ASSIGNMENT 6 DATA SCIENCE & ANALYSIS

NAME: - ASHWIN KUMAR

ROLL NO: - CE21BTECH11008

Q1.--

```
import numpy as np
from scipy.stats import norm

Eddington_report=1.61
Eddington_error=0.40

Cromellelin_report=1.98
Cromellelin_error=0.16

theta1=1.74  #Einstein predicted
#Eddington one
G1=norm.pdf(Eddington_report, loc=theta1, scale=Eddington_error)

#Cromellelin
G2=norm.pdf(Eddington_report, loc=theta1, scale=Cromellelin_error)
Combine_likelihood_Einstein=G1*G2  #as both the experiment done
independently so we can directly multyply

#now liklihood Einstein one
theta2=0.87 #Newton predicted -- half than what Einstein predicted
#Eddington one
G3=norm.pdf(Eddington_report, loc=theta2, scale=Eddington_error)

#Cromellelin
G4=norm.pdf(Eddington_report, loc=theta2, scale=0.16)

Combine_likelihood_Newton=G3*G4 #as both the experiment done independently
so we can directly multyply

print(G1)
print(G2)
print(G3)
print(G4)

#bayes factor = ratio of liklihood of the model
bayes_factor= Combine_likelihood_Einstein/Combine_likelihood_Newton

print(bayes_factor)
```

G1= 0.9460495798345487

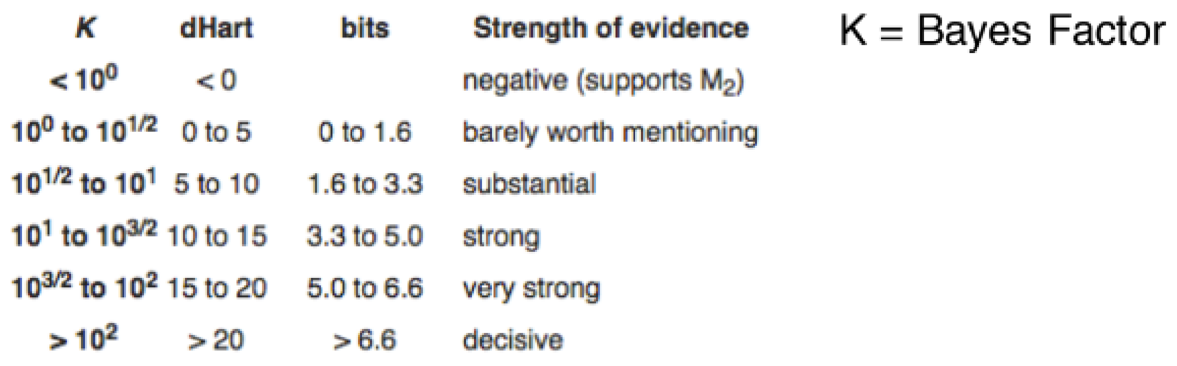
G2=1.7924166722900914

G3=0.180162185840545

G4=5.6477424306570885e-05

K(BAYES FACTOR) =166653.46

So bayes factor is coming far greater than 100 so strength of evidence is decisive and Einstein observation is most likely to be true than Nweton’s one.



K	dHart	bits	Strength of evidence
$< 10^0$	< 0		negative (supports M_2)
10^0 to $10^{1/2}$	0 to 5	0 to 1.6	barely worth mentioning
$10^{1/2}$ to 10^1	5 to 10	1.6 to 3.3	substantial
10^1 to $10^{3/2}$	10 to 15	3.3 to 5.0	strong
$10^{3/2}$ to 10^2	15 to 20	5.0 to 6.6	very strong
$> 10^2$	> 20	> 6.6	decisive

K = Bayes Factor

Q2--

INSTALLING THE NECESSARY LIBRARY

```
pip install emcee
pip install astroML
```

CODE:-

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

# Define the model function
def model(params, x):
    m, b = params
    return m * x + b
```

```python
# Define the log likelihood function
def log_likelihood(params, x, y, yerr):
    m, b = params
    model_vals = model(params, x)
    return -0.5 * np.sum((y - model_vals) ** 2 / yerr ** 2)

