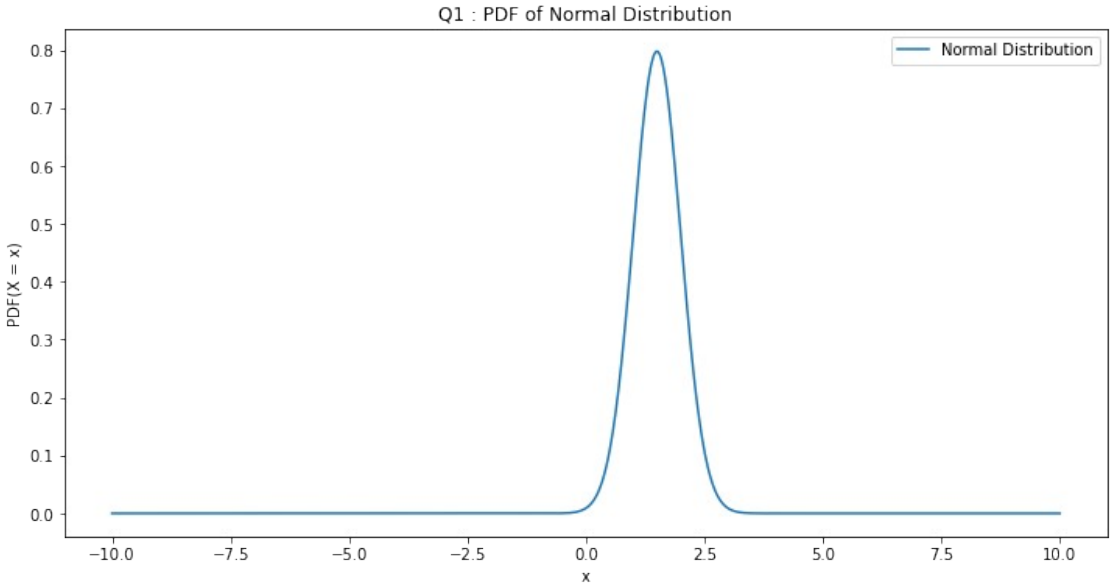
Assignment 1

EE19BTECH11050 : N Tarun

```
# Intended ONLY for Google Colab
!pip install astroML --quiet

import scipy.stats
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.figure import figaspect
import astroML.stats
```

Q1

```
# Generate normal distrubtion with mean=1.5, variance=0.5
dist = scipy.stats.norm(1.5, 0.5)

# Draw 1000 times from that distribution
draws = dist.rvs(1000)

# Adjust plot size
wid, hei = figaspect(1.)
fig, axs = plt.subplots(1, 1, figsize=(3 * wid, 1.5 * hei))

# Plot the PDF
x = np.linspace(-10, 10, 1000)
y = dist.pdf(x)
plt.plot(x, y, label="Normal Distribution")
plt.legend()
plt.xlabel("x")
plt.ylabel("PDF(X = x)")
plt.title("Q1 : PDF of Normal Distribution")
plt.show()
```




```
# This is using the in-built functions
# Note: Kurtosis has 2 definitions, so set the "fisher" parameter accordingly

# Finding out descriptive statistics
mean = np.mean(draws)
variance = np.var(draws)
skewness = scipy.stats.skew(draws)
kurtosis = scipy.stats.kurtosis(draws, fisher=False)

sigma_g = astroML.stats.sigmaG(draws)
mad = scipy.stats.median_absolute_deviation(draws)

std_dev = np.std(draws)
std_dev_from_mad = 1.482 * mad

print(f"Mean                    : {mean}")
print(f"Variance                : {variance}")
print(f"Skewness                : {skewness}")
print(f"Kurtosis                : {kurtosis}")
print()
print(f"Sigma_G                 : {sigma_g}")
print(f"Mean Absoute Devation   : {mad}")
print()
print(f"Standard Deviation      : {std_dev}")
print(f"Std. Dev. from MAD      : {std_dev_from_mad}")
```

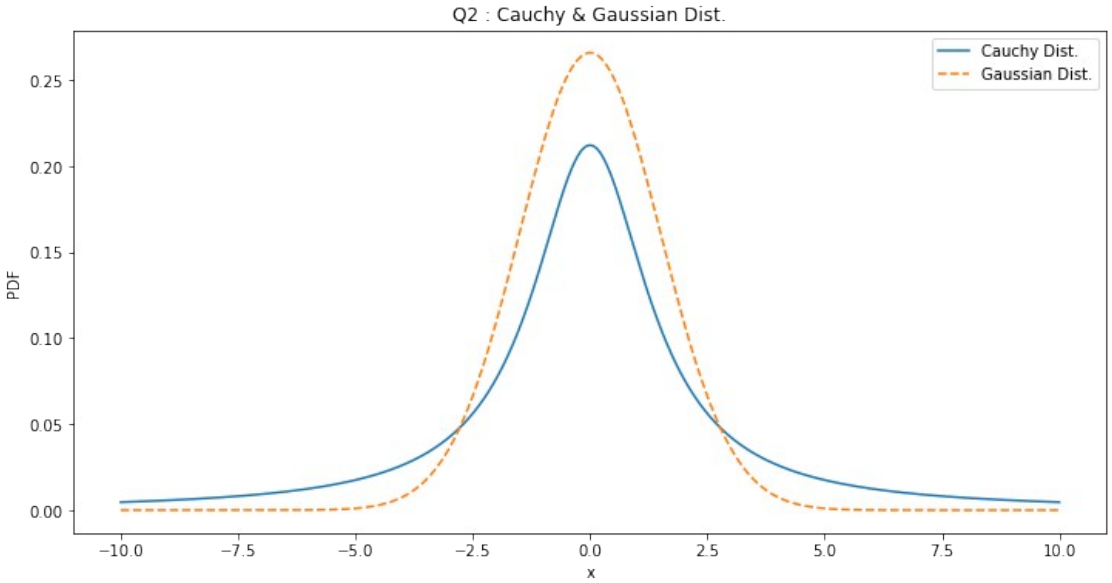
```
Mean                    : 1.517085231174232
Variance                : 0.24334417295611857
Skewness                : 0.0597160872560456
Kurtosis                : 3.2172088506819234
```

```
Sigma_G                 : 0.49429474862734474
Mean Absoute Devation   : 0.48887857436890086

Standard Deviation      : 0.49329927321669403
Std. Dev. from MAD      : 0.724518047214711
```

```
# This is using the scipy.stats.moments funciton
# There's a bit of math involved

# Finding out descriptive statistics
# mean = np.mean(draws)
# moments = scipy.stats.moment(draws, moment=[2, 3, 4])
# std_dev = scipy.stats.tstd(draws)

# print(f"Mean                    : {mean}")
# print(f"Variance                : {moments[0]}")
# print(f"Skewness                : {moments[1] / moments[0]**1.5}")
# print(f"Kurtosis                : {moments[2] / moments[0]**2}")
# print(f"Standard Deviation      : {std_dev}")
```

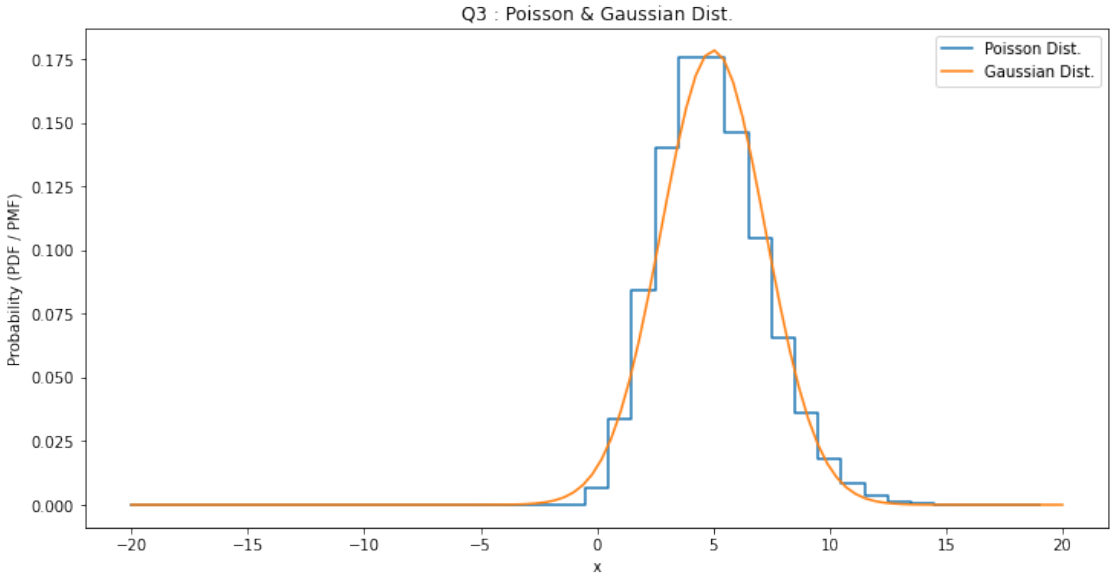
Q2

```
cauchy_dist = scipy.stats.cauchy(0, 1.5)
gaussian_dist = scipy.stats.norm(0, 1.5)

x = np.linspace(-10, 10, 1000)
cauchy_y = cauchy_dist.pdf(x)
gaussian_y = gaussian_dist.pdf(x)

# Adjust plot size
wid, hei = figaspect(1.)
fig, axs = plt.subplots(1, 1, figsize=(3 * wid, 1.5* hei))

axs.plot(x, cauchy_y, ls = "-", label="Cauchy Dist.")
axs.plot(x, gaussian_y, ls="--", label="Gaussian Dist.")
axs.legend()
plt.xlabel("x")
plt.ylabel("PDF")
plt.title("Q2 : Cauchy & Gaussian Dist.")
plt.show()
```



Q3

```
poisson_dist = scipy.stats.poisson(5)
gaussian_dist = scipy.stats.norm(5, np.sqrt(5))

poisson_x = np.arange(-20, 20)
gaussian_x = np.linspace(-20, 20, 100)
poisson_y = poisson_dist.pmf(poisson_x)
gaussian_y = gaussian_dist.pdf(gaussian_x)

# Adjust plot size
wid, hei = figaspect(1.)
fig, axs = plt.subplots(1, 1, figsize=(3 * wid, 1.5 * hei))

# axs.bar(poisson_x, poisson_y)
# axs.scatter(poisson_x, poisson_y, 60, marker="*", color="red")
axs.plot(poisson_x, poisson_y, label="Poisson Dist.", ds="steps-mid")
axs.plot(gaussian_x, gaussian_y, label="Gaussian Dist.", ls="-")
axs.legend()
plt.title("Q3 : Poisson & Gaussian Dist.")
plt.xlabel("x")
plt.ylabel("Probability (PDF / PMF)")
plt.show()
```



Q4

```
# Units: 1e-10
mean_lifetimes = [0.8920, 0.881, 0.8913, 0.9837, 0.8958]
errors = [0.00044, 0.009, 0.00032, 0.00048, 0.00045]

weights = 1 / np.square(errors)

wav_lifetime = np.average(mean_lifetimes, weights=weights)
wav_uncertainity = 1 / np.sqrt(np.sum(weights))

print(f"Weighted Mean Lifetime      : {wav_lifetime}")
print(f"Uncertainity of the Mean    : {wav_uncertainity}")
```

```
Weighted Mean Lifetime      : 0.9089185199574896
Uncertainity of the Mean    : 0.00020318737026848627
```

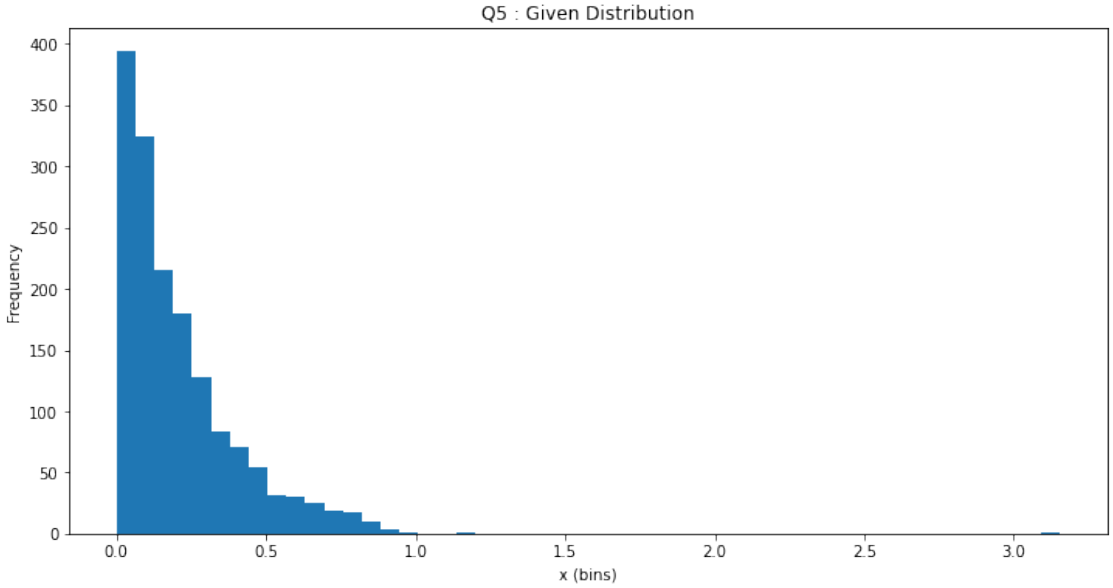
Q5

```
import pandas as pd
import os.path
folder_path = "/content/drive/My Drive/Course Work/PH6130 Data Science
Analysis"
file_name = "data.csv"

csv_path = os.path.join(folder_path, file_name)
data = pd.read_csv(csv_path, usecols=["eccentricity"])

data_without_na = data.dropna()
data_without_na.info()
```

```
processed_data = np.array(data_without_na).reshape(-1)
processed_data = processed_data[processed_data > 0]
```

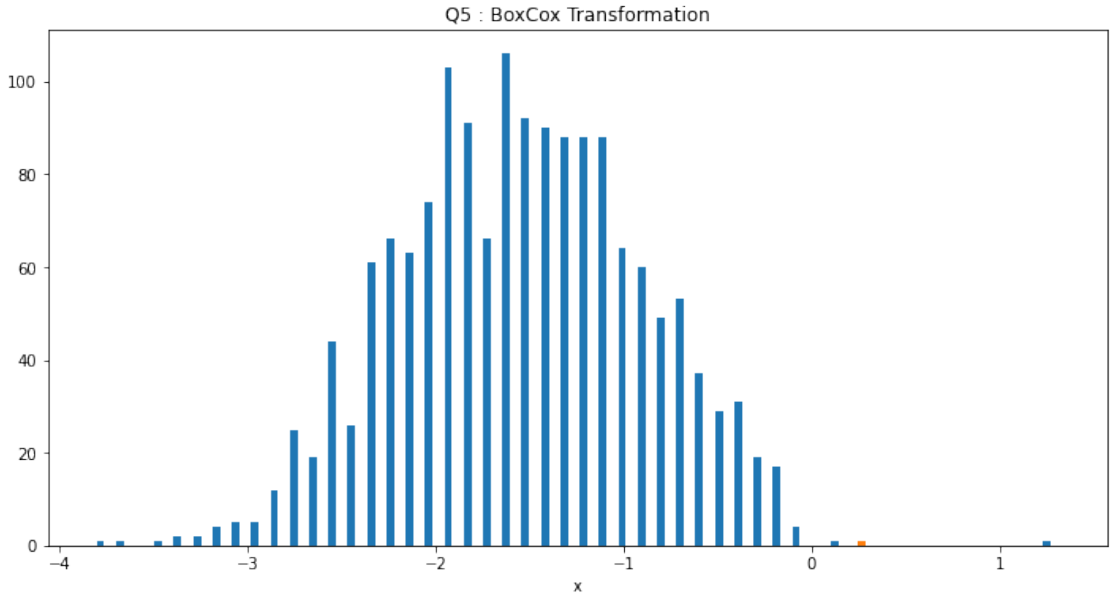
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1986 entries, 0 to 4910
Data columns (total 1 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   eccentricity  1986 non-null   float64
dtypes: float64(1)
memory usage: 31.0 KB
```

