai dari distribusi binomial
  return y, np.mean(y), np.std(y)  # Mengembalikan list 'y', nilai rata-rata ('mean'), dan deviasi standar ('std') dari 'y'

for ls in [(0.9,10),(0.2,70),(0.7,90)]: # Mengulangi proses berikut untuk setiap elemen dalam list
  p, n_experiment = ls[0], ls[1]  # Mengambil nilai 'p' dan 'n_experiment' dari elemen
  x = np.arange(n_experiment)
  y,u,s= binomial(n_experiment,p)

plt.scatter(x,y,label=r'$\mu=%.2f,\ \sigma=%.2f$' %(u,s)) # Memplot scatter plot
plt.legend()
plt.show() # Menampilkan plot
```



nilai pi dan sigma menjadi lebih kecil yang artinya sebaran data menjadi lebih padat

Flowchart

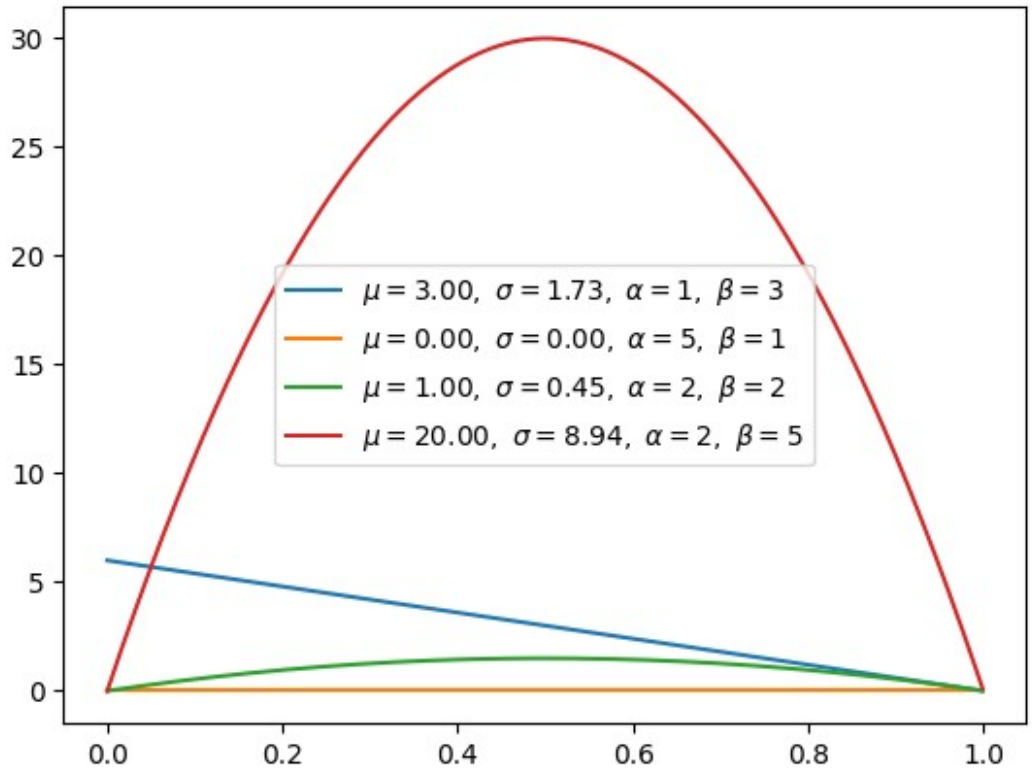

Mulai
Import Library
Membuat fungsi const
Membuat fungsi binomial
Assigment element list
element list
Yes
No
Membuat dan menampilkan hasil dengan scatter plot
Selesai

Latihan C1

jika nilai trial kita ubah menjadi 0,5 dan 0.1 apa yang terjadi ? jelaskan menggunakan markdown dalam file c1 dengan format dokumen ipnyb.