# Define the log prior function
def log_prior(params):
    m, b = params
    if -10.0 < m < 10.0 and -1000.0 < b < 1000.0:
        return 0.0
    return -np.inf

# Define the log probability function
def log_probability(params, x, y, yerr):
    lp = log_prior(params)
    if not np.isfinite(lp):
        return -np.inf
    return lp + log_likelihood(params, x, y, yerr)

# Load the data
data = np.genfromtxt("/content/data2_q2.txt", names=True)

x = data['x']
y = data['y']
yerr = data['σy']

# Set up the initial parameters
initial_params = np.random.rand(2)

# Set up the emcee sampler
nwalkers = 32
ndim = len(initial_params)
sampler = emcee.EnsembleSampler(nwalkers, ndim, log_probability, args=(x,
y, yerr))

# Run the sampler
nsteps = 8000
sampler.run_mcmc(initial_params + 1e-4 * np.random.randn(nwalkers, ndim),
nsteps, progress=True)

# Get the chains
samples = sampler.chain[:, 1000:, :].reshape((-1, ndim))
```

```
# Calculate the confidence intervals
m_median, b_median = np.median(samples, axis=0)
m_err_minus, m_err_plus = np.percentile(samples[:, 0], [16, 84])
b_err_minus, b_err_plus = np.percentile(samples[:, 1], [16, 84])

print("68% Confidence Intervals:")
print(f"m = {m_median:.2f} + {m_err_plus - m_median:.2f} - {m_median -
m_err_minus:.2f}")
print(f"b = {b_median:.2f} + {b_err_plus - b_median:.2f} - {b_median -
b_err_minus:.2f}")
print("95% Confidence Intervals:")
# Calculate the confidence intervals

m_median, b_median = np.median(samples, axis=0)
m_err_minus, m_err_plus = np.percentile(samples[:, 0], [2.5, 97.5])
b_err_minus, b_err_plus = np.percentile(samples[:, 1], [2.5, 97.5])

print(f"m = {m_median:.2f} + {m_err_plus - m_median:.2f} - {m_median -
m_err_minus:.2f}")
print(f"b = {b_median:.2f} + {b_err_plus - b_median:.2f} - {b_median -
b_err_minus:.2f}")

