Data Science Analysis Assignment - 4

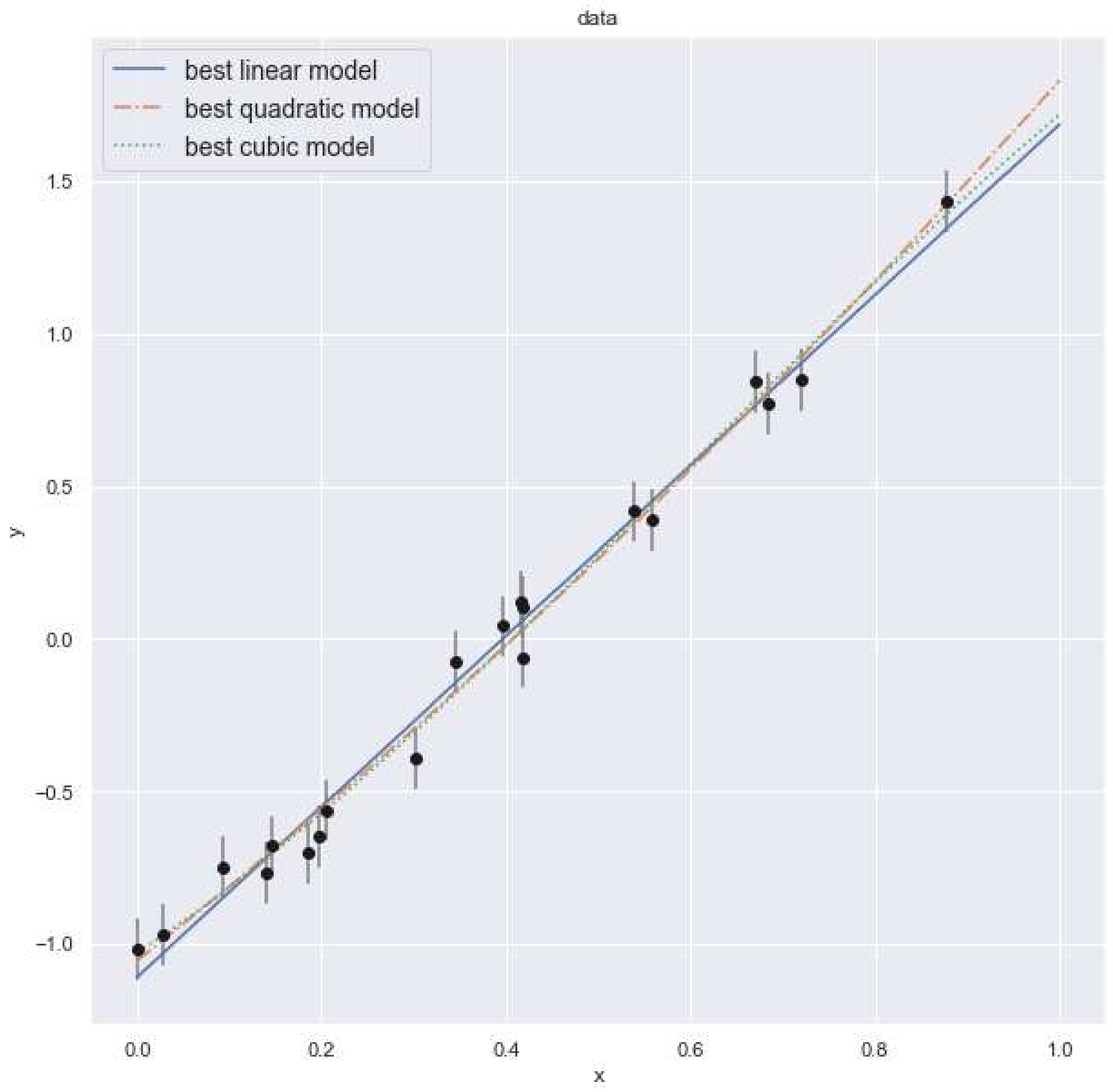
In [1]:
```
#importing required libraries
import pandas as pd

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set() # set default plot styles

from scipy import stats, optimize

import numpy as np

import statsmodels.api as sm
import math
```

Q1.

