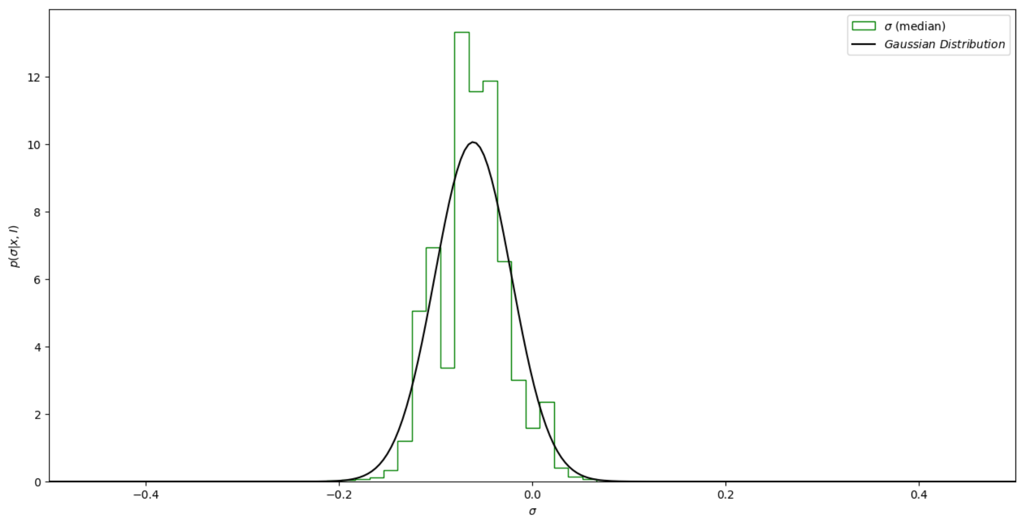
Assignment-3

Q1

```
#-------------------------------------------------------------
# Importing all libraries
import numpy as np
from scipy.stats import norm
from matplotlib import pyplot as plt
from astroML.resample import bootstrap
from astroML.stats import median_sigmaG

m = 1000  # Number of points
n = 10000  # Number of bootstraps

#-------------------------------------------------------------
# Sample values from a normal distribution
np.random.seed(0)
data = norm(0, 1).rvs(m)

#-------------------------------------------------------------
# Compute bootstrap resamplings of data
median, sigmaG = bootstrap(data, n, median_sigmaG, kwargs=dict(axis=1))

#-------------------------------------------------------------
# Compute the theoretical expectations for the two distributions
x = np.linspace(-2, 2, 1000)

sigma = np.sqrt(np.pi/(2*m))
mu = np.mean(median)
pdf = norm(mu, sigma).pdf(x)

#-------------------------------------------------------------
# Plot the results
fig, ax = plt.subplots(figsize=(5, 3.75))

ax.hist(median, bins = 20, density = True, histtype = 'step', color = 'green',
label = r'$\sigma\ {\rm (median)}$')
ax.plot(x, pdf, color = 'black', label = '$Gaussian\ Distribution$')

ax.set_xlim(-0.5, 0.5)
```

```
ax.set_xlabel(r'$\sigma$')
ax.set_ylabel(r'$p(\sigma|x,I)$')

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

Output



Q2

```
#--------------------------------------------------------------
# Importing all libraries
import numpy as np
import pandas as pd
from scipy.optimize import curve_fit
from matplotlib import pyplot as plt

#--------------------------------------------------------------
# Importing dataset
data = pd.read_excel("E:\Data Science\Assignment-3\Data.xlsx")
x_values =  data ['x']
y_values =  data ['y']
sigmaY = data ['sigmaY']

#--------------------------------------------------------------
# Objective function
def function(x, a, b):
    return a * x + b

#--------------------------------------------------------------
# Getting the optimal values for the parameters and the estimated covariance of
parameters
val = np.array([0, 0]) # Assuming the values as (0,0)

param, param_cov = curve_fit(function, x_values, y_values, val, sigmaY)

print("Line function coefficients: {}" .format(param) )
print("Covariance of coefficients: {}" .format(param_cov) )