mu_true, sigma_true = 1000, 15  # stochastic flux model

from astroML.plotting import plot_mcmc
fig = plt.figure()
ax = plot_mcmc(samples.T, fig=fig, labels=[r'$m$', r'$b$'], colors='k')
ax[0].plot(samples[:, 0], samples[:, 1], ',k', alpha=0.1)
ax[0].plot([mu_true], [sigma_true], 'o', color='red', ms=10);
```

Output: - Inner one (ellipse) is 68% confidence interval while outer one is 95%confidence interval.

68% Confidence Intervals:

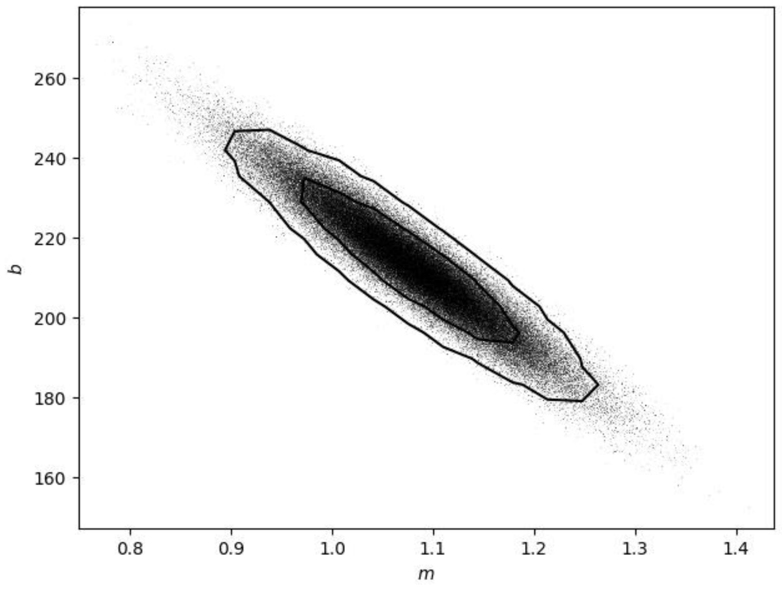
m = 1.08 (+ 0.07 - 0.08) error

b = 212.86(+ 14.40 - 13.94)error

95% Confidence Intervals:

m = 1.08 (+ 0.15 - 0.15) error

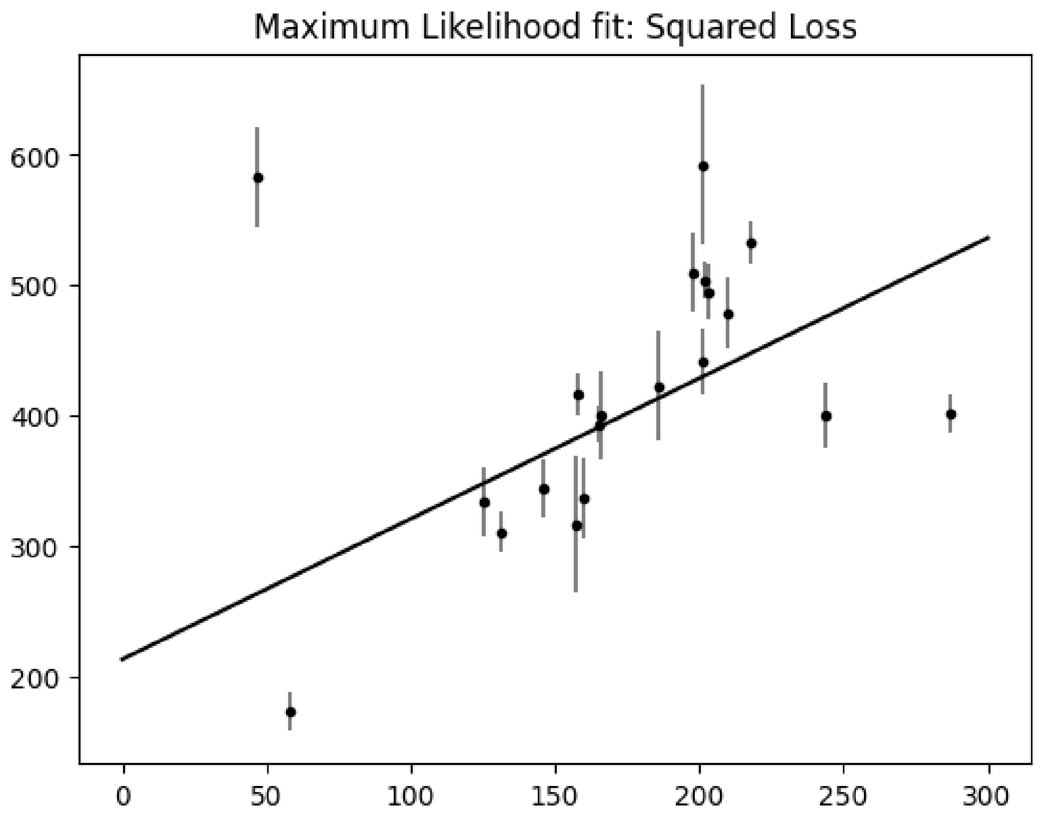
b = 212.86(+ 28.29 - 27.67)error



Q3: --

Maximum likelihood basis

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

# Assuming your data1 has the name data2_q2.txt with address
/content/data2_q2.txt
data1 = pd.read_csv('/content/data2_q2.txt',delimiter=' ') #space to
denote the separation so delimiter = ' '
#preprocessing the data
#removing the σx ρxy from the data
print(data1)
columns_to_drop = ['σx' , 'ρxy']
data1 = data1.drop(columns=columns_to_drop)
# print(data1) #data without removing first four rows

#changing its index to 1 again for better processing in the future
data1.reset_index(drop=True, inplace=True)
# print(data1) #data with removing first four rows
#Extracting x, y, and σy values from the data---
x=data1['x']
y=data1['y']
σy=data1['σy']
print(data1)

from scipy import optimize

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

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

xfit = np.linspace(0, 300)
plt.errorbar(x, y, σy , fmt='.k', ecolor='gray')
plt.plot(xfit, theta1[0] + theta1[1] * xfit, '-k')
plt.title('Maximum Likelihood fit: Squared Loss');
```

