```
#QUESTION_1


import statistics as statistics
from uncertainties import ufloat
from uncertainties.umath import *
from cmath import pi



Eddington = ufloat(1.61, 0.40)
Crommelin = ufloat(1.98, 0.16)
ProbabilityEddingtonvalue = Eddington/pi
ProbabilityCrommelinvalue = Crommelin/pi
Priorodds = ProbabilityEddingtonvalue/ProbabilityCrommelinvalue
print('The value of prior odds = ', Priorodds)

PH1givenD_Einstein = 1.74/pi
PH0givenD_Newton = 0.87/pi
Posteriorodds = PH0givenD_Newton/PH1givenD_Einstein
print( 'The value of posterior odds, ', Posteriorodds)

Bayesfactor = Posteriorodds/Priorodds
print('the value of Bayes factor is ', Bayesfactor)
```

OUTPUT

The value of prior odds = 0.81+/-0.21

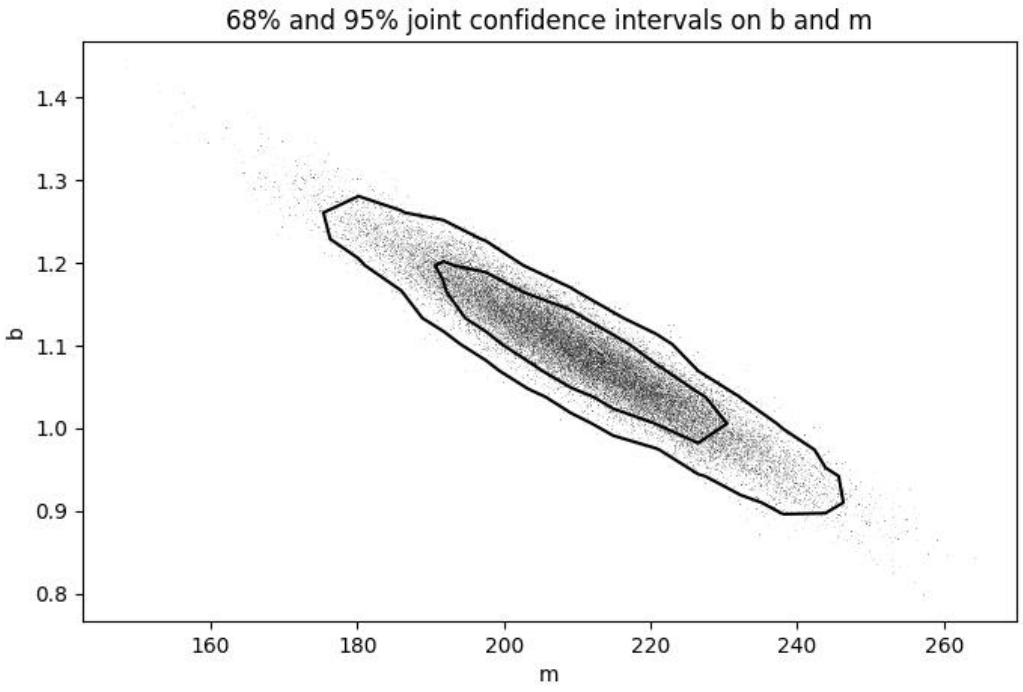
The value of posterior odds, 0.5

the value of Bayes factor is 0.61+/-0.16

```
#QUESTION_2


import emcee
import matplotlib.pyplot as plt
import numpy as np

data = np.loadtxt('C:\\Users\\Heera Baiju\\Desktop\\Q2data.txt')
xdata = data[:,1]
ydata = data[:,2]
error = data[:,3]

def compute_sigma_level(trace1, trace2, nbins = 20):
    L, xbins, ybins = np.histogram2d(trace1, trace2, nbins)
    L[L == 0] = 1E-16
    shape = L.shape
    L = L.ravel()

    i_sort = np.argsort(L)[::-1]
    i_unsort = np.argsort(i_sort)
    L_cumsum = L[i_sort].cumsum()
    L_cumsum /= L_cumsum[-1]
    xbins = 0.5 * (xbins[1:] + xbins[:-1])
    ybins = 0.5 * (ybins[1:] + ybins[:-1])
    return xbins, ybins, L_cumsum[i_unsort].reshape(shape)

def plot_MCMC_trace(ax, trace, scatter = False, **kwargs):

    xbins, ybins, sigma = compute_sigma_level(trace[0], trace[1])
    ax.contour(xbins, ybins, sigma.T, levels = [0.683, 0.955], **kwargs)
    if scatter:
        ax.plot(trace[0], trace[1], ',k', alpha = 0.1)
    ax.set_xlabel('m')
    ax.set_ylabel('b')

def plot_MCMC_results(trace, colors = 'k'):

    fig, ax = plt.subplots(1, 1, figsize = (8, 5))
    plt.title('68% and 95% joint confidence intervals on b and m')
    plot_MCMC_trace(ax, trace, True, colors = colors)

def log_prior(theta):
    beta = theta
    return -1.5 * np.log(1 + beta ** 2)

def log_likelihood(theta, x, y):
    alpha, beta = theta
    y_model = alpha + beta * x
```

```
    return -0.5 * np.sum(np.log(2 * np.pi * error ** 2) + (y - y_model) ** 2 /
error **
2)


def log_posterior(theta, x, y):
    return log_prior(theta) + log_likelihood(theta, x, y)

ndim = 2
nwalkers = 50
nburn = 1000
nsteps = 2000

np.random.seed(0)
starting_guesses = np.random.random((nwalkers, ndim))
sampler = emcee.EnsembleSampler(nwalkers, ndim, log_posterior, args = [xdata,
ydata])
sampler.run_mcmc(starting_guesses, nsteps)
emcee_trace = sampler.chain[:, nburn:, :].reshape(-1, ndim).T
plot_MCMC_results(emcee_trace)
plt.show()
```