#--------------------------------------------------------------
# Getting the y-intercept and slope given by curve-fit() function
t = np.linspace(0, 300, 1000)
c = param[1] # The y-intercept given by curve-fit() function
m = param[0] # The slope given by curve-fit() function
print("The value of y-intercept is {}" .format(c) )
print("The value of slope is {}" .format(m) )

#--------------------------------------------------------------
# Plot the results
plt.errorbar(x_values, y_values, sigmaY , fmt='.k', ecolor='gray', label='Plot of
y Vs x')
```

```
plt.plot(t, m*t+c, '--', color ='blue', label ="Optimized data")

plt.xlim(0, 300, 50)
plt.ylim(0, 700, 100)
plt.xlabel('x')
plt.ylabel('y')

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

Output

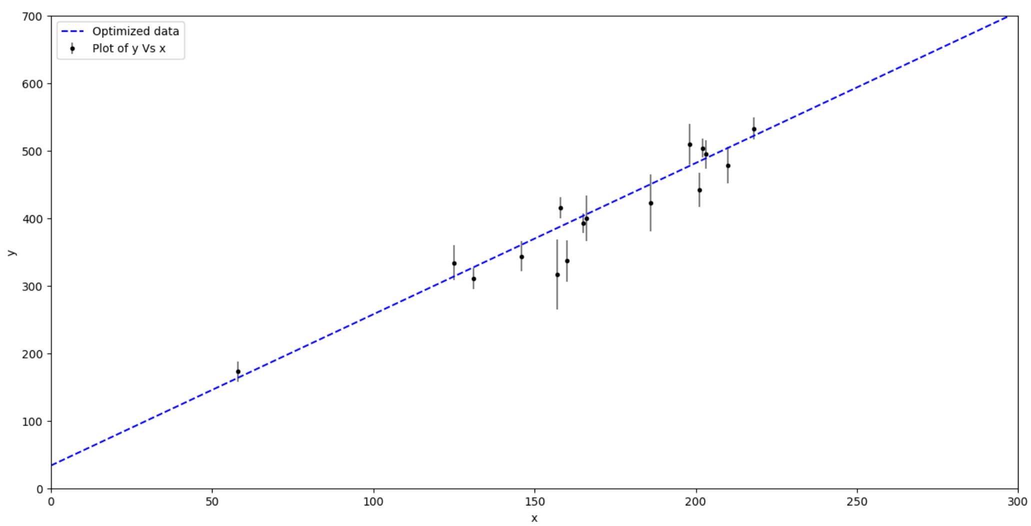
Line function coefficients: [2.2399208 34.04773403]

Covariance of coefficients: [[1.55005444e-02 -2.52129683e+00]

 [-2.52129683e+00 4.44232207e+02]]

The value of y-intercept is 34.04773403259783

The value of slope is 2.2399207961186938



Q3

```
#------------------------------------------------------------
# Importing all libraries
import numpy as np
from scipy import stats

#------------------------------------------------------------
# Generate dataset
np.random.seed(1)

N = 50
L0 = 10
dL = 0.2

t = np.linspace(0, 1, N)
L_obs = np.random.normal(L0, dL, N)

y_vals = [L_obs, L_obs, L_obs, L_obs + 0.5 - t ** 2]
y_errs = [dL, dL * 2, dL / 2, dL]
titles = ['correct errors', 'overestimated errors', 'underestimated errors',
'incorrect model']

for i in range(4):
    #------------------------------------------------------------
    # Compute the mean and the chisquare/dof
    mu = np.mean(y_vals[i])
    z = (y_vals[i] - mu) / y_errs[i]
    chi2 = np.sum(z ** 2)
    chi2dof = chi2 / (N - 1)

    #------------------------------------------------------------
    # Calculating p value
    p_value = stats.chi2(N-1).sf(chi2dof*chi2)
    print("p-value for the {} is: {}" .format(titles[i], p_value))
```

Output

p-value for the correct errors is: 0.6323042459089494

p-value for the overestimated errors is: 1.0

p-value for the underestimated errors is: 2.2834522158057202e-120

p-value for the incorrect model is: 4.7450723075838625e-56