```
wid, hei = figaspect(1.)
fig, axs = plt.subplots(1, 1, figsize=(3 * wid, 1.5 * hei))

plt.hist(processed_data, bins=50);
plt.title("Q5 : Given Distribution")
plt.xlabel("x (bins)")
plt.ylabel("Frequency")
plt.show()
```

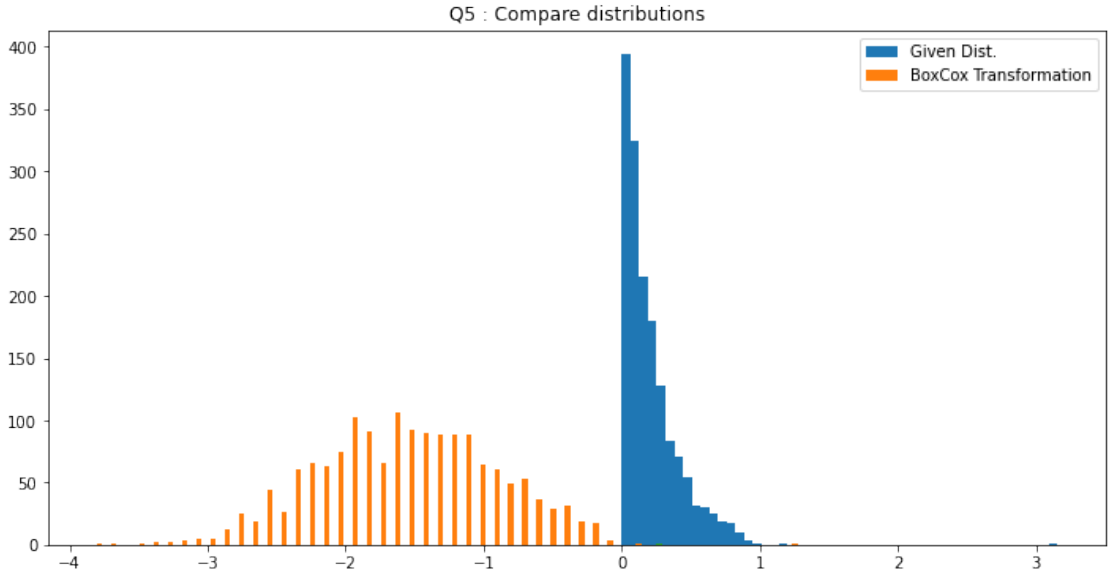



```
boxcox_data = np.array(scipy.stats.boxcox(processed_data),
dtype=object)
```

```
wid, hei = figaspect(1.)
fig, axs = plt.subplots(1, 1, figsize=(3 * wid, 1.5 * hei))

plt.hist(boxcox_data, bins=50, label="Given Dist.");
plt.title("Q5 : BoxCox Transformation")
plt.xlabel("x")
plt.ylabel("")
plt.show()
```




```
wid, hei = figaspect(1.)
fig, axs = plt.subplots(1, 1, figsize=(3 * wid, 1.5 * hei))

plt.hist(processed_data, 50, label="Given Dist.");
plt.hist(boxcox_data, 50, label="BoxCox Transformation");
plt.legend()
plt.title("Q5 : Compare distributions")
plt.show()
```



Assignment 2

PH6130 Data Science Analysis

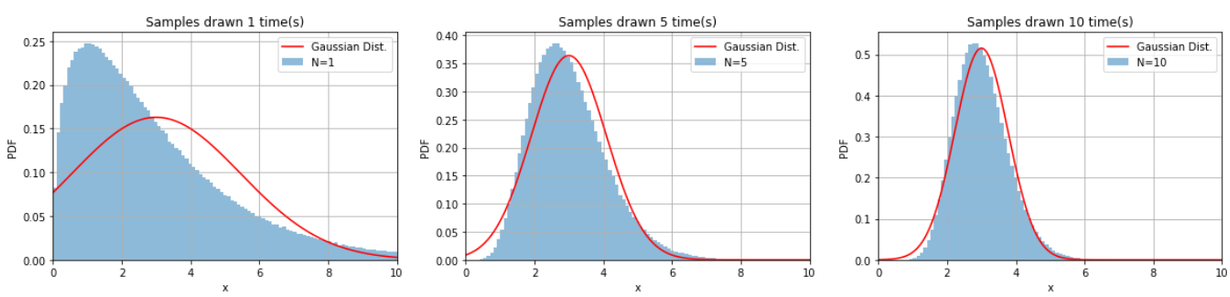
EE19BTECH11050 : N Tarun

In [ ]:
```
import scipy.stats
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.figure import figaspect
import pandas as pd
```

Q1

In [ ]:
```
# Generate the chi-square samples
N = [1, 5, 10]

x = np.random.chisquare(3, (max(N), int(1e6)))
```

In [ ]:
```
# Plot the results
fig = plt.figure(figsize=(20, 4))

for i in range(len(N)):
    ax = fig.add_subplot(1, 3, i + 1)

    # take the mean of the first N[i] samples
    x_i = x[:N[i], :].mean(0)

    # histogram the data
    ax.hist(x_i, bins=np.linspace(0, 10, 100), alpha=0.5, density=True, label=f"N={N
[i]}")

    # plot the expected gaussian pdf
    mu = 3
    std_dev = np.sqrt(2 * 3)
    dist = scipy.stats.norm(3, std_dev / np.sqrt(N[i]))

    x_pdf = np.linspace(0, 10, 1000)
    ax.plot(x_pdf, dist.pdf(x_pdf), color="r", label="Gaussian Dist.")

    ax.set_title(f"Samples drawn {N[i]} time(s)")

    plt.xlim(0, 10)
    plt.grid()
    plt.legend()
    plt.xlabel("x")
    plt.ylabel("PDF")

plt.show()
```



Q2

In [ ]:
```
data = pd.read_csv("http://www.iith.ac.in/~shantanud/test.dat",  delim_whitespace=True)
```

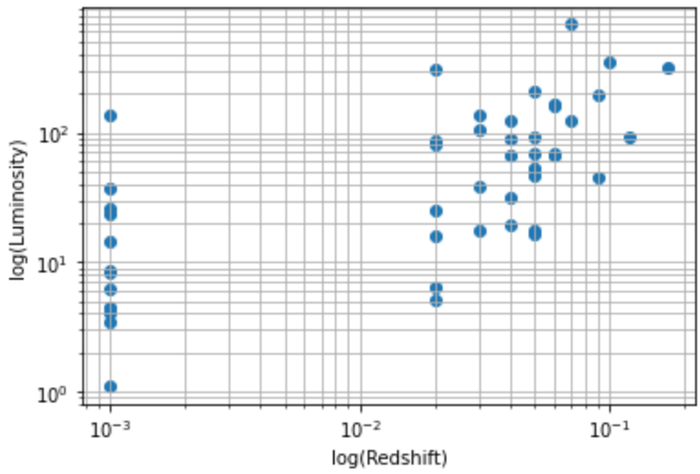
In [ ]:
```
data.head()
```

Out[ ]:

	#Lx	z
0	345.2	0.10
1	66.3	0.04
2	684.0	0.07
3	209.0	0.05
4	16.0	0.02

In [ ]:
```
datanp = np.array(data)
x = datanp[:, 1]
y = datanp[:, 0]
```

In [ ]:
```
plt.scatter(x, y);
plt.xscale("log")
plt.yscale("log")
plt.grid(True, which="both")
plt.xlabel("log(Redshift)")
plt.ylabel("log(Luminosity)")
plt.show();
```



In [ ]:
```
# spearman, pearson, kendall-tau & p-value
rho, spearman_p = scipy.stats.spearmanr(datanp[:, 1], datanp[:, 0])
corr_coeff, pearson_p = scipy.stats.pearsonr(datanp[:, 1], datanp[:, 0])
tau, kendall_p = scipy.stats.kendalltau(datanp[:, 1], datanp[:, 0])
```

In [ ]:
```
print(f"Spearman    : r             = {rho} \t p = {spearman_p}")
print(f"Pearson     : Corr.Coeff.   = {corr_coeff} \t p = {pearson_p}")
print(f"Kendall's   : tau           = {tau} \t p = {kendall_p}")
```

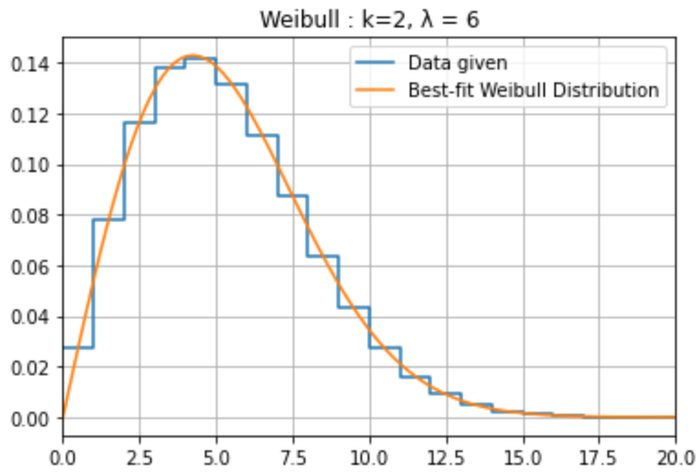
```
Spearman    : r             = 0.6596325957535454        p = 6.166489759081011e-07
Pearson     : Corr.Coeff.   = 0.5144497852670243        p = 0.00025464716576124137
Kendall's   : tau           = 0.5029584682704178        p = 2.969686227473415e-06
```

It is not trivial to determine if the data is correlated, from the plot.

But from the correlation coefficient values listed above, it can be seen that the data is correlated to a certain degree.

Q3

In [ ]:
```
x = np.arange(0, 21)
y = [2.75, 7.80, 11.64, 13.79, 14.20, 13.15, 11.14, 8.72, 6.34, 4.30, 2.73, 1.62, 0.91,
0.48, 0.24, 0.11, 0.05, 0.02, 0.01, 0.00, 0.00]
y = [s/100 for s in y]

x2 = np.linspace(0, 20, 100)
k = 2
L = 6
y2 = (k/L) * (x2 / L)**(k-1) * np.exp(-(x2/L)**k)

plt.plot(x, y, ds="steps-post", label="Data given");
plt.plot(x2, y2, label="Best-fit Weibull Distribution")

plt.grid()
plt.xlim(0, 20)
plt.legend()
plt.title("Weibull : k=2, λ = 6")
plt.show();
```



Q4

In [ ]:
```
draws = np.random.normal(0, 1, (1000, 2))

r, pearson_p = scipy.stats.pearsonr(draws[:, 0], draws[:, 1])

t = r * np.sqrt((1000 - 2) / (1 - r**2))

p_value = 0

if(t > 0):
    p_value = 2 * (1 - scipy.stats.t.cdf(t, 1000 - 2))
else:
    p_value = 2 * scipy.stats.t.cdf(t, 1000 - 2)

print(f"Pearson Correlation Coefficient : {corr_coeff}")
print(f"p-value from Pearson            : {pearson_p}")
print(f"p-value from Student's t dist.  : {p_value}")
```

```
Pearson Correlation Coefficient : 0.5144497852670243
p-value from Pearson            : 0.4283142290564171
p-value from Student's t dist.  : 0.4283142290564004
```

p-value calculated using Pearson Scipy module and using the Student's t distribution are very close.

Assignment 3 - Data Science Analysis

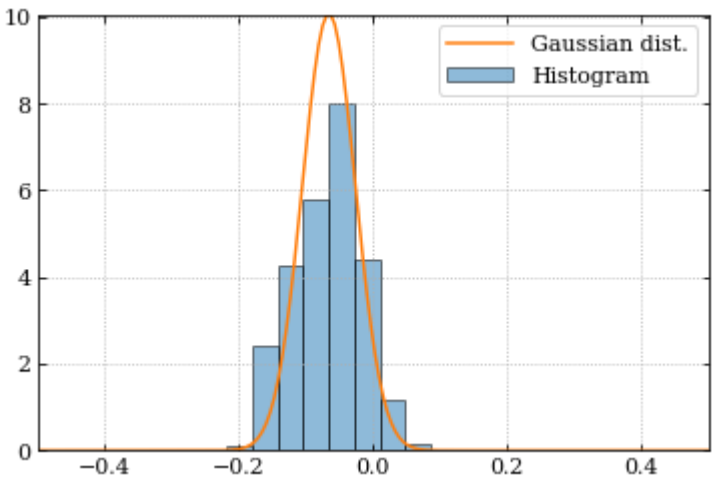
EE19BTECH11050 - N Tarun

In [89]:
```
!pip install astroML --quiet
```

In [90]:
```
import scipy.stats
from scipy.optimize import curve_fit
import numpy as np
import matplotlib.pyplot as plt
from astroML.stats import median_sigmaG
from astroML.resample import bootstrap

from astroML.plotting import setup_text_plots
setup_text_plots(fontsize=11, usetex=False)
```

Q1

In [91]:
```
n = 1000
N = 10000
samples = np.random.normal(0, 1, n)

medians, sigma_Gs = bootstrap(samples, N, median_sigmaG, dict(axis=1))
mu = np.mean(medians)
sigma = np.sqrt(np.pi / (2 * n))

x = np.linspace(-3, 3, 1000)
dist = scipy.stats.norm(mu, sigma)

plt.hist(medians, density=True, alpha=0.5, label="Histogram")
plt.plot(x, dist.pdf(x), label="Gaussian dist.")
plt.xlim(-0.5, 0.5)
plt.legend()
plt.grid()
plt.show();
```



Q2

In [92]:
```
# Row 5 to 20.
# Columns: x, y, sigma_y

data_given = np.array([
[1, 201, 592, 61, 9, -0.84],
[2 , 244, 401, 25, 4, 0.31],
[3 , 47, 583, 38, 11, 0.64],
[4 , 287, 402, 15, 7, -0.27],
[5 , 203, 495, 21, 5, -0.33],
[6 , 58, 173, 15, 9, 0.67],
[7 , 210, 479, 27, 4, -0.02],
[8 , 202, 504, 14, 4, -0.05],
[9 , 198, 510, 30, 11, -0.84],
[10, 158, 416, 16, 7, -0.69],
[11, 165, 393, 14, 5, 0.30],
[12, 201, 442, 25, 5, -0.46],
[13, 157, 317, 52, 5, -0.03],
[14, 131, 311, 16, 6, 0.50],
[15, 166, 400, 34, 6, 0.73],
[16, 160, 337, 31, 5, -0.52],
[17, 186, 423, 42, 9, 0.90],
[18, 125, 334, 26, 8, 0.40],
[19, 218, 533, 16, 6, -0.78],
[20, 146, 344, 22, 5, -0.56],
])
```

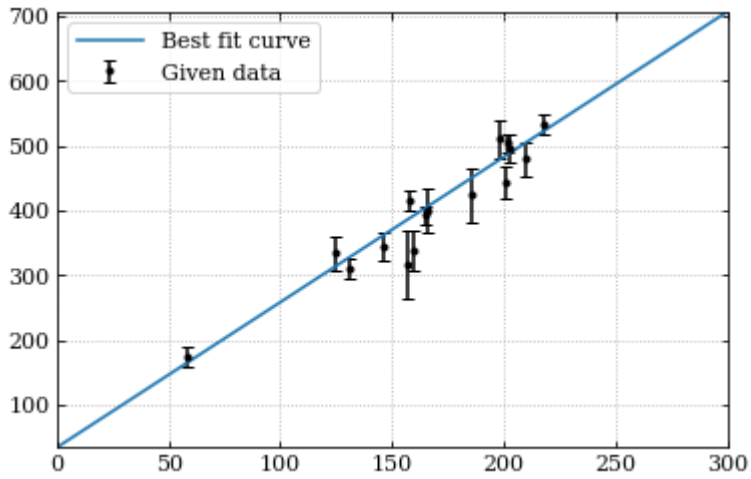
In [93]:
```
x = data_given[4:, 1]
y = data_given[4:, 2]
sigma_y = data_given[4:, 3]

def f(x, m, b):
    return m*x + b

params, _ = curve_fit(f, x, y, sigma=sigma_y)
m = params[0]
b = params[1]
print(f"m : {m}")
print(f"b : {b}")

x2 = np.linspace(0, 300, 100)
y2 = m * x2 + b
```

```
m : 2.2399208553933314
b : 34.047723577096654
```

In [94]:
```
print(sigma_y)
```

