Q1.

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

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_data, y_data, sigma_y_data = data

# defining linear, quadratic and cubic function
def linear_function(x, a, b):
    # returns a line function with parameters a and b
    return a * x + b

def quadratic_function(x, a, b, c):
    # returns a quadratic function with parameters a, b, and c
    return a * (x**2) + b * x + c

def cubic_function(x, a, b, c, d):
    # returns a cubic function with parameters a, b, c, and d
    return a * (x**3) + b * (x**2) + c * x + d
```

```
# helper function for log likelihood calculation
def log_likelihood(degree, params, data):
    x, y, sigma_y = data
    if degree == 1:
        y_fit = linear_function(x, params[0], params[1])
    elif degree == 2:
        y_fit = quadratic_function(x, params[0], params[1], params[2])
    elif degree == 3:
        y_fit = cubic_function(x, params[0], params[1], params[2],
params[3])
    return sum(stats.norm.logpdf(*args) for args in zip(y, y_fit,
sigma_y))

# functions for finding AIC & BIC
def find_AIC(degree, params, data):
    return -2 * log_likelihood(degree, params, data) + 2 * (degree + 1)

def find_AICc(degree, params, N, data):
    k = degree + 1
    return find_AIC(degree, params, data) + (2 * k * k + 2 * k) / (N - k -
1)

def find_BIC(degree, params, N, data):
    return -2 * log_likelihood(degree, params, data) + (degree + 1) *
np.log(N)

# function for finding error wrt curve fit parameters
def find_error(degree, params, data):
    if degree == 1:
        return (y_data - linear_function(x_data, params[0], params[1])) /
sigma_y_data
    elif degree == 2:
        return (y_data - quadratic_function(x_data, params[0], params[1],
params[2])) / sigma_y_data
    elif degree == 3:
        return (y_data - cubic_function(x_data, params[0], params[1],
params[2], params[3])) / sigma_y_data

# helper function for finding chi-square values
def compute_chi_square(degree, params, data):
    x, y, sigma_y = data
    temp = find_error(degree, params, data)
    return np.sum(temp ** 2)
```

```python
# function for computing degree of freedom
def compute_dof(degree, data):
    return data.shape[1] - (degree + 1)

# function for finding chi-square likelihood
def chi_square_likelihood(degree, params, data):
    chi2 = compute_chi_square(degree, params, data)
    dof = compute_dof(degree, data)
    return stats.chi2(dof).pdf(chi2)

# finding curve fit parameters
linear_params, _ = optimize.curve_fit(linear_function, xdata=x_data,
ydata=y_data, sigma=sigma_y_data)
quadratic_params, _ = optimize.curve_fit(quadratic_function, xdata=x_data,
ydata=y_data, sigma=sigma_y_data)
cubic_params, _ = optimize.curve_fit(cubic_function, xdata=x_data,
ydata=y_data, sigma=sigma_y_data)

# finding error corresponding to each fitting
linear_error = find_error(1, linear_params, data)
quadratic_error = find_error(2, quadratic_params, data)
cubic_error = find_error(3, cubic_params, data)

# finding and printing required values
print(f"Best fit values for Linear model (ax+b) are [a b] =
{linear_params}")
print(f"Best fit values for Quadratic model (ax^2 + bx + c) are [a b c] =
{quadratic_params}")
print(f"Best fit values for Cubic model (ax^3 + bx^2 + cx + d) are [a b c
d] = {cubic_params}\n")

print("chi-square likelihood")
print(f"linear model: {chi_square_likelihood(1, linear_params, data)}")
print(f"quadratic model: {chi_square_likelihood(2, quadratic_params,
data)}")
print(f"cubic model: {chi_square_likelihood(3, cubic_params, data)}")

print(f"AIC Values (Linear Model) = {find_AIC(1, linear_params, data)}")
print(f"AIC Values (Quadratic Model) = {find_AIC(2, quadratic_params,
data)}")
print(f"AIC Values (Cubic Model) = {find_AIC(3, cubic_params, data)}")

print(f"AICc Values (Linear Model) = {find_AICc(1, linear_params,
len(x_data), data)}")
```

```
print(f"AICc Values (Quadratic Model) = {find_AICc(2, quadratic_params,
len(x_data), data)}")
print(f"AICc Values (Cubic Model) = {find_AICc(3, cubic_params,
len(x_data), data)}")

print(f"BIC Values (Linear Model) = {find_BIC(1, linear_params,
len(x_data), data)}")
print(f"BIC Values (Quadratic Model) = {find_BIC(2, quadratic_params,
len(x_data), data)}")
print(f"BIC Values (Cubic Model) = {find_BIC(3, cubic_params, len(x_data),
data)}")
print("Since Linear model has least BIC value, it is most suitable.\n")