OUTPUT

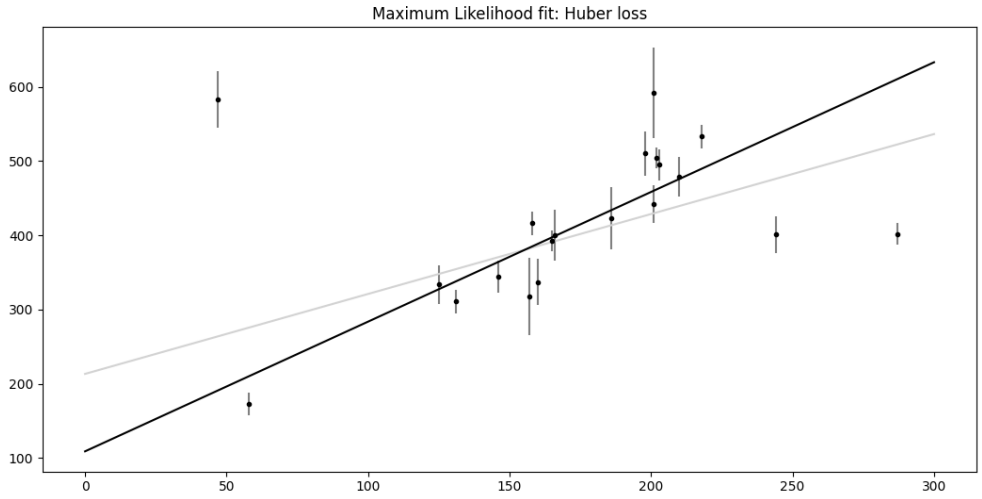


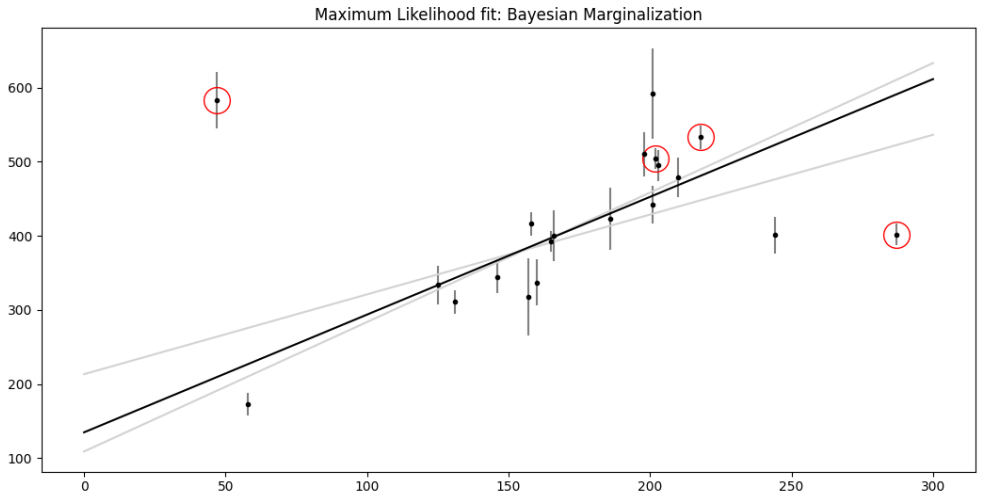
#QUESTION_3

```
import emcee
import numpy as np
from scipy import optimize
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv("C:\\Users\\Heera Baiju\\Desktop\\Q3data.csv")
x = data['x'].values
y = data['y'].values
e = data['sigma_y'].values
xfit = np.linspace(0, 300, 1000)
t = np.linspace(-20, 20)

def squared_loss(theta, x = x, y = y, e = e):
    dy = y - theta[0] - theta[1] * x
    return np.sum(0.5 * (dy / e) ** 2)

theta1 = optimize.fmin(squared_loss, [0, 0], disp = False)

def huber_loss(t, c = 3):
    return ((abs(t) < c) * 0.5 * t ** 2 + (abs(t) >= c) * -c * (0.5 * c -
abs(t)))

def total_huber_loss(theta, x = x, y = y, e = e, c = 3):
    return huber_loss((y - theta[0] - theta[1] * x) / e, c).sum()
theta2 = optimize.fmin(total_huber_loss, [0, 0], disp = False)
plt.errorbar(x, y, e, fmt = '.k', ecolor = 'gray')
plt.plot(xfit, theta1[0] + theta1[1] * xfit, color = 'lightgray')
plt.plot(xfit, theta2[0] + theta2[1] * xfit, color = 'black')
plt.title('Maximum Likelihood fit: Huber loss')
plt.show()

def log_prior(theta):

    if (all(theta[2:] > 0) and all(theta[2:] < 1)):
        return 0
    else:
        return -np.inf

def log_likelihood(theta, x, y, e, sigma_B):
    dy = y - theta[0] - theta[1] * x
    g = np.clip(theta[2:], 0, 1)
    logL1 = np.log(g) - 0.5 * np.log(2 * np.pi * e ** 2) - 0.5 * (dy / e) ** 2