```
[21. 15. 27. 14. 30. 16. 14. 25. 52. 16. 34. 31. 42. 26. 16. 22.]
```

In [95]:
```
plt.errorbar(x, y, sigma_y, fmt=".", color="black", label="Given data")
plt.plot(x2, y2, label="Best fit curve")
plt.legend()
plt.grid()
plt.show()
```



Q3

In [96]:
```
dofs = np.array([0.96, 0.24, 3.84, 2.85])
n = 50
m = 1
dof = n - m
chi_sq = dofs * dof

p = 1 - scipy.stats.chi2(dof).cdf(chi_sq)
print(f"p-values : {p}")
```

```
p-values : [5.52926434e-01 9.99999992e-01 0.00000000e+00 1.21072929e-10]
```

Data Science Analysis - Assignment 4

EE19BTECH11050 - N Tarun

Q1

In [1]:
```
!pip install astroML --quiet
```

```
     |████████████████████████████████| 134 kB 26.0 MB/s
```

In [2]:
```
import numpy as np
import pandas as pd
import scipy.stats
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
from matplotlib.figure import figaspect


from astroML.plotting import setup_text_plots
setup_text_plots(fontsize=11, usetex=False)
```

In [3]:
```
data = np.loadtxt("https://www.iith.ac.in/~shantanud/testdata.dat")
```

In [4]:
```
x = data[:, 0]
y = data[:, 1]
sigma_y = data[:, 2]
```

In [5]:
```
def linear_func(x, a, b):
    return a*x + b

def quad_func(x, a, b, c):
    return a*(x**2) + b*x + c

def cubic_func(x, a, b, c, d):
    return a*(x**3) + b*(x**2) + c*x + d
```

In [6]:
```
popt_1, pconv_1 = curve_fit(linear_func, x, y, sigma=sigma_y)
popt_2, pconv_2 = curve_fit(quad_func, x, y, sigma=sigma_y)
popt_3, pconv_3 = curve_fit(cubic_func, x, y, sigma=sigma_y)
```

In [7]:
```
print(f"Best fit parameters for linear      : {popt_1}")
print(f"Best fit parameters for quadratic   : {popt_2}")
print(f"Best fit parameters for cubic       : {popt_3}")
```

```
Best fit parameters for linear      : [ 2.79789861 -1.11028082]
Best fit parameters for quadratic   : [ 0.50261293  2.38475187 -1.05578915]
Best fit parameters for cubic       : [-0.96724992  1.74451332  1.97184055 -1.02910462]
```

In [8]:
```
y_linear = linear_func(x, *popt_1)
y_quad = quad_func(x, *popt_2)
y_cubic = cubic_func(x, *popt_3)
```

In [9]:
```
chi_sq_lin = np.sum(np.square((y - y_linear) / sigma_y))
chi_sq_quad = np.sum(np.square((y - y_quad) / sigma_y))
chi_sq_cubic = np.sum(np.square((y - y_cubic) / sigma_y))
```

In [10]:
```
print(chi_sq_cubic)
```

```
9.085043873972351
```

In [11]:
```
v_quad = 3 - 2 # quad - linear
v_cubic = 4 - 2 # cubic - linear

# delta χ2
dcs_quad = np.abs(chi_sq_quad - chi_sq_lin)
dcs_cubic = np.abs(chi_sq_cubic - chi_sq_lin)

p_quad = 1 - scipy.stats.chi2(v_quad).cdf(dcs_quad)
p_cubic = 1 - scipy.stats.chi2(v_cubic).cdf(dcs_cubic)

print(f"p-value for quadratic best fit  : {p_quad}")
print(f"p-value for cubic best fit      : {p_cubic}")
```

```
p-value for quadratic best fit  : 0.17813275695316733
p-value for cubic best fit      : 0.32887884419522884
```

In [12]:
```
v_lin = 3 - 2 # |linear - quad|
v_cubic = 4 - 3 # cubic - quad

# delta χ2
dcs_lin = np.abs(chi_sq_lin - chi_sq_quad)
dcs_cubic = np.abs(chi_sq_cubic - chi_sq_quad)

p_lin = 1 - scipy.stats.chi2(v_lin).cdf(dcs_lin)
p_cubic = 1 - scipy.stats.chi2(v_cubic).cdf(dcs_cubic)
print(f"p-value for linear best fit : {p_lin}")
print(f"p-value for cubic best fit  : {p_cubic}")
```

```
p-value for linear best fit : 0.17813275695316733
p-value for cubic best fit  : 0.5214634480821769
```

In [13]:
```
def log_Lmax(y, y_fit, sigma_y):
    return np.sum(scipy.stats.norm.logpdf(y, y_fit, sigma_y))

def AIC(y, y_fit, sigma_y, k):
    return -2 * log_Lmax(y, y_fit, sigma_y) + 2 * k

def AIC_c(y, y_fit, sigma_y, k):
    N = np.size(y)
    return AIC(y, y_fit, sigma_y, k) + 2 * k * (k + 1) / (N - k - 1)

def BIC(y, y_fit, sigma_y, k):
    N = np.size(y)
    return -2 * log_Lmax(y, y_fit, sigma_y) + k * np.log(N)
```

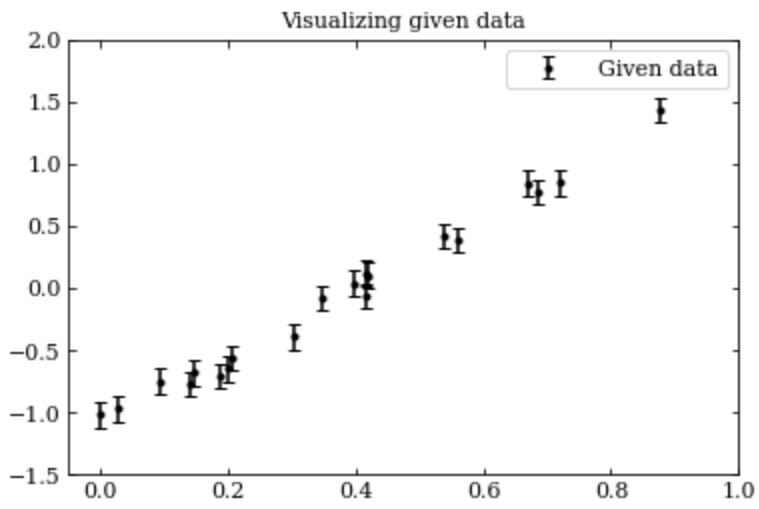
In [14]:
```
aic_1 = AIC(y, y_linear, sigma_y, 2)
aic_2 = AIC(y, y_quad, sigma_y, 3)
aic_3 = AIC(y, y_cubic, sigma_y, 4)

aic_c1 = AIC_c(y, y_linear, sigma_y, 2)
aic_c2 = AIC_c(y, y_quad, sigma_y, 3)
aic_c3 = AIC_c(y, y_cubic, sigma_y, 4)

bic_1 = BIC(y, y_linear, sigma_y, 2)
bic_2 = BIC(y, y_quad, sigma_y, 3)
bic_3 = BIC(y, y_cubic, sigma_y, 4)

print(f"AIC for linear best fit                : {aic_1}")
print(f"AIC for quadratic best fit             : {aic_2}")
print(f"AIC for cubic best fit                 : {aic_3}")
print()
print(f"AIC_corrected for linear best fit      : {aic_c1}")
print(f"AIC_corrected for quadratic best fit   : {aic_c2}")
print(f"AIC_corrected for cubic best fit       : {aic_c3}")
print()
print(f"BIC for linear best fit                : {bic_1}")
print(f"BIC for quadratic best fit             : {bic_2}")
print(f"BIC for cubic best fit                 : {bic_3}")
```

```
AIC for linear best fit                : -40.03668681607269
AIC for quadratic best fit             : -39.849820624005616
AIC for cubic best fit                 : -38.26081851760256

AIC_corrected for linear best fit      : -39.330804463131514
AIC_corrected for quadratic best fit   : -38.349820624005616
AIC_corrected for cubic best fit       : -35.594151850935894

BIC for linear best fit                : -38.04522226896471
BIC for quadratic best fit             : -36.862623803343645
BIC for cubic best fit                 : -34.2778894233866
```

According to AIC, AIC & AIC_corrected for linear model are the least, so it's the best model.

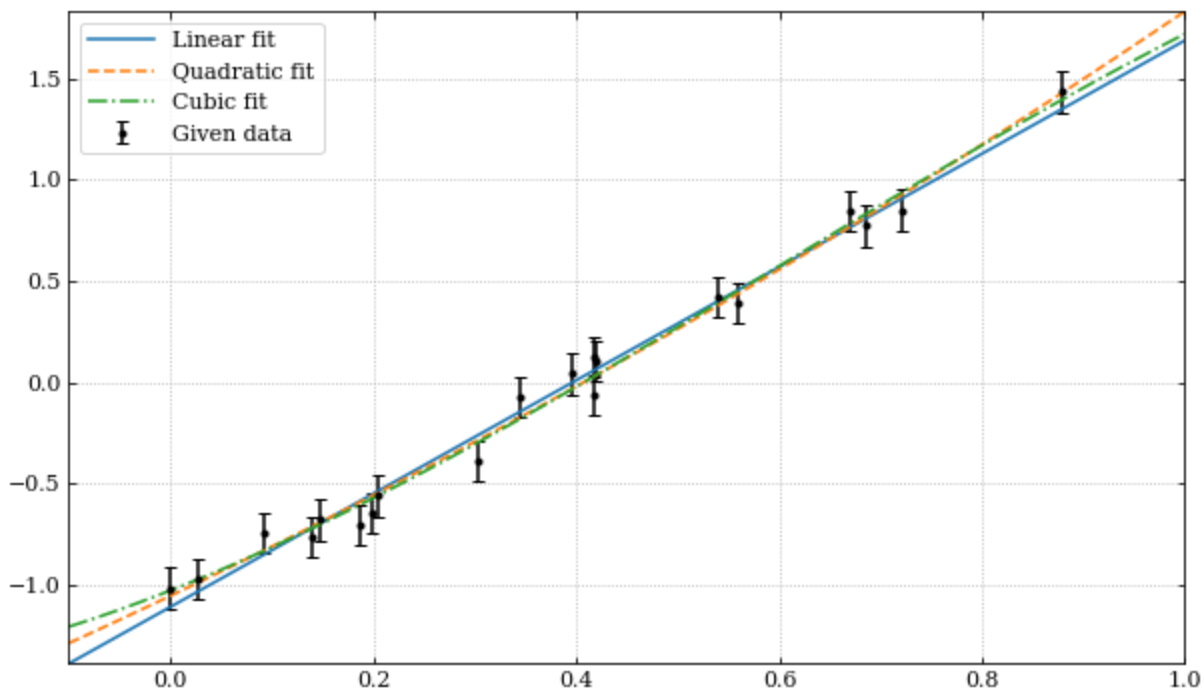
According to BIC, BIC for linear model is the least, so it's the best model

In [15]:
```
plt.errorbar(x, y, sigma_y, fmt=".", color="black", label="Given data")
plt.legend()
plt.title("Visualizing given data")
plt.xlim(-0.05, 1)
plt.ylim(-1.5, 2)
plt.show()
```



In [16]:
```
x_ = np.linspace(-0.1, 1, 100)
y_1 = linear_func(x_, *popt_1)
y_2 = quad_func(x_, *popt_2)
y_3 = cubic_func(x_, *popt_3)
```

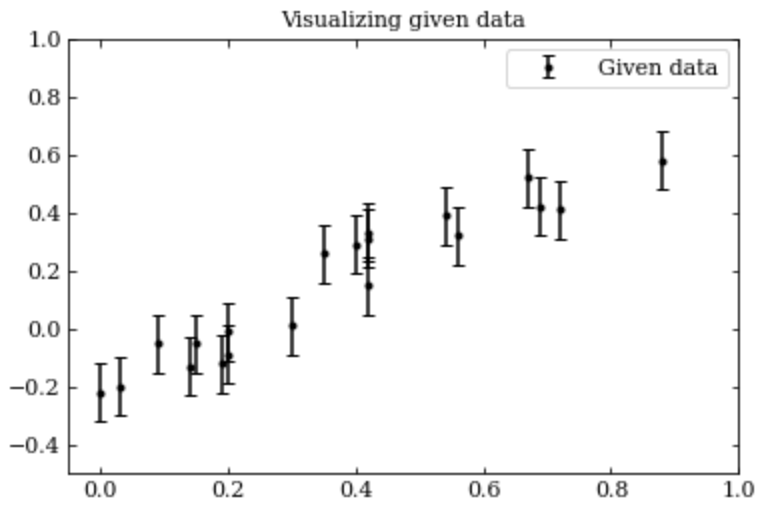
In [17]:
```
# wid, hei = figaspect(1)
# fig, axs = plt.subplots(1, 3, figsize=(4 * wid, 1 * hei))

# for i in range(3):
#     # axs[i].errorbar(x, y, sigma_y, fmt=".", color="black", label="Given data")
#     axs[i].scatter(x, y, color="black", marker="*", label="Given data")

# axs[0].plot(x_, y_1)
# axs[1].plot(x_, y_2)
# axs[2].plot(x_, y_3)
# plt.show()

plt.figure(figsize=(10, 6))
plt.errorbar(x, y, sigma_y, fmt=".", color="black", label="Given data")
plt.plot(x_, y_1, ls="-", label="Linear fit")
plt.plot(x_, y_2, ls="--", label="Quadratic fit")
plt.plot(x_, y_3, ls="-.", label="Cubic fit")
plt.legend()
plt.grid()
plt.show()
```



Q2

In [18]:
```
data = np.array([[ 0.42,  0.72,  0.  ,  0.3 ,  0.15,
                   0.09,  0.19,  0.35,  0.4 ,  0.54,
                   0.42,  0.69,  0.2 ,  0.88,  0.03,
                   0.67,  0.42,  0.56,  0.14,  0.2  ],
                 [ 0.33,  0.41, -0.22,  0.01, -0.05,
                  -0.05, -0.12,  0.26,  0.29,  0.39,
                   0.31,  0.42, -0.01,  0.58, -0.2 ,
                   0.52,  0.15,  0.32, -0.13, -0.09 ],
                 [ 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
```

In [19]:
```
plt.errorbar(x, y, sigma_y, fmt=".", color="black", label="Given data")
plt.legend()
plt.title("Visualizing given data")
plt.xlim(-0.05, 1)
plt.ylim(-0.5, 1)
plt.show()
```



In [20]:
```
popt_1, pconv_1 = curve_fit(linear_func, x, y, sigma=sigma_y)
popt_2, pconv_2 = curve_fit(quad_func, x, y, sigma=sigma_y)

print(f"Best fit linear parameters      : {popt_1}")
print(f"Best fit quadratic parameters   : {popt_2}")
```

```
Best fit linear parameters      : [ 0.97502243 -0.20329576]
Best fit quadratic parameters   : [-0.5079882   1.39347967 -0.25864529]
```

In [21]:
```
y_linear = linear_func(x, *popt_1)
y_quad = quad_func(x, *popt_2)
```

In [22]:
```
aic_1 = AIC(y, y_linear, sigma_y, 2)
aic_2 = AIC(y, y_quad, sigma_y, 3)

bic_1 = BIC(y, y_linear, sigma_y, 2)
bic_2 = BIC(y, y_quad, sigma_y, 3)

print(f"AIC for linear best fit                : {aic_1}")
print(f"AIC for quadratic best fit             : {aic_2}")
print()
print(f"BIC for linear best fit                : {bic_1}")
print(f"BIC for quadratic best fit             : {bic_2}")
```

```
AIC for linear best fit                : -40.02173401322526
AIC for quadratic best fit             : -39.883027173008216