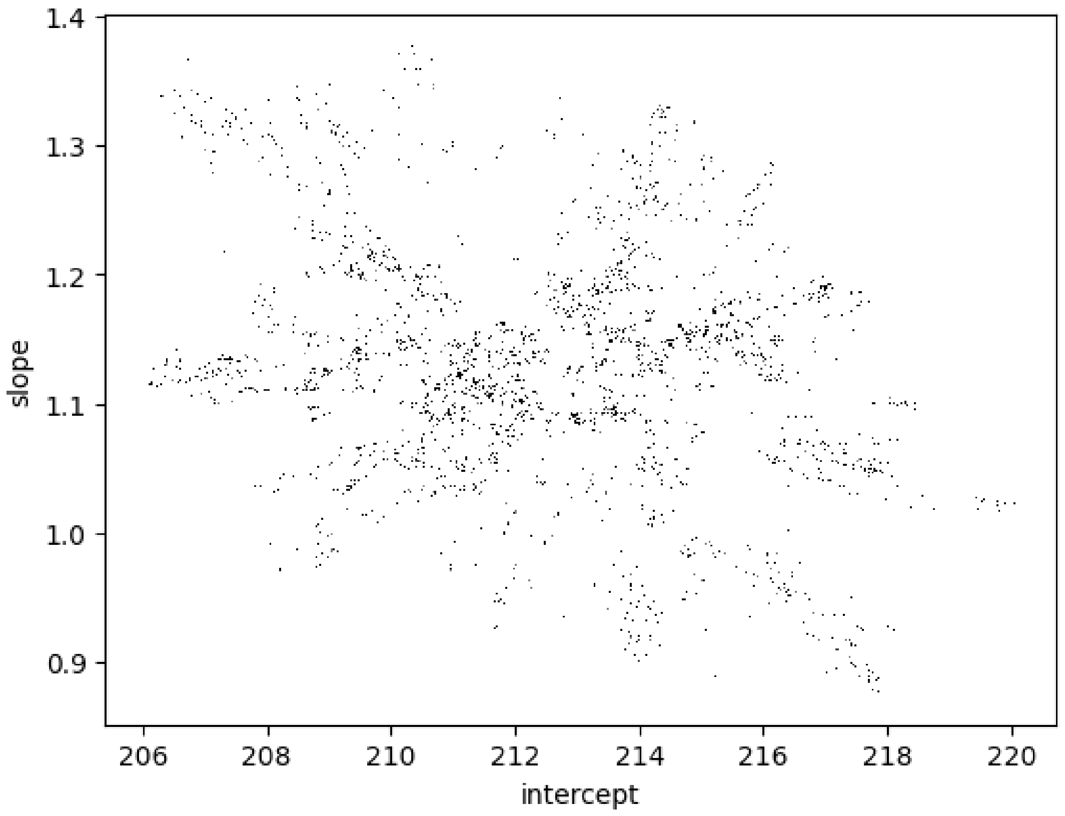


Bayesian Approach to Outliers

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

# Assuming your data1 has the name data2_q2.txt with address
/content/data2_q2.txt
data1 = pd.read_csv('/content/data2_q2.txt', delimiter=' ')  # space to
denote the separation so delimiter = ' '
# preprocessing the data
# removing the σx ρxy from the data
columns_to_drop = ['σx', 'ρxy']
data1 = data1.drop(columns=columns_to_drop)

# Extracting x, y, and σy values from the data---
```

```
x = data1['x']
y = data1['y']
e = data1['σy']

