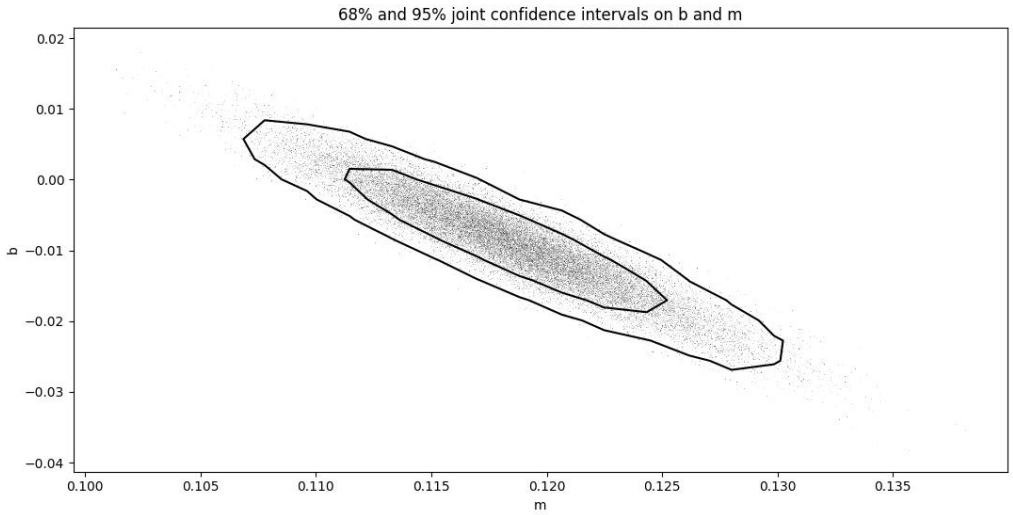
```
#QUESTION-1

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

gasdata = pd.read_csv("C:\\Users\\Heera Baiju\\Downloads\\Data
Science\\Assignments\\Assignment 7\\SPT.csv")
xd = gasdata['#z'].values
yd = gasdata['fgas'].values
e = gasdata['fgas_error'].values

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 * e ** 2) + (y - y_model) ** 2 / e
**
2)

def log_post(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_post, args = [xd,
yd])
sampler.run_mcmc(starting_guesses, nsteps)
emcee_trace = sampler.chain[:, nburn:, :].reshape(-1, ndim).T
plot_MCMC_results(emcee_trace)
plt.show()
```

RESULTS



#QUESTION-2

```
import numpy as np
import nestle
from scipy import stats
global data, x, y, sigma_y

data = np.array([[ 0.417022004703, 0.720324493442, 0.000114374817345,
0.302332572632,
0.146755890817, 0.0923385947688, 0.186260211378,
0.345560727043,
0.396767474231, 0.538816734003, 0.419194514403,
0.685219500397,
0.204452249732, 0.878117436391, 0.0273875931979,
0.670467510178,
0.417304802367, 0.558689828446, 0.140386938595, 0.198101489085
],
[ 0.121328306045, 0.849527236006, -1.01701405804, -
0.391715712054,
-0.680729552205, -0.748514873007, -0.702848628623, -
0.0749939588554,
0.041118449128, 0.418206374739, 0.104198664639, 0.7715919786,
-0.561583800669, 1.43374816145, -0.971263541306,
0.843497249235,
-0.0604131723596, 0.389838628615, -0.768234900293, -
0.649073386002 ],
[ 0.1 , 0.1 , 0.1 , 0.1 , 0.1 ,
0.1 , 0.1 , 0.1 , 0.1 , 0.1 ,
0.1 , 0.1 , 0.1 , 0.1 , 0.1 ,
0.1 , 0.1 , 0.1 , 0.1 , 0.1 ]])
x, y, sigma_y = data

def polynomial_fit(theta, x):

    return sum(t * x ** n for (n, t) in enumerate(theta))

def logL(theta):

    y_fit = polynomial_fit(theta, x)
    return sum(stats.norm.logpdf(*args) for args in zip(y, y_fit, sigma_y))

def prior_transform(x):
    return 10.0 * x - 5.0

Lin = nestle.sample(logL, prior_transform, 2)
Quad = nestle.sample(logL, prior_transform, 2)

print(" Linear model's Bayesian evidence : ", Lin.logz)
```

```
print(" Quadratic model's Bayesian evidence : ", Quad.logz)
```