BIC for linear best fit                : -38.03026946611728
BIC for quadratic best fit             : -36.895830352346245
```

According to the website, linear model is favored, which matches the results from the above AIC & BIC values.

- Since AIC & BIC values for linear model are lower, it's the favored model

Q3

arXiv:2112.13903

Modeling Sparse Data Using MLE with Applications to Microbiome Data

About the paper:

- The paper deals with modelling sparse data such as microbiome data and transcriptomics, as the data is skewed and contains exceedingly high number of zeros.
- The paper derives maximum likelihood estimator (MLE) for zero-inflated models. And then derives the corresponding Fisher information matrices.
- The paper is using a statistical procedure for identifying most appropriate discrete probabilistic model for zero-inflated models based on the p-value of K-S test
- The paper is using K-S test to test if samples come from a discrete or mixed distribution.
- The paper uses the p-value of K-S test to discard features that do not show significant divergence (p-value > 0.05)
- The paper tests out various probabilistic models like Poisson, zero-inflated poisson and lists out the percentage of microbiomes features that are appropriately fitted by the distributions

The above paper does not violate guidelines listed on the Penn State website

- It is applied to 1D data (i.e. not applied to 2+ dimesnions)
- Zero-altered or hurdle model is selected based on previous applications for modeling data with excess or deficit of zeros (i.e. model not derived from dataset)

Q4 ¶

In [23]:
```
p_higgs = 10.0 ** np.array([-1, -2, -3, -5, -7, -9])
p_ligo = 2 * 1e-7

sig_higgs = scipy.stats.norm.isf(p_higgs)
sig_ligo = scipy.stats.norm.isf(p_ligo)

chi_sq = 65.2
v = 67
chi_sq_gof = 1 - scipy.stats.chi2(v).cdf(chi_sq)

print(f"Significance in terms of no. of sigmas of Higgs Boson Discovery : {sig_higgs}")
print(f"Significance in terms of no. of sigmas of LIGO discovery        : {sig_ligo}")
print(f"χ2 GOF                                                          : {chi_sq_gof}")
```

```
Significance in terms of no. of sigmas of Higgs Boson Discovery : [1.28155157 2.32634787 3.09023231 4.26489079 5.19933
758 5.99780702]
Significance in terms of no. of sigmas of LIGO discovery        : 5.068957749717791
χ2 GOF                                                          : 0.5394901931099038
```

Data Science Analysis - PH6130

N Tarun - EE19BTECH11050

Assignment 5

Q1

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats
```

In [2]:
```
# To download the .dat file
!wget http://astrostatistics.psu.edu/datasets/asteroid_dens.dat --no-check-certificate
```

```
--2022-02-21 11:04:41--  http://astrostatistics.psu.edu/datasets/asteroid_dens.dat
Resolving astrostatistics.psu.edu (astrostatistics.psu.edu)... 168.62.182.234
Connecting to astrostatistics.psu.edu (astrostatistics.psu.edu)|168.62.182.234|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://astrostatistics.psu.edu/datasets/asteroid_dens.dat [following]
--2022-02-21 11:04:41--  https://astrostatistics.psu.edu/datasets/asteroid_dens.dat
Connecting to astrostatistics.psu.edu (astrostatistics.psu.edu)|168.62.182.234|:443... connected.
WARNING: cannot verify astrostatistics.psu.edu's certificate, issued by ‘CN=InCommon RSA Server CA,OU=InCommon,O=Inter
net2,L=Ann Arbor,ST=MI,C=US’:
  Unable to locally verify the issuer's authority.
HTTP request sent, awaiting response... 200 OK
Length: 1517 (1.5K) [text/plain]
Saving to: ‘asteroid_dens.dat.3’

asteroid_dens.dat.3 100%[===================>]   1.48K  --.-KB/s    in 0s

2022-02-21 11:04:41 (104 MB/s) - ‘asteroid_dens.dat.3’ saved [1517/1517]
```

In [3]:
```
data = pd.read_csv("asteroid_dens.dat", skiprows=17, delim_whitespace=True).fillna(0)
display(data)
```

	Asteroid	Dens	+/-
1	Ceres	2.12	0.04
2	Pallas	2.71	0.11
4	Vesta	3.44	0.12
10	Hygiea	2.76	1.20
11	Parthenope	2.72	0.12
15	Eunomia	0.96	0.30
16	Psyche	2.00	0.60
20	Massalia	3.26	0.60
22	Kalliope	2.50	0.30
45	Eugenia	1.20	0.40
87	Sylvia	1.62	0.30
90	Antiope	1.30	0.00
121	Hermione	1.96	0.34
243	Ida	2.60	0.50
253	Mathilde	1.30	0.20
433	Eros	2.67	0.03
704	Interamnia	4.40	2.10
762	Pulcova	1.80	0.80
804	Hispania	4.90	3.90
1999	KW4	2.39	0.90
2000	DP107	1.62	1.05
2000	UG11	1.47	0.95
854	Frostia	0.89	0.13
1089	Tama	2.52	0.30
1313	Berna	1.21	0.25
4492	Debussy	0.90	0.10
617	Patroclus	0.80	0.15

In [4]:
```
asteroids = data.loc[:, "Asteroid"].to_numpy()
dens = data.loc[:, "Dens"].to_numpy()
errors = data.loc[:, "+/-"].to_numpy()
print(dens.shape)

log_dens = np.log10(dens, where=dens!=0)
```

```
(27,)
```

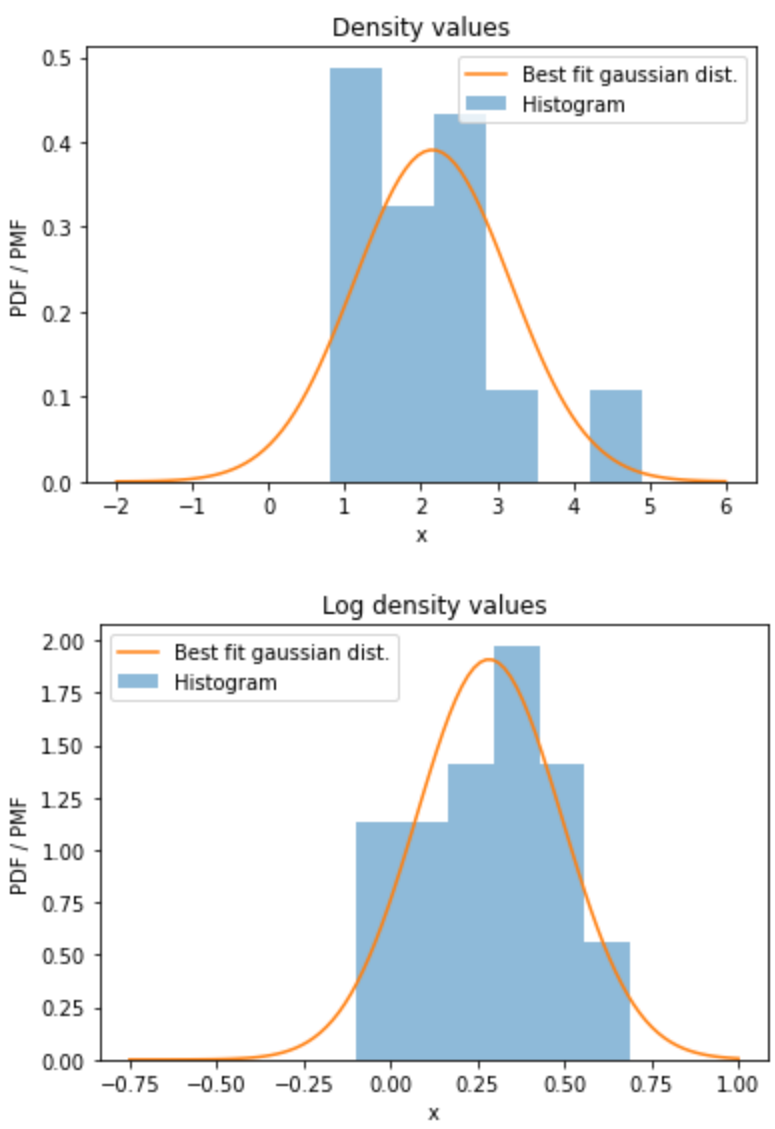
In [5]:
```
print(f"p-value for density values      : {scipy.stats.shapiro(dens)[1]}")
print(f"p-value for log density values  : {scipy.stats.shapiro(log_dens)[1]}")
```

```
p-value for density values      : 0.051220282912254333
p-value for log density values  : 0.5660525560379028
```

Log density data is closer to a Gaussian distribution than density data, because the p-value of < 0.05 rejects the null hypothesis that the data is from a gaussian distribution.

p-value being close to 0.05 makes density data less likely to be from gaussian distribution

In [6]:
```
x = np.linspace(-2, 6, 100)
plt.hist(dens, bins="auto", density=True, alpha=0.5, label="Histogram")
plt.plot(x, scipy.stats.norm.pdf(x, *scipy.stats.norm.fit(dens)), label="Best fit gaussian dist.")
plt.title("Density values")
plt.xlabel("x")
plt.ylabel("PDF / PMF")
plt.legend()
plt.show()

x2 = np.linspace(-0.75, 1, 100)
plt.hist(log_dens, bins="auto", density=True, alpha=0.5, label="Histogram")
plt.plot(x2, scipy.stats.norm.pdf(x2, *scipy.stats.norm.fit(log_dens)), label="Best fit gaussian dist.")
plt.title("Log density values")
plt.xlabel("x")
plt.ylabel("PDF / PMF")
plt.legend()
plt.show()
```



Q2

In [7]:
```
data2 = pd.read_csv("https://iith.ac.in/~shantanud/HIP_star.dat", delim_whitespace=True)
display(data2)
```

	HIP	Vmag	RA	DE	Plx	pmRA	pmDE	e_Plx	B-V
0	2	9.27	0.003797	-19.498837	21.90	181.21	-0.93	3.10	0.999
1	38	8.65	0.111047	-79.061831	23.84	162.30	-62.40	0.78	0.778
2	47	10.78	0.135192	-56.835248	24.45	-44.21	-145.90	1.97	1.150
3	54	10.57	0.151656	17.968956	20.97	367.14	-19.49	1.71	1.030
4	74	9.93	0.221873	35.752722	24.22	157.73	-40.31	1.36	1.068
...	...	...	...	...	...	...	...	...	...
2714	118207	8.38	359.662248	77.262113	23.37	18.87	-44.04	0.70	0.651
2715	118213	8.28	359.690763	31.939823	20.33	76.66	-134.59	0.94	0.734
2716	118251	8.16	359.778318	41.170547	22.91	82.20	3.56	0.82	0.652
2717	118254	7.72	359.787381	41.201736	22.19	80.21	4.40	0.80	0.563
2718	118311	11.85	359.954685	-38.252603	24.63	337.76	-112.81	2.96	1.391

2719 rows × 9 columns

In [8]:
```
colors_hyades = data2.query("RA >= 50 & RA <= 100 & DE >= 0 & DE <= 25 & pmRA >= 90 & pmRA <= 130 & pmDE >= -60 & pmDE
 <= -10")
display(colors_hyades)

colors_non_hyades = data2.query("RA < 50 | RA > 100 | DE < 0 | DE > 25 | pmRA < 90 | pmRA > 130 | pmDE < -60 | pmDE > -
10")
display(colors_non_hyades)
```

	HIP	Vmag	RA	DE	Plx	pmRA	pmDE	e_Plx	B-V
532	18735	5.89	60.202858	18.194069	21.99	129.49	-28.27	0.81	0.319
536	18946	10.12	60.912353	19.455094	23.07	119.02	-34.19	2.12	1.095
540	19148	7.85	61.566899	15.698168	21.41	118.53	-19.59	1.47	0.593
542	19207	10.49	61.754794	15.335078	23.57	122.63	-18.96	2.26	1.180
544	19261	6.02	61.924609	15.162843	21.27	127.06	-22.75	1.03	0.397
...	...	...	...	...	...	...	...	...	...
679	21762	9.47	70.105891	16.513734	23.65	91.94	-30.69	2.53	1.096
688	22044	5.39	71.107373	11.146169	20.73	98.87	-13.47	0.88	0.251
694	22224	9.60	71.705813	17.748406	24.11	96.93	-33.93	1.72	0.967
704	22496	7.10	72.599450	17.202738	22.96	102.78	-29.70	1.17	0.563
709	22607	6.30	72.958017	13.655194	23.91	106.84	-16.00	1.04	0.502

93 rows × 9 columns

	HIP	Vmag	RA	DE	Plx	pmRA	pmDE	e_Plx	B-V
0	2	9.27	0.003797	-19.498837	21.90	181.21	-0.93	3.10	0.999
1	38	8.65	0.111047	-79.061831	23.84	162.30	-62.40	0.78	0.778
2	47	10.78	0.135192	-56.835248	24.45	-44.21	-145.90	1.97	1.150
3	54	10.57	0.151656	17.968956	20.97	367.14	-19.49	1.71	1.030
4	74	9.93	0.221873	35.752722	24.22	157.73	-40.31	1.36	1.068
...	...	...	...	...	...	...	...	...	...
2714	118207	8.38	359.662248	77.262113	23.37	18.87	-44.04	0.70	0.651
2715	118213	8.28	359.690763	31.939823	20.33	76.66	-134.59	0.94	0.734
2716	118251	8.16	359.778318	41.170547	22.91	82.20	3.56	0.82	0.652
2717	118254	7.72	359.787381	41.201736	22.19	80.21	4.40	0.80	0.563
2718	118311	11.85	359.954685	-38.252603	24.63	337.76	-112.81	2.96	1.391

2626 rows × 9 columns

In [9]:
```
print(scipy.stats.ttest_ind(colors_hyades["B-V"], colors_non_hyades["B-V"]))
```

```
Ttest_indResult(statistic=-3.860436921860911, pvalue=0.00011582222192442334)
```

p-value < 0.05. This means null-hypothesis false (is rejected).

Hyades and Non-Hyades colors are not related.

This means Hyades color differ from that of non-Hyades.

PH6130 Assignment 6 - Data Science Analysis

N Tarun - EE19BTECH11050

In [27]:
```
!pip install emcee --quiet
!pip install astroML --quiet
```

In [28]:
```
from io import StringIO # to read data
import pandas as pd
import numpy as np
import emcee
from astroML.plotting import plot_mcmc
import matplotlib.pyplot as plt
import scipy.stats
```

Q1

Einstein prediction: 1.74 arc-seconds
Newtonian prediction: 0.87 arc-seconds

Eddington team report: 1:61 +/- 0:40 arc-seconds
Crommelin team report: 1:98 +/- 0:16 arc-seconds

Given assumption of Gaussian likelihood.