```
def gamma_function(n): # Membuat fungsi gamma
  cal=1
  for i in range(2,n):
    cal *=i
  return cal

def beta(x,a,b): # Membuat fungsi beta
  gamma = gamma_function(a+b)/ \
    (gamma_function(a)*gamma_function(b))
  y = gamma * (x**(a-1))* ((1-x)*(b-1))
  return x,y, np.mean(y), np.std(y) # Mengembalikan 'x', 'y', rata-
rata ('u'), dan deviasi standar ('s') dari 'y'

for ls in [(1,3),(5,1),(2,2),(2,5)]: # Mengiterasi melalui beberapa
pasangan nilai (a, b) yang diberikan dalam list
  a,b = ls[0], ls[1]

  x = np.arange(0,1,0.001, dtype=np.float)
  x,y,u,s = beta(x, a=a, b=b)
  plt.plot(x,y, label=r'$\mu=%.2f,\ \sigma=%.2f,'r'\ \alpha=%d,\ \
beta=%d$' % (u,s,a,b))
plt.legend()
plt.show()
```

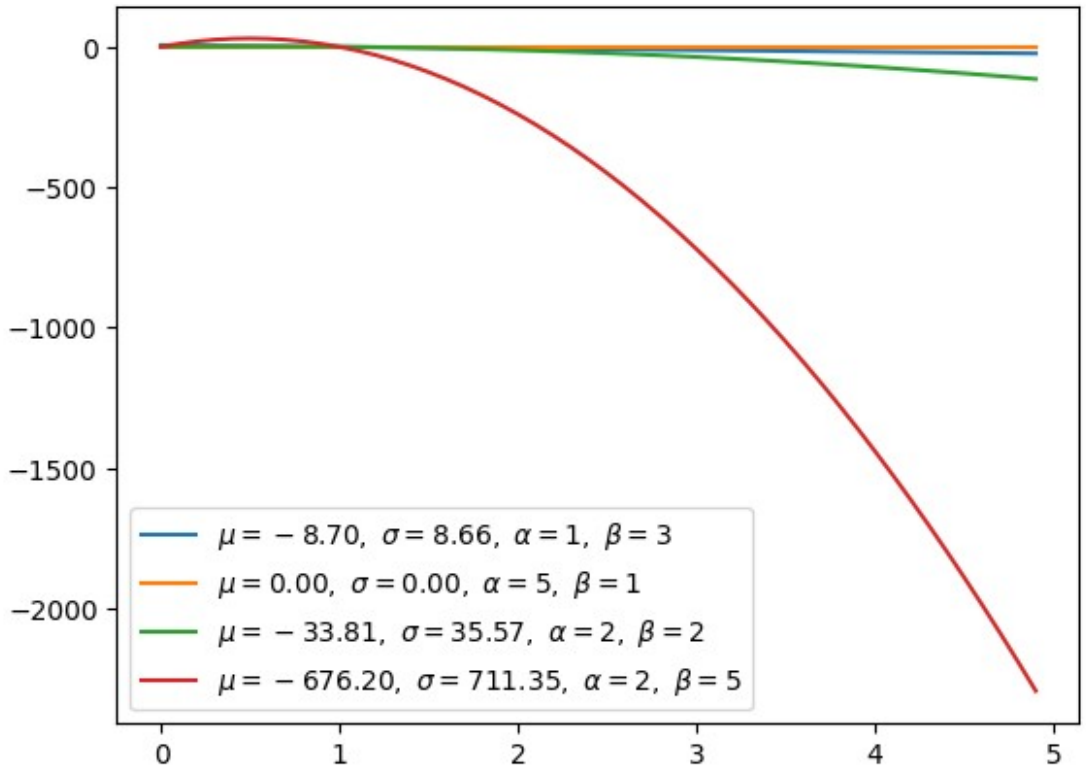
```
<ipython-input-16-17c02ac73592>:16: DeprecationWarning: `np.float` is
a deprecated alias for the builtin `float`. To silence this warning,
use `float` by itself. Doing this will not modify any behavior and is
safe. If you specifically wanted the numpy scalar type, use
`np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  x = np.arange(0,1,0.001, dtype=np.float)
```