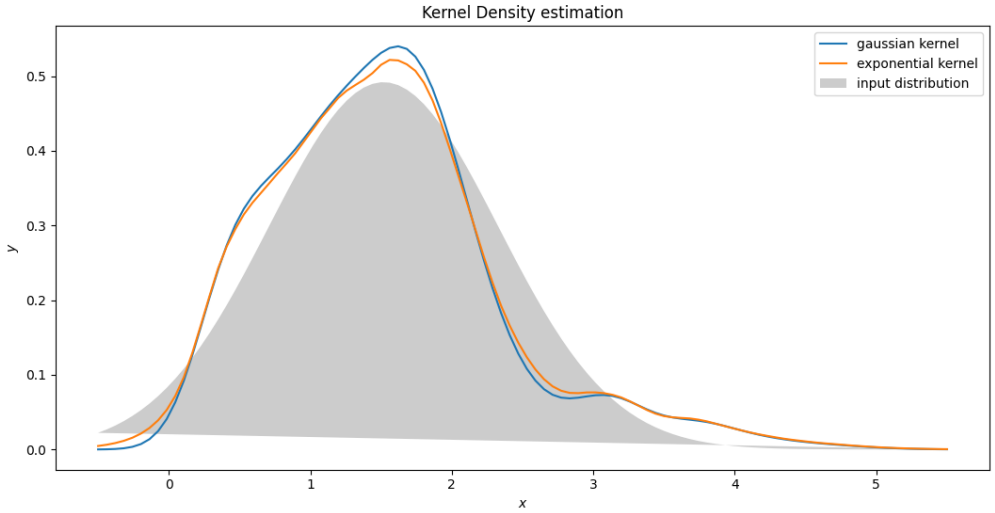
RESULTS

Linear model's Bayesian evidence : 13.065918330361148

Quadratic model's Bayesian evidence : 13.144615933238683

```
#QUESTION-3

import numpy as np
import csv
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import norm
from sklearn.neighbors import KernelDensity

datContent = [i.strip().split() for i in open("C:\\Users\\Heera
Baiju\\Downloads\\Data Science\\Assignments\\Assignment
7\\SDSS_quasar.dat").readlines()]

with open("./SDSS_quasar.csv", "w") as f:
     writer = csv.writer(f)
     writer.writerows(datContent)

d = pd.read_csv('SDSS_quasar.csv', usecols=['z'])
data = d.values
t = np.linspace(-0.5, 5.5, 100)

kde1 = KernelDensity(kernel='gaussian', bandwidth=0.2).fit(data)
kde1 = kde1.score_samples(t.reshape(-1,1))
kde2 = KernelDensity(kernel='exponential', bandwidth=0.2).fit(data)
kde2 = kde2.score_samples(t.reshape(-1,1))
dist = norm(np.mean(data), np.std(data)).pdf(t.reshape(-1,1))
plt.plot(t, np.exp(kde1), label='gaussian kernel')
plt.plot(t, np.exp(kde2), label='exponential kernel')
plt.fill(t.reshape(-1,1), dist, fc='black', alpha=0.2,label='input
distribution')
plt.title('Kernel Density estimation')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.legend()
plt.show()
```

RESULTS


Kernel Density estimation
gaussian kernel
exponential kernel
input distribution
0.5
0.4
0.3
0.2
0.1
0.0
y
0
1
2
3
4
5
x