In [29]:
```
einstein_e = scipy.stats.norm.pdf(1.74, 1.61, 0.40)
einstein_c = scipy.stats.norm.pdf(1.74, 1.98, 0.16)
newton_e = scipy.stats.norm.pdf(1.74 / 2, 1.61, 0.40)
newton_c = scipy.stats.norm.pdf(1.74 / 2, 1.98, 0.16)

# M1 = Einsteinian, M2 = Newtonian
bayes_factor = einstein_e * einstein_c / (newton_e * newton_c)
print(f"Bayes factor (einstein / newton): {bayes_factor}")

# M1 = Newtonian, M2 = Einsteinian
print(f"Bayes factor (newton / einstein): {1/bayes_factor}")
```

```
Bayes factor (einstein / newton): 48164622958.34179
Bayes factor (newton / einstein): 2.076212661033209e-11
```

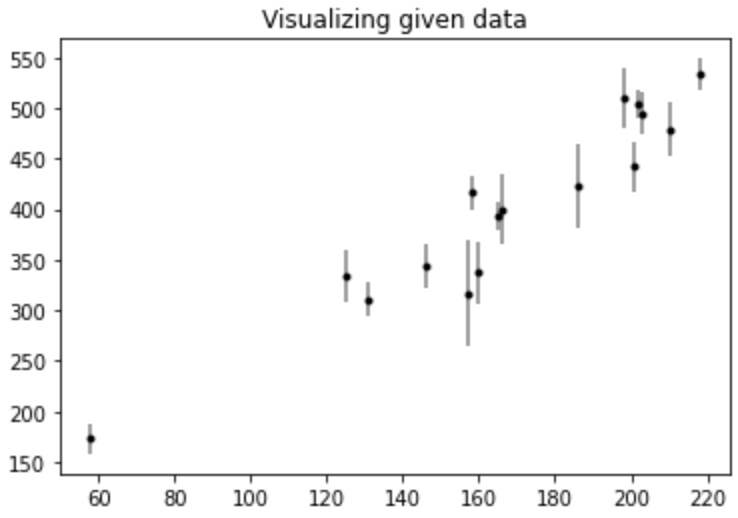
Q2

In [30]:
```
data_string = """1 201 592 61 9 -0.84
2 244 401 25 4 0.31
3 47 583 38 11 0.64
4 287 402 15 7 -0.27
5 203 495 21 5 -0.33
6 58 173 15 9 0.67
7 210 479 27 4 -0.02
8 202 504 14 4 -0.05
9 198 510 30 11 -0.84
10 158 416 16 7 -0.69
11 165 393 14 5 0.30
12 201 442 25 5 -0.46
13 157 317 52 5 -0.03
14 131 311 16 6 0.50
15 166 400 34 6 0.73
16 160 337 31 5 -0.52
17 186 423 42 9 0.90
18 125 334 26 8 0.40
19 218 533 16 6 -0.78
20 146 344 22 5 -0.56"""
```

In [31]:
```
data = np.loadtxt(StringIO(data_string), delimiter=" ")
display(data)
```

```
array([[ 1.00e+00,  2.01e+02,  5.92e+02,  6.10e+01,  9.00e+00, -8.40e-01],
       [ 2.00e+00,  2.44e+02,  4.01e+02,  2.50e+01,  4.00e+00,  3.10e-01],
       [ 3.00e+00,  4.70e+01,  5.83e+02,  3.80e+01,  1.10e+01,  6.40e-01],
       [ 4.00e+00,  2.87e+02,  4.02e+02,  1.50e+01,  7.00e+00, -2.70e-01],
       [ 5.00e+00,  2.03e+02,  4.95e+02,  2.10e+01,  5.00e+00, -3.30e-01],
       [ 6.00e+00,  5.80e+01,  1.73e+02,  1.50e+01,  9.00e+00,  6.70e-01],
       [ 7.00e+00,  2.10e+02,  4.79e+02,  2.70e+01,  4.00e+00, -2.00e-02],
       [ 8.00e+00,  2.02e+02,  5.04e+02,  1.40e+01,  4.00e+00, -5.00e-02],
       [ 9.00e+00,  1.98e+02,  5.10e+02,  3.00e+01,  1.10e+01, -8.40e-01],
       [ 1.00e+01,  1.58e+02,  4.16e+02,  1.60e+01,  7.00e+00, -6.90e-01],
       [ 1.10e+01,  1.65e+02,  3.93e+02,  1.40e+01,  5.00e+00,  3.00e-01],
       [ 1.20e+01,  2.01e+02,  4.42e+02,  2.50e+01,  5.00e+00, -4.60e-01],
       [ 1.30e+01,  1.57e+02,  3.17e+02,  5.20e+01,  5.00e+00, -3.00e-02],
       [ 1.40e+01,  1.31e+02,  3.11e+02,  1.60e+01,  6.00e+00,  5.00e-01],
       [ 1.50e+01,  1.66e+02,  4.00e+02,  3.40e+01,  6.00e+00,  7.30e-01],
       [ 1.60e+01,  1.60e+02,  3.37e+02,  3.10e+01,  5.00e+00, -5.20e-01],
       [ 1.70e+01,  1.86e+02,  4.23e+02,  4.20e+01,  9.00e+00,  9.00e-01],
       [ 1.80e+01,  1.25e+02,  3.34e+02,  2.60e+01,  8.00e+00,  4.00e-01],
       [ 1.90e+01,  2.18e+02,  5.33e+02,  1.60e+01,  6.00e+00, -7.80e-01],
       [ 2.00e+01,  1.46e+02,  3.44e+02,  2.20e+01,  5.00e+00, -5.60e-01]])
```

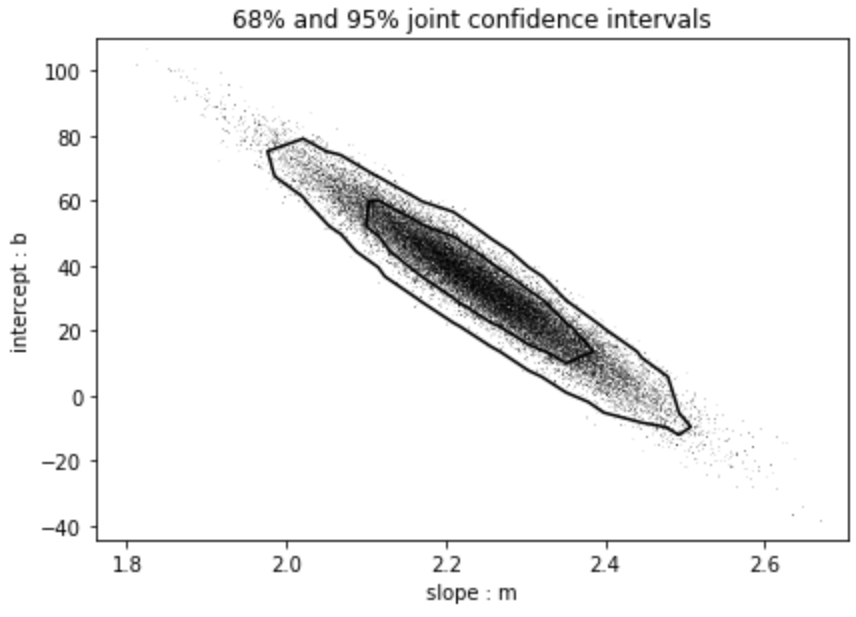
In [32]:
```
x = data[4:20, 1]
y = data[4:20, 2]
sigma_y = data[4:20, 3]

plt.errorbar(x, y, sigma_y, fmt='.k', ecolor='gray');
plt.title("Visualizing given data")
plt.show()
```



In [33]:
```
# model: y = mx + b
# m = theta[0], b = theta[1]
def log_prior(theta):
    # m,b can take any real value
    return 0

def log_likelihood(theta, x, y, e):
    return -0.5 * np.sum(np.log(2 * np.pi * e ** 2)
                         + (y - (theta[0] * x + theta[1])) ** 2 / e ** 2)

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

# same setup as above:
ndim, nwalkers = 2, 100
nsteps, nburn = 1000, 500

starting_guesses = np.random.rand(nwalkers, ndim)

# start m in [0, 100]
# and b in [0, 10]
starting_guesses[:, 0] *= 100
starting_guesses[:, 1] *= 10
# starting_guesses[:, 1] *= 1

sampler = emcee.EnsembleSampler(nwalkers, ndim, log_posterior, args=[x, y, sigma_y])
sampler.run_mcmc(starting_guesses, nsteps)

sample = sampler.chain  # shape = (nwalkers, nsteps, ndim)
sample = sampler.chain[:, nburn:, :].reshape(-1, 2)
print(sample.shape)
```

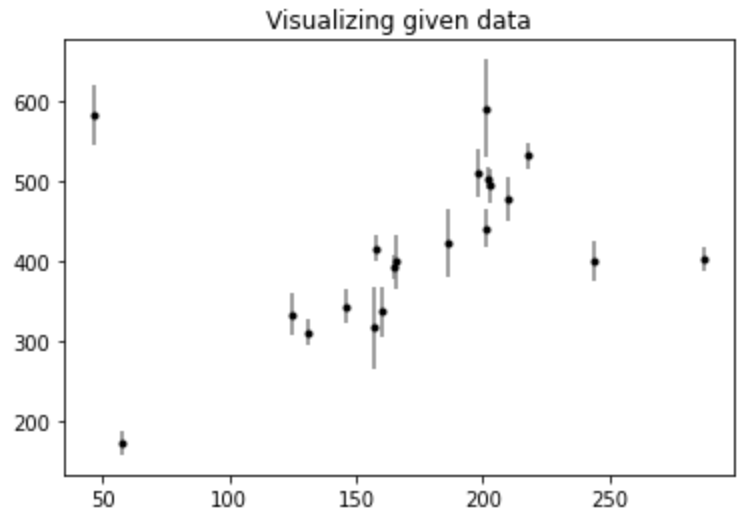
```
(50000, 2)
```

In [34]:
```
# astroML plot

fig = plt.figure()
ax = plot_mcmc(sample.T, fig=fig, labels=[r'$\mu$', r'$\sigma$'], colors='k')
ax[0].plot(sample[:, 0], sample[:, 1], ',k', alpha=0.1)
plt.xlabel('slope : m')
plt.ylabel('intercept : b')
plt.title("68% and 95% joint confidence intervals")
plt.show();
```

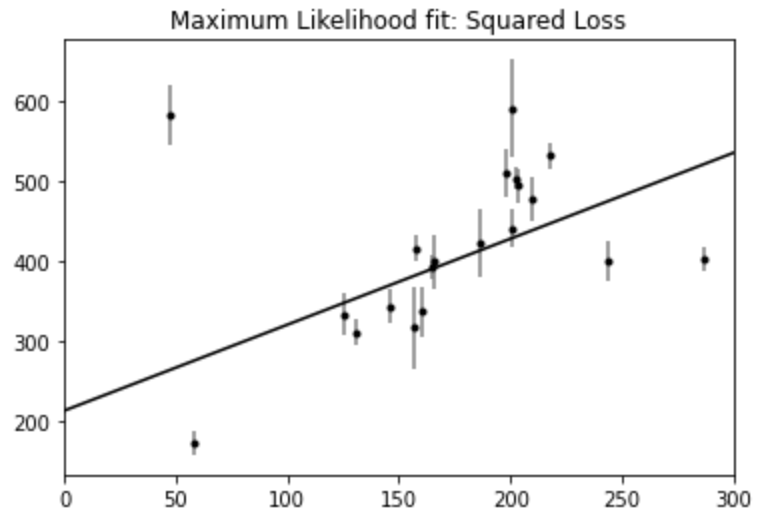


Q3

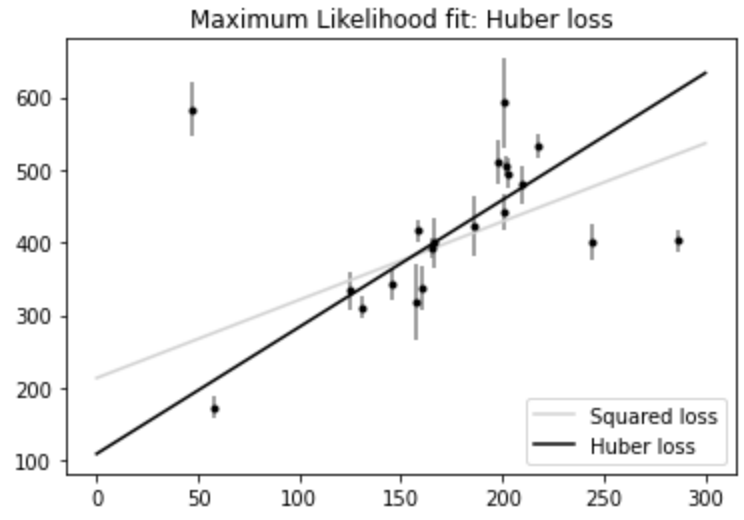
In [19]:
```
x = data[:, 1]
y = data[:, 2]
sigma_y = data[:, 3]

plt.errorbar(x, y, sigma_y, fmt='.k', ecolor='gray');
plt.title("Visualizing given data")
plt.show()
```



In [20]:
```
from scipy import optimize

def squared_loss(theta, x=x, y=y, e=sigma_y):
    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, sigma_y, fmt='.k', ecolor='gray')
plt.plot(xfit, theta1[0] + theta1[1] * xfit, '-k')
plt.title('Maximum Likelihood fit: Squared Loss');
plt.xlim(0, 300)
plt.show();
```



In [21]:
```
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=sigma_y, 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, sigma_y, fmt='.k', ecolor='gray')
plt.plot(xfit, theta1[0] + theta1[1] * xfit, color='lightgray', label="Squared loss")
plt.plot(xfit, theta2[0] + theta2[1] * xfit, color='black', label="Huber loss")
plt.legend()
plt.title('Maximum Likelihood fit: Huber loss');
```

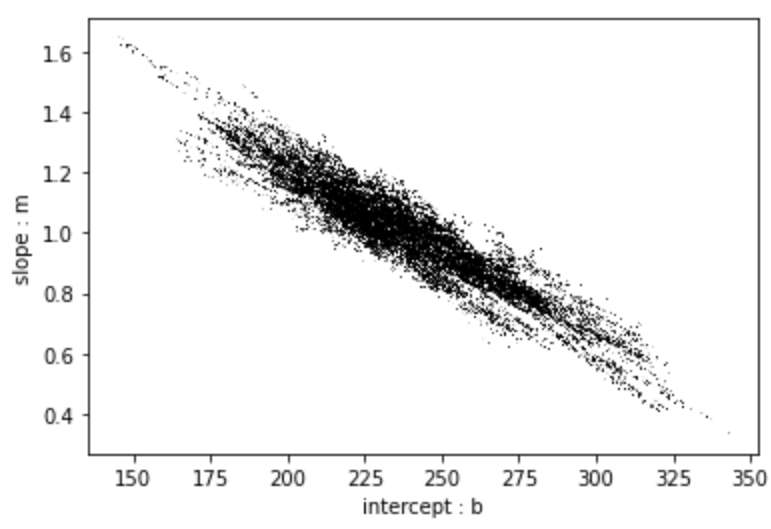


In [22]:
```
def log_prior(theta):
    #g_i needs to be between 0 and 1
    if (all(theta[2:] > 0) and all(theta[2:] < 1)):
        return 0
    else:
        return -np.inf  # recall log(0) = -inf

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
    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)
```

In [23]:
```
# Note that this step will take a few minutes to run!

ndim = 2 + len(x)  # number of parameters in the model
nwalkers = 50  # number of MCMC walkers
nburn = 10000  # "burn-in" period to let chains stabilize
nsteps = 15000  # number of MCMC steps to take

# set theta near the maximum likelihood, with
np.random.seed(0)
starting_guesses = np.zeros((nwalkers, ndim))
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, sigma_y, 50])
sampler.run_mcmc(starting_guesses, nsteps)

sample = sampler.chain  # shape = (nwalkers, nsteps, ndim)
sample = sampler.chain[:, nburn:, :].reshape(-1, ndim)
```