```
def gamma_function(n): # Membuat fungsi gamma
  cal=1
  for i in range(2,n):
    cal *=i
  return cal

def beta(x,a,b): # Membuat fungsi beta
  gamma = gamma_function(a+b)/ \
    (gamma_function(a)*gamma_function(b))
  y = gamma * (x**(a-1))* ((1-x)*(b-1))
  return x,y, np.mean(y), np.std(y) # Mengembalikan 'x', 'y', rata-
rata ('u'), dan deviasi standar ('s') dari 'y'

for ls in [(1,3),(5,1),(2,2),(2,5)]: # Mengiterasi melalui beberapa
pasangan nilai (a, b) yang diberikan dalam list
  a,b = ls[0], ls[1]

  x = np.arange(0,5,0.1, dtype=np.float)
  x,y,u,s = beta(x, a=a, b=b)
  plt.plot(x,y, label=r'$\mu=%.2f,\ \sigma=%.2f,'r'\ \alpha=%d,\ \
beta=%d$' % (u,s,a,b))
plt.legend()
plt.show()
```

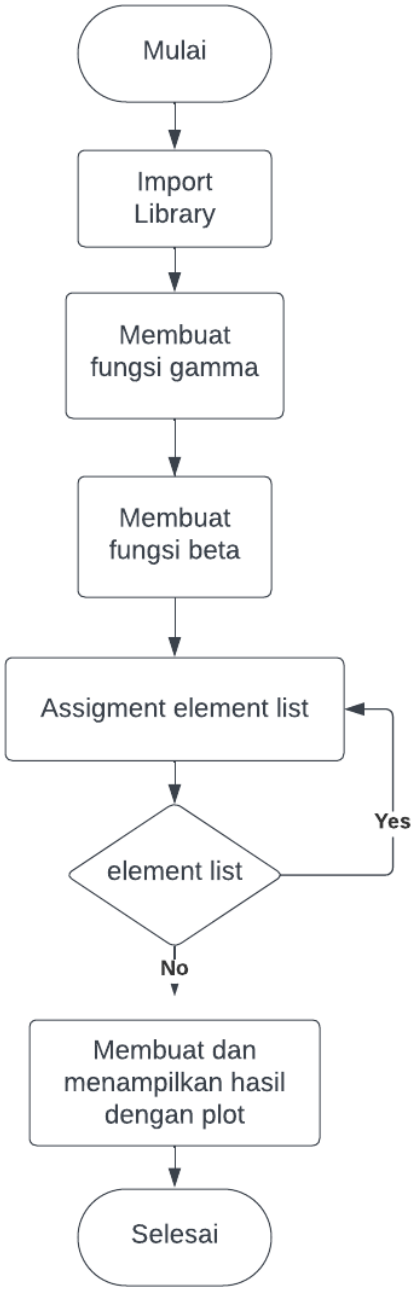
```
<ipython-input-4-a02fbcc85056>:16: DeprecationWarning: `np.float` is a
deprecated alias for the builtin `float`. To silence this warning, use
```