# finding and printing p-value of quadratic and cubic model with respect
to linear mode
p_val_quad = 1 - stats.chi2(1).cdf(stats.chi2(len(x_data) -
2).pdf(np.sum(linear_error ** 2)) - stats.chi2(len(x_data) -
3).pdf(np.sum(quadratic_error ** 2)))
p_val_cubic = 1 - stats.chi2(2).cdf(stats.chi2(len(x_data) -
2).pdf(np.sum(linear_error ** 2)) - stats.chi2(len(x_data) -
4).pdf(np.sum(cubic_error ** 2)))
print(f"Quadratic P-value wrt Linear Model is {p_val_quad}")
print(f"Cubic P-value wrt Linear Model is {p_val_cubic}\n")

# declaring points on x-axis
x_axis = np.linspace(0, 1, 1000)

# plot
# fig, ax
fig, ax = plt.subplots()
ax.errorbar(x_data, y_data, sigma_y_data, fmt='ok', ecolor='k',
label='data points')
ax.plot(x_axis, linear_function(x_axis, linear_params[0],
linear_params[1]) , color = 'green', label='best fitted linear model')
ax.plot(x_axis, quadratic_function(x_axis, quadratic_params[0],
quadratic_params[1], quadratic_params[2]), color = 'red',label='best
fitted quadratic model')
ax.plot(x_axis, cubic_function(x_axis, cubic_params[0], cubic_params[1],
cubic_params[2], cubic_params[3]), color = 'blue', label='bestfitted cubic
model')
ax.legend(loc='lower center')
ax.set(xlabel='x values', ylabel='y values ', title='comparison between
models')
plt.grid()
plt.show()
```

```
Best fit values for Linear model (ax+b) are [a b] = [ 2.79789861 -
1.11028082]
Best fit values for Quadratic model (ax^2 + bx + c) are [a b c] = [
0.50261293  2.38475187 -1.05578915]
Best fit values for Cubic model (ax^3 + bx^2 + cx + d) are [a b c d] = [-
0.96724992  1.74451332  1.97184055 -1.02910462]

chi-square likelihood
linear model: 0.045383795585918596
quadratic model: 0.036608447550140304
cubic model: 0.04215280601005994

AIC Values (Linear Model) = -40.0366868160727
AIC Values (Quadratic Model) = -39.84982062400561
AIC Values (Cubic Model) = -38.26081851760256

AICc Values (Linear Model) = -39.33080446313153
AICc Values (Quadratic Model) = -38.34982062400561
AICc Values (Cubic Model) = -35.594151850935894

BIC Values (Linear Model) = -38.04522226896472
BIC Values (Quadratic Model) = -36.86262380334364
BIC Values (Cubic Model) = -34.2778894233866
Since Linear model has least BIC value, it is most suitable.

Quadratic P-value wrt Linear Model is 0.925365878185452
Cubic P-value wrt Linear Model is 0.9983858094213666
```

2.

```python
#importing the library

import numpy as np
from scipy import stats
from scipy import optimize

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

#polynomial function to fit the model with given theta

def polynomial_fit(theta, x):
    """Polynomial model of degree (len(theta) - 1)"""
    return sum(t * x ** n for (n, t) in enumerate(theta))


def logL(theta, model=polynomial_fit, data=data):
    """Gaussian log-likelihood of the model at theta"""
    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))

def best_theta(degree, model=polynomial_fit, data=data):
    theta_0 = (degree + 1) * [0]
    neg_logL = lambda theta: -logL(theta, model, data)
    return optimize.fmin_bfgs(neg_logL, theta_0, disp=False)

max_liklihood_linear=logL(best_theta(1));
max_liklihood_quadratic=logL(best_theta(2));
```

```python
print("linear model:    logL =",max_liklihood_linear )
print("quadratic model: logL =",max_liklihood_quadratic)

#there are 20 datapoints
N=20
# AIC = -2 * log(L) + 2 * k


#for linear number of parameter will be 2
AIC_linear= -2*max_liklihood_linear +2*(2)
#for quadratic number of parameter will be 2
AIC_quadratic= -2*max_liklihood_quadratic +2*(3)

print("AIC for linear model:  =",AIC_linear )
print("AIC for quadratic model:  =",AIC_quadratic)