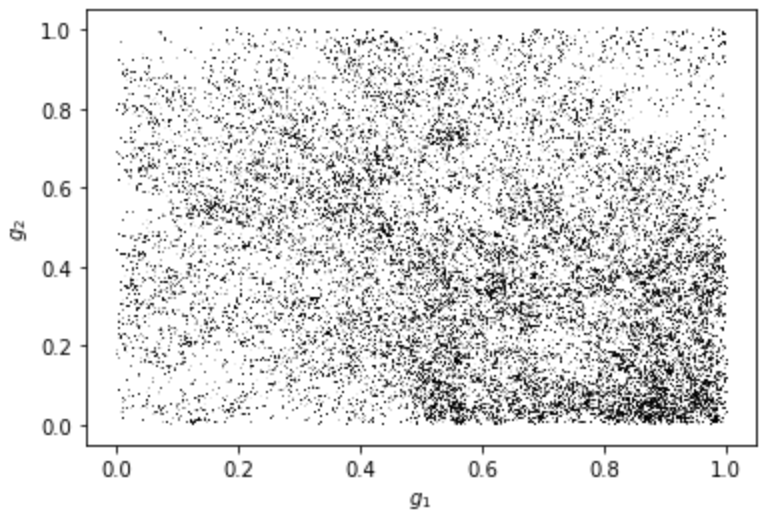
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: RuntimeWarning: divide by zero encountered in log
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: RuntimeWarning: divide by zero encountered in log
  # This is added back by InteractiveShellApp.init_path()
```

In [24]:
```
plt.plot(sample[:, 0], sample[:, 1], ',k', alpha=0.1)
plt.xlabel('intercept : b')
plt.ylabel('slope : m')
plt.show();
```

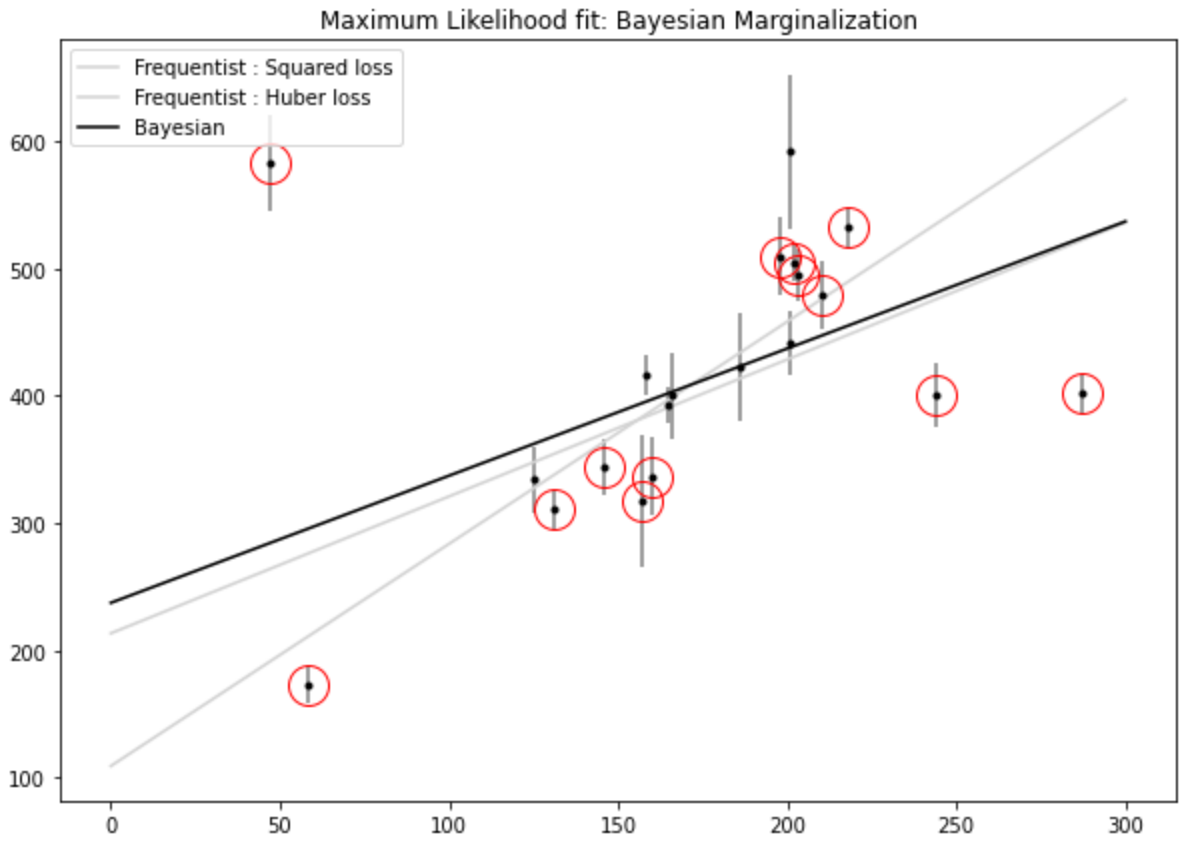


In [25]:
```
plt.plot(sample[:, 2], sample[:, 3], ',k', alpha=0.1)
plt.xlabel('$g_1$')
plt.ylabel('$g_2$')

print("g1 mean: {0:.2f}".format(sample[:, 2].mean()))
print("g2 mean: {0:.2f}".format(sample[:, 3].mean()))
```

```
g1 mean: 0.59
g2 mean: 0.39
```



In [26]:
```
theta3 = np.mean(sample[:, :2], 0)
g = np.mean(sample[:, 2:], 0)
outliers = (g < 0.5)

plt.figure(figsize=(10, 7))
plt.errorbar(x, y, sigma_y, fmt='.k', ecolor='gray')
plt.plot(xfit, theta1[0] + theta1[1] * xfit, color='lightgray', label="Frequentist : Squared loss")
plt.plot(xfit, theta2[0] + theta2[1] * xfit, color='lightgray', label="Frequentist : Huber loss")
plt.plot(xfit, theta3[0] + theta3[1] * xfit, color='black', label="Bayesian")
plt.plot(x[outliers], y[outliers], 'ro', ms=20, mfc='none', mec='red')
plt.legend()
plt.title('Maximum Likelihood fit: Bayesian Marginalization')
plt.show();
```



PH6130 Assignment 6 - Data Science Analysis

N Tarun - EE19BTECH11050

In [40]:
```
!pip install emcee --quiet
!pip install astroML --quiet
```

In [41]:
```
from io import StringIO # to read data
import pandas as pd
import numpy as np
import emcee
from astroML.plotting import plot_mcmc
import matplotlib.pyplot as plt
import scipy.stats
```

Q1

Einstein prediction: 1.74 arc-seconds
Newtonian prediction: 0.87 arc-seconds

Eddington team report: 1:61 +/- 0:40 arc-seconds
Crommelin team report: 1:98 +/- 0:16 arc-seconds

Given assumption of Gaussian likelihood.

In [42]:
```
einstein_e = scipy.stats.norm.pdf(1.74, 1.61, 0.40)
einstein_c = scipy.stats.norm.pdf(1.74, 1.98, 0.16)
newton_e = scipy.stats.norm.pdf(1.74 / 2, 1.61, 0.40)
newton_c = scipy.stats.norm.pdf(1.74 / 2, 1.98, 0.16)

# M1 = Einsteinian, M2 = Newtonian
bayes_factor = einstein_e * einstein_c / (newton_e * newton_c)
print(f"Bayes factor (einstein / newton): {bayes_factor}")

# M1 = Newtonian, M2 = Einsteinian
print(f"Bayes factor (newton / einstein): {1/bayes_factor}")
```

```
Bayes factor (einstein / newton): 48164622958.34179
Bayes factor (newton / einstein): 2.076212661033209e-11
```

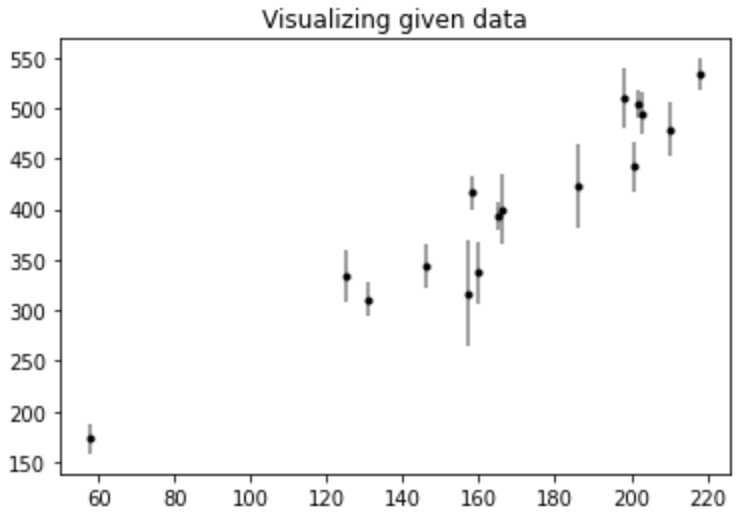
Q2

In [43]:
```
data_string = """1 201 592 61 9 -0.84
2 244 401 25 4 0.31
3 47 583 38 11 0.64
4 287 402 15 7 -0.27
5 203 495 21 5 -0.33
6 58 173 15 9 0.67
7 210 479 27 4 -0.02
8 202 504 14 4 -0.05
9 198 510 30 11 -0.84
10 158 416 16 7 -0.69
11 165 393 14 5 0.30
12 201 442 25 5 -0.46
13 157 317 52 5 -0.03
14 131 311 16 6 0.50
15 166 400 34 6 0.73
16 160 337 31 5 -0.52
17 186 423 42 9 0.90
18 125 334 26 8 0.40
19 218 533 16 6 -0.78
20 146 344 22 5 -0.56"""
```

In [44]:
```
data = np.loadtxt(StringIO(data_string), delimiter=" ")
display(data)
```

```
array([[ 1.00e+00,  2.01e+02,  5.92e+02,  6.10e+01,  9.00e+00, -8.40e-01],
       [ 2.00e+00,  2.44e+02,  4.01e+02,  2.50e+01,  4.00e+00,  3.10e-01],
       [ 3.00e+00,  4.70e+01,  5.83e+02,  3.80e+01,  1.10e+01,  6.40e-01],
       [ 4.00e+00,  2.87e+02,  4.02e+02,  1.50e+01,  7.00e+00, -2.70e-01],
       [ 5.00e+00,  2.03e+02,  4.95e+02,  2.10e+01,  5.00e+00, -3.30e-01],
       [ 6.00e+00,  5.80e+01,  1.73e+02,  1.50e+01,  9.00e+00,  6.70e-01],
       [ 7.00e+00,  2.10e+02,  4.79e+02,  2.70e+01,  4.00e+00, -2.00e-02],
       [ 8.00e+00,  2.02e+02,  5.04e+02,  1.40e+01,  4.00e+00, -5.00e-02],
       [ 9.00e+00,  1.98e+02,  5.10e+02,  3.00e+01,  1.10e+01, -8.40e-01],
       [ 1.00e+01,  1.58e+02,  4.16e+02,  1.60e+01,  7.00e+00, -6.90e-01],
       [ 1.10e+01,  1.65e+02,  3.93e+02,  1.40e+01,  5.00e+00,  3.00e-01],
       [ 1.20e+01,  2.01e+02,  4.42e+02,  2.50e+01,  5.00e+00, -4.60e-01],
       [ 1.30e+01,  1.57e+02,  3.17e+02,  5.20e+01,  5.00e+00, -3.00e-02],
       [ 1.40e+01,  1.31e+02,  3.11e+02,  1.60e+01,  6.00e+00,  5.00e-01],
       [ 1.50e+01,  1.66e+02,  4.00e+02,  3.40e+01,  6.00e+00,  7.30e-01],
       [ 1.60e+01,  1.60e+02,  3.37e+02,  3.10e+01,  5.00e+00, -5.20e-01],
       [ 1.70e+01,  1.86e+02,  4.23e+02,  4.20e+01,  9.00e+00,  9.00e-01],
       [ 1.80e+01,  1.25e+02,  3.34e+02,  2.60e+01,  8.00e+00,  4.00e-01],
       [ 1.90e+01,  2.18e+02,  5.33e+02,  1.60e+01,  6.00e+00, -7.80e-01],
       [ 2.00e+01,  1.46e+02,  3.44e+02,  2.20e+01,  5.00e+00, -5.60e-01]])
```

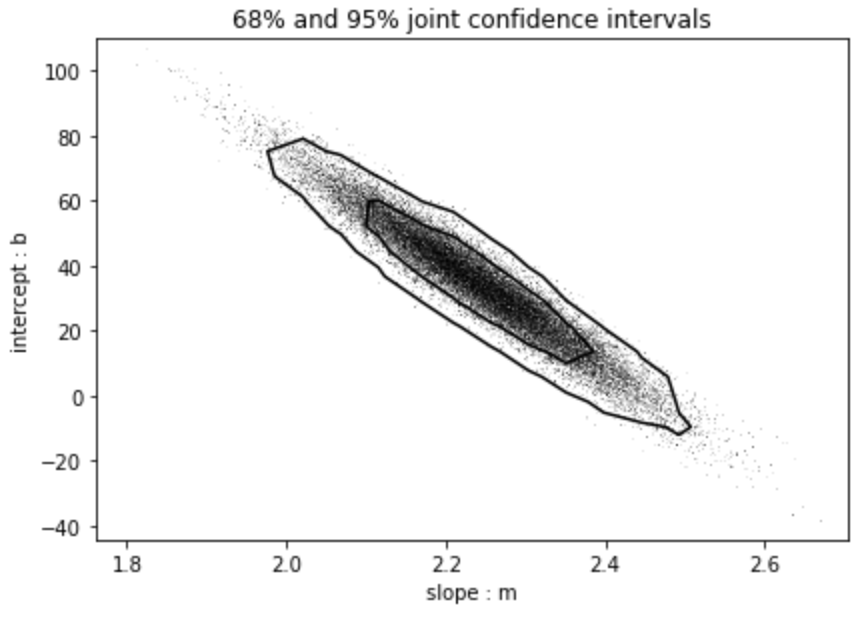
In [45]:
```
x = data[4:20, 1]
y = data[4:20, 2]
sigma_y = data[4:20, 3]

plt.errorbar(x, y, sigma_y, fmt='.k', ecolor='gray');
plt.title("Visualizing given data")
plt.show()
```



In [46]:
```
# model: y = mx + b
# m = theta[0], b = theta[1]
def log_prior(theta):
    # m,b can take any real value
    return 0

def log_likelihood(theta, x, y, e):
    return -0.5 * np.sum(np.log(2 * np.pi * e ** 2)
                         + (y - (theta[0] * x + theta[1])) ** 2 / e ** 2)

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

# same setup as above:
ndim, nwalkers = 2, 100
nsteps, nburn = 1000, 500

starting_guesses = np.random.rand(nwalkers, ndim)

# start m in [0, 100]
# and b in [0, 10]
starting_guesses[:, 0] *= 100
starting_guesses[:, 1] *= 10
# starting_guesses[:, 1] *= 1

sampler = emcee.EnsembleSampler(nwalkers, ndim, log_posterior, args=[x, y, sigma_y])
sampler.run_mcmc(starting_guesses, nsteps)

sample = sampler.chain  # shape = (nwalkers, nsteps, ndim)
sample = sampler.chain[:, nburn:, :].reshape(-1, 2)
print(sample.shape)
```

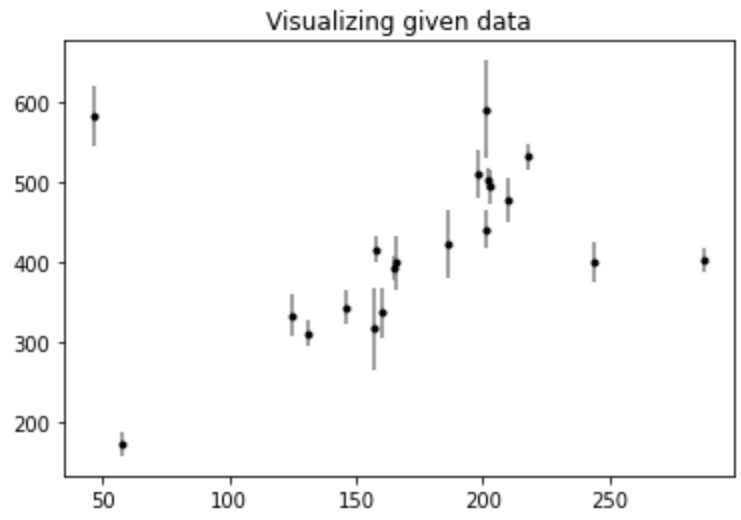
```
(50000, 2)
```

In [47]:
```
# astroML plot

fig = plt.figure()
ax = plot_mcmc(sample.T, fig=fig, labels=[r'$\mu$', r'$\sigma$'], colors='k')
ax[0].plot(sample[:, 0], sample[:, 1], ',k', alpha=0.1)
plt.xlabel('slope : m')
plt.ylabel('intercept : b')
plt.title("68% and 95% joint confidence intervals")
plt.show();
```

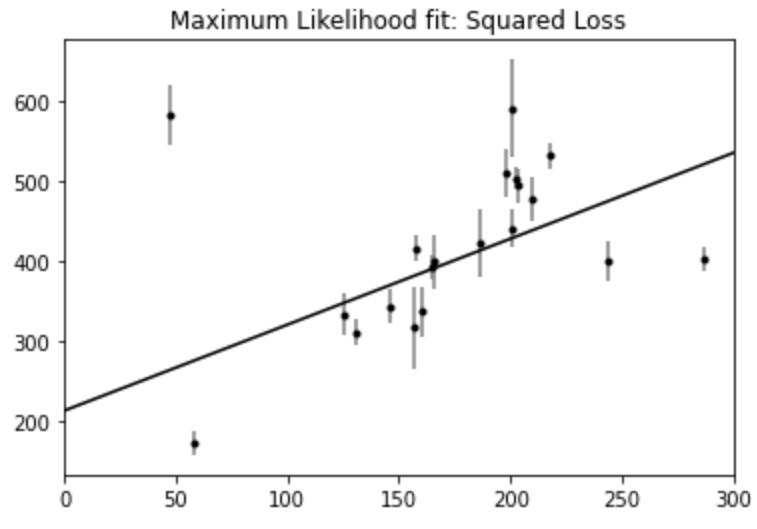


Q3

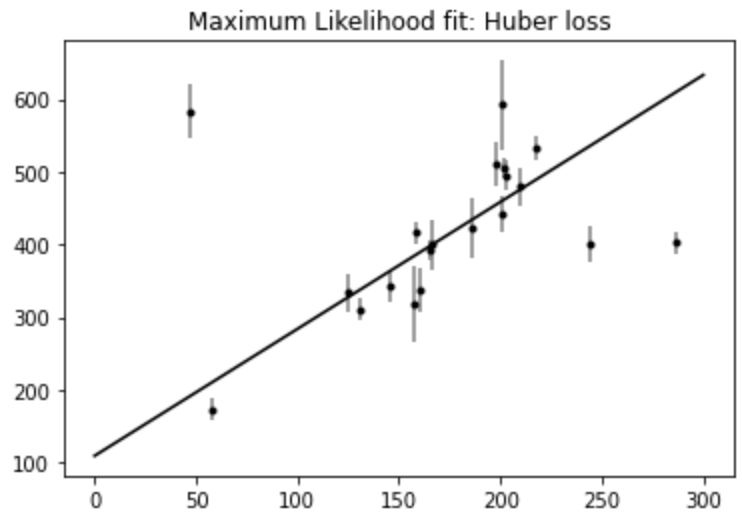
In [48]:
```
x = data[:, 1]
y = data[:, 2]
sigma_y = data[:, 3]