```
`float` by itself. Doing this will not modify any behavior and is
safe. If you specifically wanted the numpy scalar type, use
`np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  x = np.arange(0,5,0.1, dtype=np.float)
```



menghasilkan plot distribusi beta yang terlihat kurang menggambarkan data karena setiap data berjarak 0.1 dalam range 0 dan 5

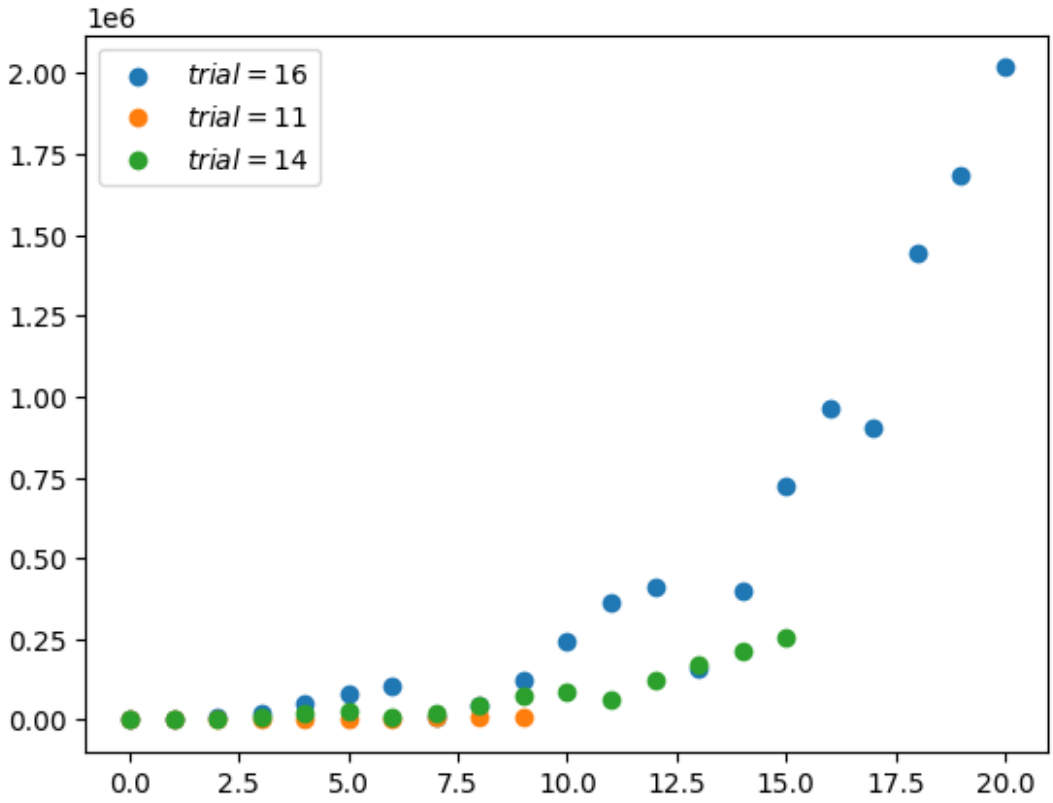
Flowchart


Mulai
Import Library
Membuat fungsi gamma
Membuat fungsi beta
Assigment element list
element list
Yes
No
Membuat dan menampilkan hasil dengan plot
Selesai

Latihan D1

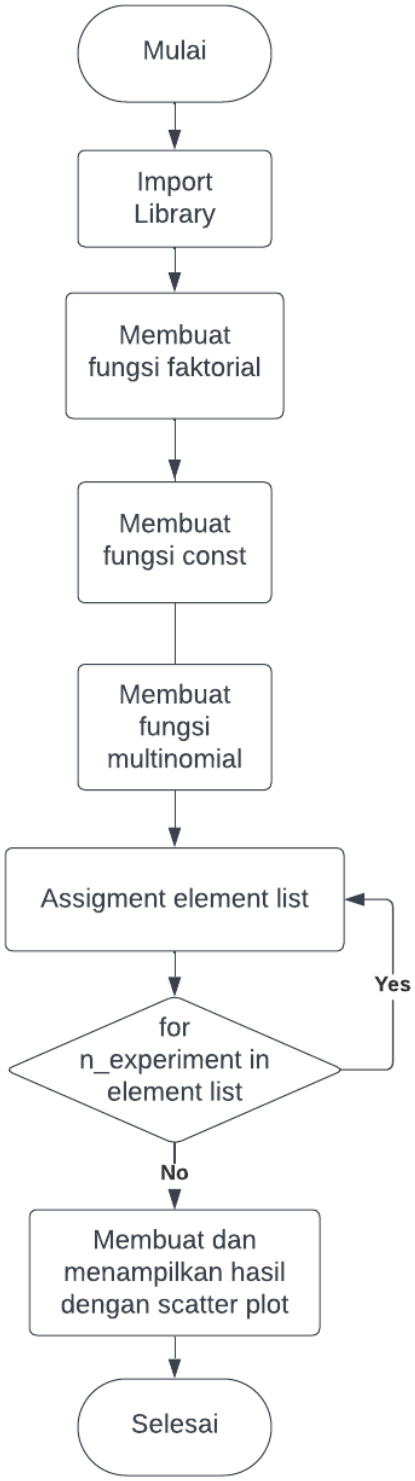
Jika array di ubah menjadi [16, 11, 14] maka apa yang terjadi ? jelaskan menggunakan markdown dalam file c1 dengan format dokumen ipnyb.