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


theta1 = minimize(squared_loss, [0, 0], args=(x, y, e)).x


def log_prior(theta):
    # g_i needs to be between 0 and 1
    if np.all((theta[2:] > 0) & (theta[2:] < 1)):
        return 0
    else:
        return -np.inf  # recall log(0) = -inf


def log_likelihood(theta, x, y, e, sigma_B):
    dy = y - theta[0] - theta[1] * x
    # g = np.clip(theta[2:], 0, 1)  # g<0 or g>1 leads to NaNs in
logarithm
    g = np.clip(theta[2:], 1e-10, 1-1e-10)  # Clip values to avoid NaNs in
logarithm
    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)  # number of parameters in the model
nwalkers = 50  # number of MCMC walkers
nburn = 1000  # "burn-in" period to let chains stabilize
nsteps = 1500  # number of MCMC steps to take

# set theta near the maximum likelihood, with
np.random.seed(0)
starting_guesses = np.zeros((nwalkers, ndim))
```

```python
starting_guesses[:, :2] = np.random.normal(theta1, 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)

plt.plot(sample[:, 0], sample[:, 1], ',k', alpha=0.5)
plt.xlabel('intercept')
plt.ylabel('slope')
plt.show()
```



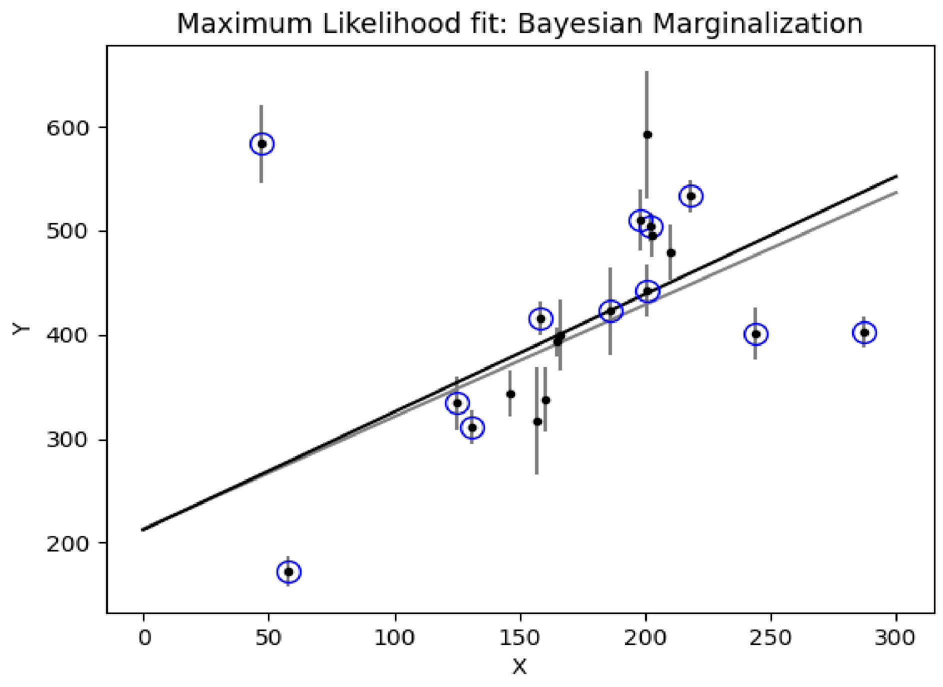
Intercept vs slope clustering. Intercept is coming around 214 and with slope of 1.1.

Comparison between both(Maximum Likelihood & Bayesian Approach to Outliers

CODE:-

```
theta3 = np.mean(sample[:, :2], 0)
g = np.mean(sample[:, 2:], 0)
outliers = (g < 0.5)
xfit = np.linspace(0, 300)

plt.errorbar(x, y, e, fmt='.k', ecolor='gray')
plt.plot(xfit, theta1[0] + theta1[1] * xfit, color='gray')
# 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=10, mfc='none', mec='blue')
plt.title('Maximum Likelihood fit: Bayesian Marginalization');
plt.xlabel("X")
plt.ylabel("Y")
```

Output:-



Blue circle denotes the outliers. Black one represents Bayesian fit while grey one Maximum likelihood fit. The points in blue circle denote the outlier.