plt.errorbar(x, y, sigma_y, fmt='.k', ecolor='gray');
plt.title("Visualizing given data")
plt.show()
```



In [49]:
```
from scipy import optimize

def squared_loss(theta, x=x, y=y, e=sigma_y):
    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, sigma_y, fmt='.k', ecolor='gray')
plt.plot(xfit, theta1[0] + theta1[1] * xfit, '-k')
plt.title('Maximum Likelihood fit: Squared Loss');
plt.xlim(0, 300)
plt.show();
```



In [50]:
```
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=sigma_y, 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, sigma_y, fmt='.k', ecolor='gray')
plt.plot(xfit, theta2[0] + theta2[1] * xfit, color='black')
plt.title('Maximum Likelihood fit: Huber loss');
```

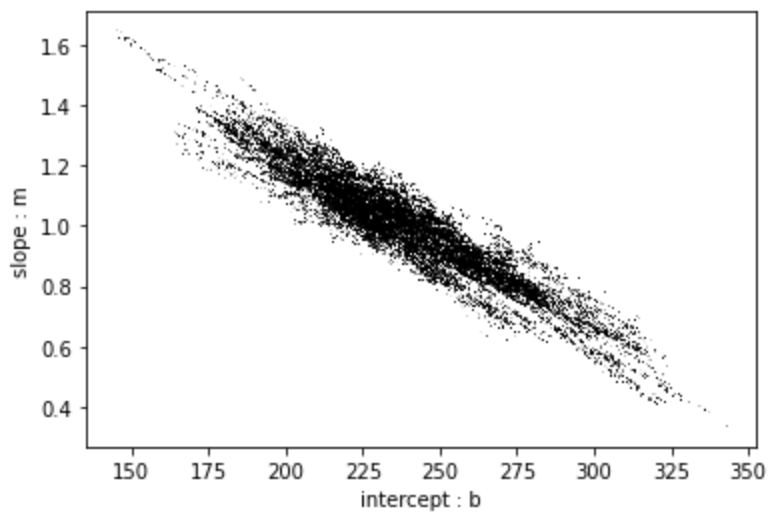


In [51]:
```
def log_prior(theta):
    #g_i needs to be between 0 and 1
    if (all(theta[2:] > 0) and all(theta[2:] < 1)):
        return 0
    else:
        return -np.inf  # recall log(0) = -inf

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
    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)
```

In [52]:
```
# Note that this step will take a few minutes to run!

ndim = 2 + len(x)  # number of parameters in the model
nwalkers = 50  # number of MCMC walkers
nburn = 10000  # "burn-in" period to let chains stabilize
nsteps = 15000  # number of MCMC steps to take

# set theta near the maximum likelihood, with
np.random.seed(0)
starting_guesses = np.zeros((nwalkers, ndim))
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, sigma_y, 50])
sampler.run_mcmc(starting_guesses, nsteps)

sample = sampler.chain  # shape = (nwalkers, nsteps, ndim)
sample = sampler.chain[:, nburn:, :].reshape(-1, ndim)
```

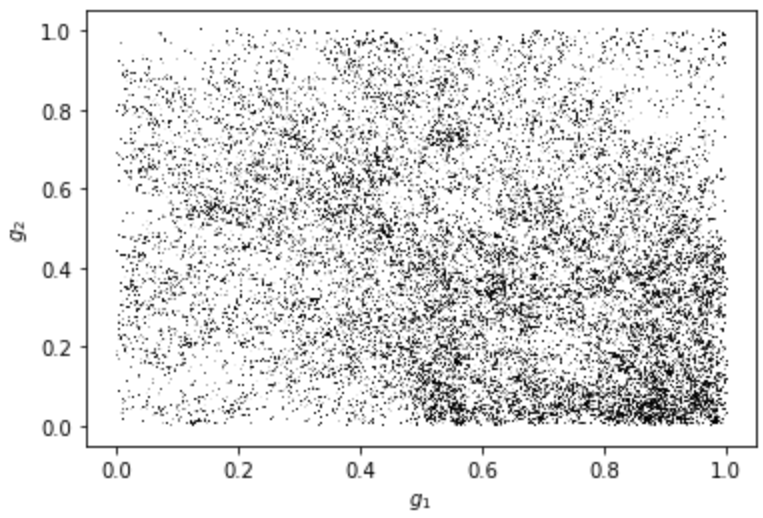
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: RuntimeWarning: divide by zero encountered in log
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: RuntimeWarning: divide by zero encountered in log
  # This is added back by InteractiveShellApp.init_path()
```

In [53]:
```
plt.plot(sample[:, 0], sample[:, 1], ',k', alpha=0.1)
plt.xlabel('intercept : b')
plt.ylabel('slope : m')
plt.show();
```

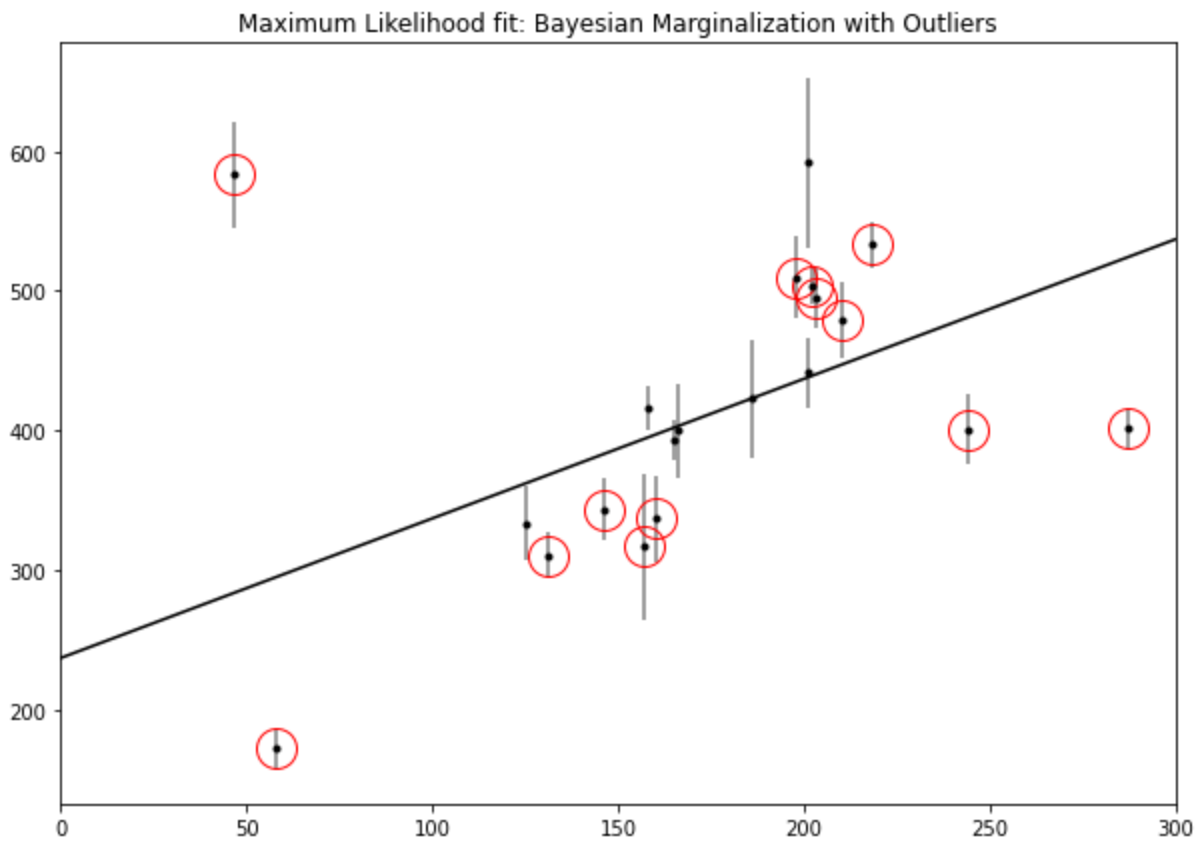


In [54]:
```
plt.plot(sample[:, 2], sample[:, 3], ',k', alpha=0.1)
plt.xlabel('$g_1$')
plt.ylabel('$g_2$')

print("g1 mean: {0:.2f}".format(sample[:, 2].mean()))
print("g2 mean: {0:.2f}".format(sample[:, 3].mean()))
```

```
g1 mean: 0.59
g2 mean: 0.39
```



In [55]:
```
theta3 = np.mean(sample[:, :2], 0)
g = np.mean(sample[:, 2:], 0)
outliers = (g < 0.5)

plt.figure(figsize=(10, 7))
plt.errorbar(x, y, sigma_y, fmt='.k', ecolor='gray')
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 with Outliers')
plt.xlim(0, 300)
plt.show();
```



In [ ]:
```
plt.figure(figsize=(10, 7))
plt.errorbar(x, y, sigma_y, fmt='.k', ecolor='gray')
plt.plot(xfit, theta3[0] + theta3[1] * xfit, color='gray')
plt.plot(xfit, theta3[0] + theta3[1] * xfit, color='gray')
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 with Outliers')
plt.xlim(0, 300)
plt.show();
```

PH6130 Data Science Analysis

Assigment 7

N Tarun - EE19BTECh11050

In [32]:
```
!pip install emcee --quiet
!pip install astroML --quiet
!pip install corner --quiet
!pip install dynesty --quiet
!pip install nestle --quiet
```

In [33]:
```
import numpy as np
import pandas as pd
import corner
import emcee
import matplotlib.pyplot as plt
from scipy.special import erfinv, ndtri
from scipy import stats, optimize
from sklearn.neighbors import KernelDensity
from astroML.density_estimation import KNeighborsDensity
import dynesty
import nestle
from dynesty import plotting as dyplot
```

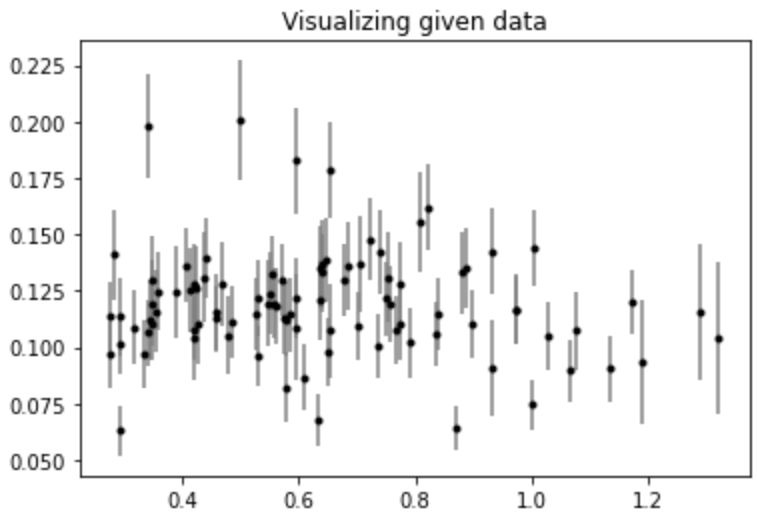
Q1

In [34]:
```
data = pd.read_csv("https://iith.ac.in/~shantanud/fgas_spt.txt", delim_whitespace=True)
display(data)
```

	#z	fgas	fgas_error	ignore
0	0.2777	0.096610	0.014883	0.00000
1	0.2786	0.113973	0.015304	0.00000
2	0.2836	0.141139	0.019905	0.00000
3	0.2950	0.113488	0.017296	0.00000
4	0.2960	0.063003	0.011192	0.00000
...	...	...	...	...
89	1.1320	0.090407	0.014689	0.00000
90	1.1700	0.119809	0.014109	0.09765
91	1.1900	0.093100	0.027400	0.09855
92	1.2880	0.115700	0.030000	0.00000
93	1.3200	0.103800	0.033800	0.00000

94 rows × 4 columns

In [35]:
```
x = data["#z"].to_numpy()
y = data["fgas"].to_numpy()
sigma_y = data["fgas_error"].to_numpy()

plt.errorbar(x, y, sigma_y, fmt='.k', ecolor='gray');
plt.title("Visualizing given data")
plt.show()
```



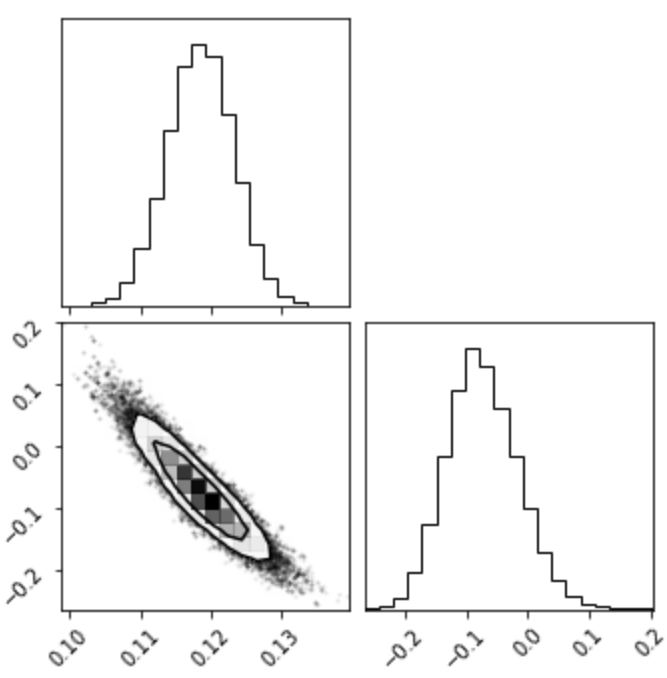
In [36]:
```
# theta[0] = f0, theta[1] = f[1]
def log_prior(theta):
    if(0 < theta[0] and theta[0] < 0.5 and -0.5 < theta[1] and theta[1] < 0.5):
        return 0
    else:
        return -np.inf

# y_hat = f0 * (1 + f1 * z)
def model(x, theta):
    return theta[0] * (1 + theta[1] * x)

def log_likelihood(theta, x, y, e):
    y_hat = model(x, theta)
    return -0.5 * np.sum(np.log(2 * np.pi * e ** 2) + (y - y_hat) ** 2 / e ** 2)

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

In [37]:
```
ndim, nwalkers = 2, 100
nsteps, nburn = 1000, 500

starting_guesses = np.random.rand(nwalkers, ndim)

# start f0 in [0, 0.5]
# and f1 in [-0.5, 0.5]
starting_guesses[:, 0] *= 0.5
starting_guesses[:, 1] -= 0.5

sampler = emcee.EnsembleSampler(nwalkers, ndim, log_posterior, args=[x, y, sigma_y])
sampler.run_mcmc(starting_guesses, nsteps)

sample = sampler.chain  # shape = (nwalkers, nsteps, ndim)
sample = sampler.chain[:, nburn:, :].reshape(-1, 2)
print(sample.shape)
```