```
def factorial(n): # Membuat fungsi faktorial
  return reduce(op.mul, range(1, n+1),1)

def const(n,a,b,c): # Membuat fungsi 'const'
  assert a + b + c == n
  numer = factorial(n)
  denom = factorial(a)*factorial(b)*factorial(c)
  return numer/denom

def multinomial(n): # Membuat fungsi multinomial
  ls=[]
  for i in range(1,n+1):
    for j in range(i, n+1):
      for k in range(j,n+1):
        if i +j+k == n:
          ls.append([i,j,k])
  y = [const(n, l[0],l[1],l[2])for l in ls]  # Menghitung nilai konstanta untuk setiap kombinasi dalam ls dan menyimpannya dalam daftar y
  x = np.arange(len(y))  # Membuat array x yang berisi indeks dari setiap nilai dalam y
  return x,y, np.mean(y), np.std(y)  # Menghitung nilai rata-rata dan deviasi standar dari y

for n_experiment in[16,11,14]: # Melakukan eksperimen untuk nilai n 16, 11, dan 14
  x,y,u,s = multinomial(n_experiment) # Memanggil fungsi multinomial dengan n_experiment sebagai argumen
  plt.scatter(x,y, label=r'$trial=%d$' % (n_experiment))  # Membuat scatter plot

plt.legend()
plt.show()
```



semakin banyak experimen yang dilakukan nilai trial yang dihasilkan akan semakin besar

Flowchart

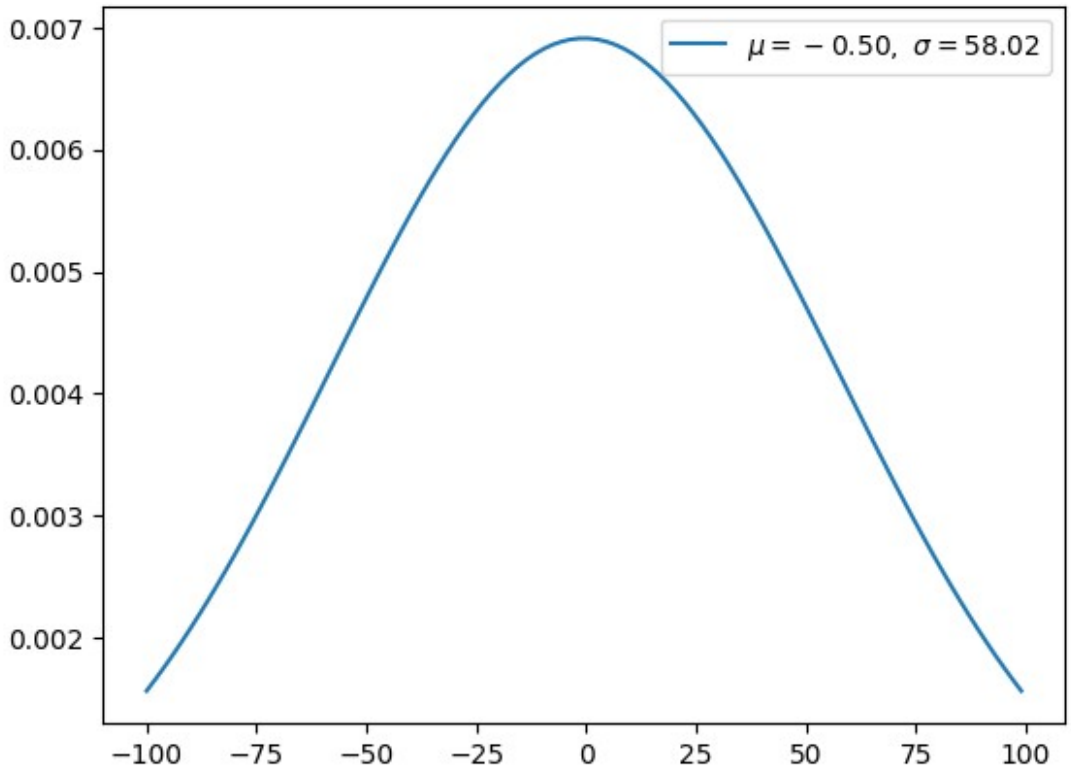

Mulai
Import Library
Membuat fungsi faktorial
Membuat fungsi const
Membuat fungsi multinomial
Assigment element list
for n_experiment in element list
Yes
No
Membuat dan menampilkan hasil dengan scatter plot
Selesai

Latihan E1

Buatlah jika nilai nilai gaussian bernilai 100 analisis hasilnya.

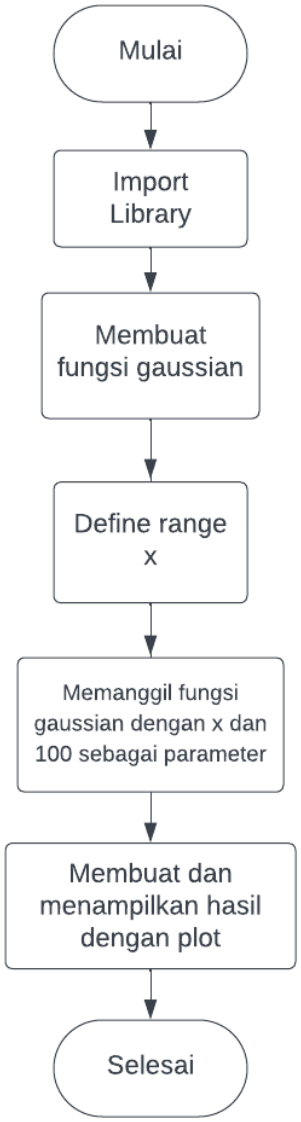
```
def gaussian (x,n):  # Membuat fungsi gaussian
  u = x.mean() # Menghitung nilai rata-rata (mean) dari x
  s = x.std() # Menghitung deviasi standar (standard deviation) dari x