# BIC = -2 * log(L) + k * log(n)
#for linear number of parameter will be 2
BIC_linear= -2*max_liklihood_linear +np.log(20)*(2)
#for quadratic number of parameter will be 2
BIC_quadratic= -2*max_liklihood_quadratic +np.log(20)*(3)


print("BIC for linear model:  =",BIC_linear )
print("BIC for quadratic model:  =",BIC_quadratic)
```

result:--

linear model: logL = 22.010867006612543

quadratic model: logL = 22.941513586503994

AIC for linear model: = -40.021734013225085

AIC for quadratic model: = -39.88302717300799

BIC for linear model: = -38.030269466117105

BIC for quadratic model: = -36.89583035234602

From Frequentist model these are the result

chi2 likelihood

- linear model: 0.04552443406372872

- quadratic model: 0.036256174893796296

Says linera is more good fit than quadratic model and also our AIC and BIC is also telling same as lower the AIC or BIC high chances of occur. Thus all three are telling that linear model is the best fit.

AIC and BIC are used in Bayesian model selection to compare the relative quality of different models. They consider the trade-off between model complexity and goodness of fit. Frequentist models, on the other hand, rely on p-values and confidence intervals to assess significance. Also AIC or BIC does not works on some particular set of functions.A difference in AIC or BIC of less than 2 is considered trivial, a difference of 2-6 is considered positive evidence, a difference of 6-10 is considered strong evidence, and a difference greater than 10 is considered very strong evidence. Therefore, the difference in AIC and BIC between the linear and quadratic models is trivial, indicating that neither model is strongly preferred over the other.

3.

This paper is really interesting because it uses a statistical test called the KS test to look at leadership quality. The researchers used real data from leaders, employee motivation, job satisfaction, and performance to try and prove their idea. They used a special type of survey along with some other tools to do this. They looked at 45 people who work at the D'Merlion Hotel Batam. They looked at everyone, not just a few people.

What they found was that the way leaders act doesn't really make a big difference in how happy people are with their jobs. But how motivated people are does make a big difference. And how well people do their jobs is really affected by both leadership and motivation.

The paper is applying to the KS test properly. It does not contain the ambiguity like KS test probabilities are wrong if the model was derived from the dataset. EDFs differ in a global fashion near the center of the distribution, The KS test cannot be applied in two or more dimensions as stated by Penn state paper.

Reference:--

Rivaldo, Y. (2021). Leadership and motivation to performance through job satisfaction of hotel employees at D'Merlion Batam. The Winners. Retrieved from journal.binus.ac.id.

4.

```
# Importing the norm and chi2 functions from the scipy.stats module
from scipy.stats import norm, chi2

#source:--Observation of aNew Particleinthe Search forthe Standard Model
Higgs Boson with the ATLAS Detectoratthe LHC
```

```
#This observation,which has a significance of 5.9 standard
deviations,corresponding to abackground fluctuation probability of
1.7×10−9,
# is compatible with the production and decay of the Standard Model
Higgsboson.

p_value = 1.7*10**-9
# Calculating the significance in terms of the number of sigmas for the
Higgs boson using the inverse survival function (isf) of the normal
distribution
sig_value = norm.isf(p_value/2)

# Printing the significance in terms of the number of sigmas for the Higgs
boson
print("Significance value in terms of number of sigmas for higgs boson is
-- ", sig_value)

# Do the same for the LIGO discovery of GW150914, for which the p value =
2 × 10−7
p_value = 2*10**-7

# Calculating the significance in terms of the number of sigmas for LIGO
using the inverse survival function (isf) of the normal distribution
significance_value = norm.isf(p_value/2)
# Printing the significance in terms of the number of sigmas for LIGO
print("Significance value in terms of number of sigmas for LIGO is--",
sig_value)

#source:-Evidence for oscillation of atmospheric neutrinos
# The best-fit to νμ↔ντ oscillations, χ2 min= 65.2 /67 DOF.
chi_square = 65.2
DOF = 67

# Applying formula chi-square goodness of fit (GOF) for the Super-K
discovery
chisquare_gof = 1 - chi2(DOF).cdf(chi_square)

# Value of chi-square GOF for the Super-K discovery
print('chi-square GOF is: ', chisquare_gof)
```

Significance value in terms of number of sigmas for higgs boson is -- 6.024150619389806

Significance value in terms of number of sigmas for LIGO is-- 6.024150619389806 chi-square GOF is: 0.5394901931099038