```
(50000, 2)
```

In [38]:
```
theta = np.mean(sample[:, :], 0)
print(f"Best fit values of [f0, f1] : {theta}\n\n")
figure = corner.corner(sample, levels=(0.68, 0.9))
```

```
Best fit values of [f0, f1] : [ 0.11846858 -0.07330423]
```

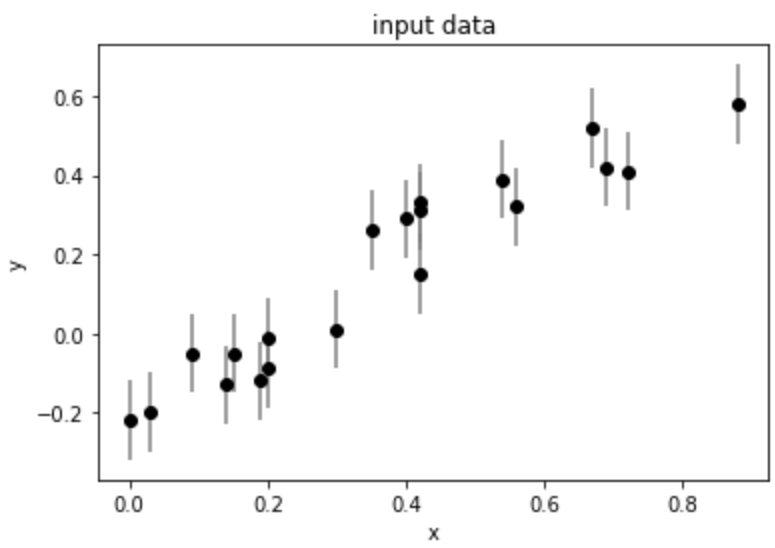


In [39]:
```
def bayes_cr(D, frac):
    """Compute the credible region on the mean"""
    Nsigma = np.sqrt(2) * erfinv(frac)
    mu = np.mean(D)
    sigma = np.std(D)
    sigma_mu = sigma * D.size ** -0.5
    return mu - Nsigma * sigma_mu, mu + Nsigma * sigma_mu

print("90% CR f0 : [{0:f}, {1:f}]".format(*bayes_cr(sample[:, 0], 0.90)))
print("68% CR f0 : [{0:f}, {1:f}]".format(*bayes_cr(sample[:, 0], 0.68)))
print("90% CR f1 : [{0:f}, {1:f}]".format(*bayes_cr(sample[:, 1], 0.90)))
print("68% CR f1 : [{0:f}, {1:f}]".format(*bayes_cr(sample[:, 1], 0.68)))
```

```
90% CR f0 : [0.118434, 0.118504]
68% CR f0 : [0.118447, 0.118490]
90% CR f1 : [-0.073726, -0.072883]
68% CR f1 : [-0.073559, -0.073049]
```

Q2

In [40]:
```
data2 = np.array([[ 0.42,  0.72,  0.  ,  0.3 ,  0.15,
                    0.09,  0.19,  0.35,  0.4 ,  0.54,
                    0.42,  0.69,  0.2 ,  0.88,  0.03,
                    0.67,  0.42,  0.56,  0.14,  0.2  ],
                  [ 0.33,  0.41, -0.22,  0.01, -0.05,
                   -0.05, -0.12,  0.26,  0.29,  0.39,
                    0.31,  0.42, -0.01,  0.58, -0.2 ,
                    0.52,  0.15,  0.32, -0.13, -0.09 ],
                  [ 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 = data2

fig, ax = plt.subplots()
ax.errorbar(x, y, sigma_y, fmt='ok', ecolor='gray')
ax.set(xlabel='x', ylabel='y', title='input data');
```



In [41]:
```
def polynomial_fit(theta, x):
    return sum(t * x ** n for (n, t) in enumerate(theta))

def logL(theta, model=polynomial_fit, data=data2):
    x, y, sigma_y = data
    y_fit = model(theta, x)
    return sum(stats.norm.logpdf(*args)
               for args in zip(y, y_fit, sigma_y))
```

In [42]:
```
def logL_lin(theta):
    y_hat = polynomial_fit(theta, x)
    chisq = np.sum(((y - y_hat) / sigma_y)**2)
    return -chisq/2

def logL_quad(theta):
    y_hat = polynomial_fit(theta, x)
    chisq = np.sum(((y - y_hat) / sigma_y)**2)
    return -chisq/2

def prior_transform(x):
    return 10 * x - 5

lin = nestle.sample(logL_lin, prior_transform, 2)
quad = nestle.sample(logL_quad, prior_transform, 3)

print(f"Bayesian evidence for linear model      : {lin.logz}")
print(f"Bayesian evidence for quadratic model   : {quad.logz}")
```

```
Bayesian evidence for linear model      : -14.773229801528379
Bayesian evidence for quadratic model   : -15.42243190399831
```

From the blog,

> chi2 likelihood
>
> - linear model: 0.0455244340637
> - quadratic model: 0.0362561748938

Hence, the Bayesian evidence values agree with the result obtained in the blog.

Q3

In [43]:
```
!wget https://astrostatistics.psu.edu/datasets/SDSS_quasar.dat --no-check-certificate
```

```
--2022-03-09 05:21:41--  https://astrostatistics.psu.edu/datasets/SDSS_quasar.dat
Resolving astrostatistics.psu.edu (astrostatistics.psu.edu)... 168.62.182.234
Connecting to astrostatistics.psu.edu (astrostatistics.psu.edu)|168.62.182.234|:443... connected.
WARNING: cannot verify astrostatistics.psu.edu's certificate, issued by ‘CN=InCommon RSA Server CA,OU=InCommon,O=Inter
net2,L=Ann Arbor,ST=MI,C=US’:
  Unable to locally verify the issuer's authority.
HTTP request sent, awaiting response... 200 OK
Length: 9469880 (9.0M)
Saving to: ‘SDSS_quasar.dat.1’

SDSS_quasar.dat.1   100%[===================>]   9.03M  15.5MB/s    in 0.6s

2022-03-09 05:21:42 (15.5 MB/s) - ‘SDSS_quasar.dat.1’ saved [9469880/9469880]
```

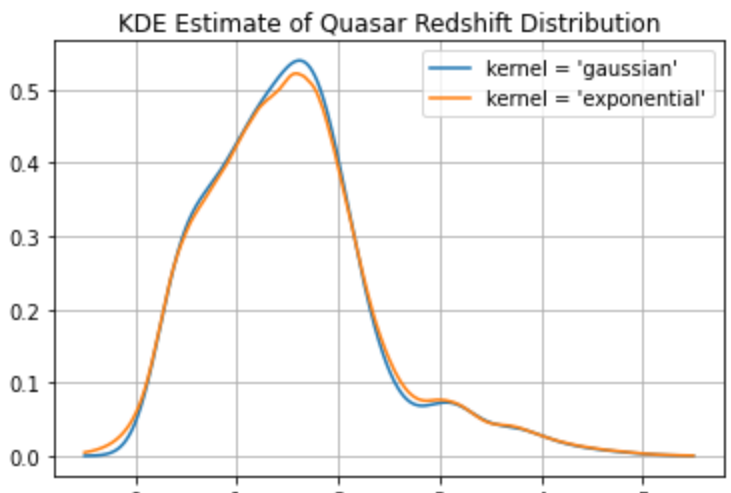
In [44]:
```
data3 = pd.read_csv("SDSS_quasar.dat", delim_whitespace=True)
display(data3)
```

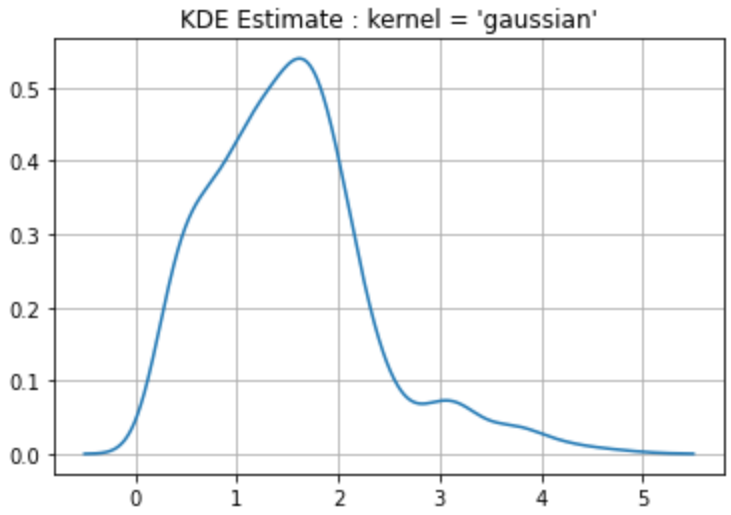
	SDSS_J	R.A.	Dec.	z	u_mag	sig_u	g_mag	sig_g	r_mag	sig_r	...	sig_z	Radio	X-ray	J_mag	sig_J	H
0	000009.26+151754.5	0.038605	15.298476	1.1986	19.921	0.042	19.811	0.036	19.386	0.017	...	0.069	-1.0	-9.000	0.000	0.000	
1	000009.38+135618.4	0.039088	13.938447	2.2400	19.218	0.026	18.893	0.022	18.445	0.018	...	0.033	-1.0	-9.000	0.000	0.000	
2	000009.42-102751.9	0.039269	-10.464428	1.8442	19.249	0.036	19.029	0.027	18.980	0.021	...	0.047	0.0	-9.000	0.000	0.000	
3	000011.41+145545.6	0.047547	14.929353	0.4596	19.637	0.030	19.466	0.024	19.362	0.022	...	0.047	-1.0	-9.000	0.000	0.000	
4	000011.96+000225.3	0.049842	0.040372	0.4790	18.237	0.028	17.971	0.020	18.025	0.019	...	0.029	0.0	-1.660	16.651	0.136	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
46415	235949.46+150430.6	359.956093	15.075185	0.2977	19.094	0.025	18.966	0.023	18.668	0.016	...	0.033	-1.0	-1.429	16.676	0.180	1
46416	235953.44-093655.6	359.972672	-9.615454	0.3585	19.509	0.045	19.276	0.022	18.895	0.018	...	0.039	0.0	-9.000	16.976	0.173	1
46417	235956.72+135131.7	359.986358	13.858825	2.3826	20.010	0.040	19.427	0.027	19.217	0.018	...	0.048	-1.0	-9.000	0.000	0.000	
46418	235958.21+005139.8	359.992546	0.861062	2.0382	19.256	0.034	19.004	0.021	18.794	0.017	...	0.036	0.0	-9.000	0.000	0.000	
46419	235959.06-090944.0	359.996089	-9.162229	1.2845	18.403	0.021	18.373	0.015	18.139	0.024	...	0.036	0.0	-9.000	0.000	0.000	

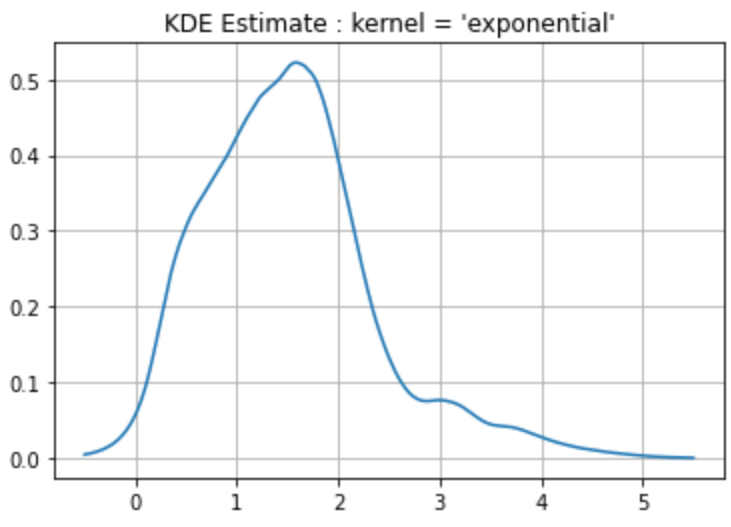
46420 rows × 23 columns

In [45]:
```
z = data3["z"].to_numpy()[:, np.newaxis]
```

In [46]:
```
X_plot = np.linspace(-0.5, 5.5, 1000)[:, np.newaxis]
kernels = ["gaussian", "exponential"]

for kernel in kernels:
    kde = KernelDensity(kernel=kernel, bandwidth=0.2).fit(z)
    log_dens = kde.score_samples(X_plot)
    plt.plot(X_plot[:, 0], np.exp(log_dens), label="kernel = '{0}'".format(kernel))
plt.legend()
plt.grid()
plt.title("KDE Estimate of Quasar Redshift Distribution")
plt.show()

for kernel in kernels:
    kde = KernelDensity(kernel=kernel, bandwidth=0.2).fit(z)
    log_dens = kde.score_samples(X_plot)
    plt.plot(X_plot[:, 0], np.exp(log_dens))
    plt.grid()
    plt.title("KDE Estimate : kernel = '{0}'".format(kernel))
    plt.show()
```







PH6130 Data Science Analysis

Assignment 8

EE19BTECH11050 - N Tarun

In [12]:
```
!pip install astroML --quiet
!apt -qq install cm-super dvipng texlive-latex-extra texlive-latex-recommended
```

```
texlive-latex-recommended is already the newest version (2017.20180305-1).
cm-super is already the newest version (0.3.4-11).
dvipng is already the newest version (1.15-1).
texlive-latex-extra is already the newest version (2017.20180305-2).
0 upgraded, 0 newly installed, 0 to remove and 39 not upgraded.
```

In [13]:
```
import numpy as np
import pandas as pd
import matplotlib
matplotlib.rcParams['text.usetex'] = True # remove if not using LaTex
import matplotlib.pyplot as plt
from astroML.correlation import bootstrap_two_point_angular

from google.colab import data_table
data_table.enable_dataframe_formatter()
```

In [14]:
```
data = pd.read_csv("https://www.iith.ac.in/~shantanud/BCS05hr_reduced.txt", delim_whitespace=True)
data.head()
```

Out[14]:

	#RA	DEC	r-mag	spread_model	spread_model_err
0	76.709724	-56.091484	22.2622	0.113884	0.002812
1	77.430664	-56.090149	23.8355	0.186889	0.003559
2	76.937309	-56.092442	17.7021	0.000614	0.000120
3	77.344833	-56.089947	23.7293	0.117396	0.000751
4	77.416412	-56.089119	23.4456	0.192760	0.004764

In [15]:
```
data = data.rename(columns={"r-mag": "r_mag"})

data = data.query("17 <= r_mag and r_mag <= 20 and spread_model > 0.002")
data.head()
```

Out[15]:

	#RA	DEC	r_mag	spread_model	spread_model_err
16	77.039696	-56.084904	19.9448	0.008856	0.000064
38	77.119270	-56.108150	19.6127	0.006623	-0.000183
43	76.676086	-56.106075	18.8138	0.002451	0.000430
151	77.118393	-56.084389	19.7339	0.009028	0.020733
153	76.823029	-56.082844	19.8468	1.937630	0.019814

In [18]:
```
def compute_results(Nbins=16, Nbootstraps=10,  method='landy-szalay', rseed=0):
    np.random.seed(rseed)
    bins = 10 ** np.linspace(np.log10(1 / 60.), np.log10(1), 16)

    results = [bins]
    for D in [data]:
        results += bootstrap_two_point_angular(D['#RA'],
                                               D['DEC'],
                                               bins=bins,
                                               method=method,
                                               Nbootstraps=Nbootstraps)

    return results

(bins, b_corr, b_corr_err, b_bootstraps) = compute_results()

bin_centers = 0.5 * (bins[1:] + bins[:-1])
```

In [21]:
```
fig = plt.figure(figsize=(8, 5))

ax = fig.add_subplot(111, xscale='log', yscale='log')
ax.errorbar(bin_centers, b_corr, b_corr_err,fmt='.k', ecolor='gray', lw=1)

t = np.array([0.01, 1])
ax.plot(t, 10 * (t / 0.01) ** -0.8, ':k', linewidth=1)

plt.xlabel(r"$\theta\ (deg)$", fontsize=18)
plt.ylabel(r"$\hat{w}(\theta)$", fontsize=18)

plt.show()
```