  x = np.linspace(x.min(), x.max(), n)
  a = ((x-u)**2) / (2*(s**2)) # Menghitung komponen eksponensial dari distribusi Gaussian
  y = 1/(s*np.sqrt(2*np.pi))*np.exp(-a) # Menghitung nilai Gaussian sesuai dengan rumus distribusi Gaussian
  return x,y,x.mean(), x.std() # Mengembalikan x, y, mean (u), dan deviasi standar (s)

x = np.arange(-100,100)
x,y,u,s = gaussian(x,100) # Memanggil fungsi gaussian dengan x dan 100 sebagai parameter
plt.plot(x,y, label=r'$\mu=%.2f,\ \sigma=%.2f$' %(u,s)) # Membuat plot dari distribusi Gaussian
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7edf943cdc60>
```



jika nilai gaussian dikecilkan maka nilai sigma akan semakin besar

Flowchart



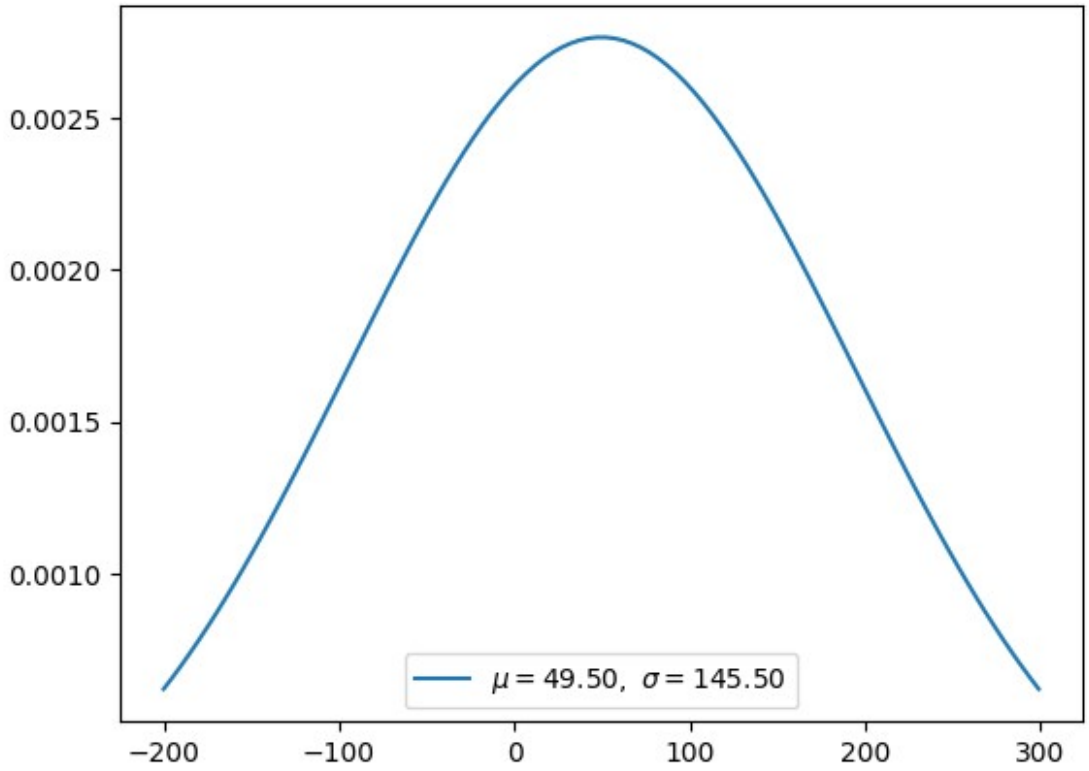
Latihan E2

Jika define range dari x diubah apa yang akan terjadi.

```
def gaussian (x,n):  # Membuat fungsi gaussian
  u = x.mean() # Menghitung nilai rata-rata (mean) dari x
  s = x.std() # Menghitung deviasi standar (standard deviation) dari x

  x = np.linspace(x.min(), x.max(), n)
```

```
  a = ((x-u)**2) / (2*(s**2)) # Menghitung komponen eksponensial dari distribusi Gaussian
  y = 1/(s*np.sqrt(2*np.pi))*np.exp(-a) # Menghitung nilai Gaussian sesuai dengan rumus distribusi Gaussian
  return x,y,x.mean(), x.std() # Mengembalikan x, y, mean (u), dan deviasi standar (s)

x = np.arange(-200,300)
x,y,u,s = gaussian(x,100) # Memanggil fungsi gaussian dengan x dan 100 sebagai parameter