    logL2 = np.log(1 - g) - 0.5 * np.log(2 * np.pi * sigma_B ** 2) - 0.5 * (dy
/
```

```
sigma_B) ** 2
    return np.sum(np.logaddexp(logL1, logL2))

def log_posterior(theta, x, y, e, sigma_B):
    return log_prior(theta) + log_likelihood(theta, x, y, e, sigma_B)
ndim = 2 + len(x)
nwalkers = 50
nburn = 10000
nsteps = 15000
np.random.seed(4)
starting_guesses = np.zeros((nwalkers, ndim))
starting_guesses[:, :2] = np.random.normal(theta2, 1, (nwalkers, 2))
starting_guesses[:, 2:] = np.random.normal(0.5, 0.1, (nwalkers, ndim - 2))
sampler = emcee.EnsembleSampler(nwalkers, ndim, log_posterior, args=[x, y, e,
50])
sampler.run_mcmc(starting_guesses, nsteps)
sample = sampler.chain # shape = (nwalkers, nsteps, ndim)
sample = sampler.chain[:, nburn:, :].reshape(-1, ndim)
theta3 = np.mean(sample[:, :2], 0)
g = np.mean(sample[:, 2:], 0)
outliers = (g < 0.38)
# Plotting
plt.errorbar(x, y, e, fmt = '.k', ecolor = 'gray')
plt.plot(xfit, theta1[0] + theta1[1] * xfit, color = 'lightgray')
plt.plot(xfit, theta2[0] + theta2[1] * xfit, color = 'lightgray')
plt.plot(xfit, theta3[0] + theta3[1] * xfit, color = 'black')
plt.plot(x[outliers], y[outliers], 'ro', ms=20, mfc = 'none', mec='red')
plt.title('Maximum Likelihood fit: Bayesian Marginalization')
plt.show()
```

OUTPUT




Maximum Likelihood fit: Bayesian Marginalization
600
500
400
300
200
100
0
50
100
150
200
250
300