plt.plot(x,y, label=r'$\mu=%.2f,\ \sigma=%.2f$' %(u,s)) # Membuat plot dari distribusi Gaussian
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7edf59d2a0b0>
```



Jika x range diubah akan mengubah nilai dari sigma

Tugas F1

Tampilkan keluaran dari semua model yang ada diatas.

```
#Contoh 1: Model Probit
import numpy as np
from scipy import stats
import statsmodels.api as sm
from statsmodels.base.model import GenericLikelihoodModel

data = sm.datasets.spector.load_pandas()
exog = data.exog
endog = data.endog
print(sm.datasets.spector.NOTE)
print(data.exog.head())

#Mereka, kami menambahkan konstanta ke matriks regresi
exog = sm.add_constant(exog, prepend=True)

#Untuk membuat Model Kemungkinan Anda sendiri, Anda hanya perlu
menimpa metode loglike.
class MyProbit(GenericLikelihoodModel):
    def loglike(self, params):
        exog = self.exog
        endog = self.endog
        q = 2 * endog - 1
        return stats.norm.logcdf(q*np.dot(exog, params)).sum()

#Perkirakan modelnya dan cetak ringkasannya:
sm_probit_manual = MyProbit(endog, exog).fit()
print(sm_probit_manual.summary())

#Compare your Probit implementation to statsmodel's "canned"
implementation:
sm_probit_canned = sm.Probit(endog, exog).fit()
print(sm_probit_canned.params)
print(sm_probit_manual.params)

print(sm_probit_canned.cov_params())
print(sm_probit_manual.cov_params())

::

    Number of Observations - 32

    Number of Variables - 4

    Variable name definitions::

        Grade - binary variable indicating whether or not a student's
```

```
grade
                improved.  1 indicates an improvement.
        TUCE  - Test score on economics test
        PSI   - participation in program
        GPA   - Student's grade point average

    GPA  TUCE  PSI
0  2.66  20.0  0.0
1  2.89  22.0  0.0
2  3.28  24.0  0.0
3  2.92  12.0  0.0
4  4.00  21.0  0.0
Optimization terminated successfully.
         Current function value: 0.400588
         Iterations: 292
         Function evaluations: 494
                              MyProbit Results

==============================================================================
========
Dep. Variable:                   GRADE   Log-Likelihood:
-12.819
Model:                         MyProbit   AIC:
33.64
Method:            Maximum Likelihood   BIC:
39.50
Date:                  Mon, 18 Sep 2023

Time:                          11:41:56

No. Observations:                   32

Df Residuals:                       28

Df Model:                            3

==============================================================================
========
                  coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
--------
const         -7.4523      2.542     -2.931      0.003     -12.435
-2.469
GPA            1.6258      0.694      2.343      0.019       0.266
2.986
TUCE           0.0517      0.084      0.617      0.537      -0.113
0.216
PSI            1.4263      0.595      2.397      0.017       0.260
2.593
```

```
==============================================================================
========
Optimization terminated successfully.
         Current function value: 0.400588
         Iterations 6
const   -7.452320
GPA      1.625810
TUCE     0.051729
PSI      1.426332
dtype: float64
[-7.45233176  1.62580888  0.05172971  1.42631954]
          const       GPA      TUCE       PSI
const  6.464166 -1.169668 -0.101173 -0.594792
GPA   -1.169668  0.481473 -0.018914  0.105439
TUCE  -0.101173 -0.018914  0.007038  0.002472
PSI   -0.594792  0.105439  0.002472  0.354070
[[ 6.46416776e+00 -1.16966614e+00 -1.01173187e-01 -5.94788999e-01]
 [-1.16966614e+00  4.81472101e-01 -1.89134577e-02  1.05438217e-01]
 [-1.01173187e-01 -1.89134577e-02  7.03758407e-03  2.47189354e-03]
 [-5.94788999e-01  1.05438217e-01  2.47189354e-03  3.54069513e-01]]
```

Latihan F2

Jelaskan alur dari code ipnyb diatas, apa saja yang dapat dianalisis dari hasil output diatas.

```
from scipy import stats
import statsmodels.api as sm
from statsmodels.base.model import GenericLikelihoodModel

data = sm.datasets.spector.load_pandas() # Memuat data dari dataset
Spector
exog = data.exog # (variabel bebas)
endog = data.endog # (variabel terikat)
print(sm.datasets.spector.NOTE) # Mencetak catatan tentang dataset
print(data.exog.head()) # Mencetak beberapa baris pertama dari
variabel exog

exog = sm.add_constant(exog, prepend=True) # Menambahkan konstanta ke
matriks regresi

## Membuat kelas MyProbit yang merupakan turunan dari
GenericLikelihoodModel, Anda hanya perlu menimpa metode loglike.
class MyProbit(GenericLikelihoodModel):
    def loglike(self, params):
        exog = self.exog
        endog = self.endog
```

```
        q = 2 * endog - 1
        return stats.norm.logcdf(q*np.dot(exog, params)).sum()

sm_probit_manual = MyProbit(endog, exog).fit() # Perkirakan model
Probit menggunakan metode yang telah dibuat
print(sm_probit_manual.summary())

# Bandingkan hasil implementasi Probit
sm_probit_canned = sm.Probit(endog, exog).fit()
print(sm_probit_canned.params) # Cetak parameter estimasi dari Probit
"canned"
print(sm_probit_manual.params) # Cetak parameter estimasi dari Probit
manual

print(sm_probit_canned.cov_params()) # Cetak matriks kovariansi dari
Probit "canned"
print(sm_probit_manual.cov_params()) # Cetak matriks kovariansi dari
Probit manual
```