In [12]:
```
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

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

#function to compute log-likelihood for two models
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)
```

```python
    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)

theta1 = best_theta(1)
theta2 = best_theta(2)
theta3 = best_theta(3)

print("Best-fit values after fitting the data into")
print("  -linear polynomial (a + bx): ",theta1)
print("  -quadratic polynomial (a + bx + cx^2): ",theta2)
print("  -cubic polynomial (a + bx + cx^2 + dx^3): ",theta3)

xfit = np.linspace(0, 1, 1000)

fig, ax = plt.subplots(figsize=(10, 10))

ax.errorbar(x, y, sigma_y, fmt='ok', ecolor='gray')

ax.plot(xfit, polynomial_fit(theta1, xfit), label='best linear model')
ax.plot(xfit, polynomial_fit(theta2, xfit), ls='dashdot', label='best quadratic model')
ax.plot(xfit, polynomial_fit(theta3, xfit), ls='dotted', label='best cubic model')

ax.legend(loc='best', fontsize=14)
ax.set(xlabel='x', ylabel='y', title='data');
```

```
Best-fit values after fitting the data into
  -linear polynomial (a + bx):  [-1.11028083  2.79789863]
  -quadratic polynomial (a + bx + cx^2):  [-1.05578917  2.38475189  0.50261294]
  -cubic polynomial (a + bx + cx^2 + dx^3):  [-1.02910472  1.9718407   1.74451449 -0.96725156]
```



In [3]:

```
#Naive approach: comparing maximum likelihoods
print("linear model:    logL =", logL(best_theta(1)))
print("quadratic model: logL =", logL(best_theta(2)))
print("quadratic model: logL =", logL(best_theta(3)))
```

```
##from the output we can see that quadratic models yields a higher log-likelihood,
#but this doesnt not mean it is a better model.
```

```
linear model:    logL = 22.018343408036255
quadratic model: logL = 22.924910312002737
quadratic model: logL = 23.130409258797258
```

In [4]:

```
#Frequency Model Selection
def compute_chi2(degree, data=data):
    x, y, sigma_y = data
    theta = best_theta(degree, data=data)
    resid = (y - polynomial_fit(theta, x)) / sigma_y
    return np.sum(resid ** 2)

def compute_dof(degree, data=data):
    return data.shape[1] - (degree + 1)

def chi2_likelihood(degree, data=data):
    chi2 = compute_chi2(degree, data)
    dof = compute_dof(degree, data)
    return stats.chi2(dof).pdf(chi2)

print("chi2 likelihood")
print("- linear model:    ", chi2_likelihood(1))
print("- quadratic model: ", chi2_likelihood(2))
print("- cubic model: ", chi2_likelihood(3))
```

```
chi2 likelihood
- linear model:     0.04538379558592037
- quadratic model:  0.036608447550141865
- cubic model:  0.04215280601015174
```

We find that the likelihood of the observed residuals under linear model is higher than the other models. Thus 'linear model' is favoured.

In [5]:

```
#AIC and BIC
model1 = sm.OLS(polynomial_fit(theta1,x), x).fit()
model2 = sm.OLS(polynomial_fit(theta2,x), x).fit()
model3 = sm.OLS(polynomial_fit(theta3,x), x).fit()

print("AIC of")
print(" -linear model:",model1.aic)
print(" -quadratic model:",model2.aic)
print(" -cubic model:",model3.aic)
print("\nBIC of")
print(" -linear model:",model1.bic)
print(" -quadratic model:",model2.bic)
print(" -cubic model:",model3.bic)
```

```
AIC of
 -linear model: 39.04981797236597
 -quadratic model: 39.098330098016405
 -cubic model: 39.109311106076944

BIC of
 -linear model: 40.04555024591996
 -quadratic model: 40.094062371570395
 -cubic model: 40.105043379630935
```

Since, the first model (linear model) has lower AIC and BIC value, it is the better fitting model, which agrees with the frequentist model result.

In [6]:
```
#p-value
v = np.linspace(0, 5, 1000)

#as linear model is the prefered model, we compare with quadratic model
chi2_diff = compute_chi2(1) - compute_chi2(2)
chi2_dist = stats.chi2(2).pdf(v)
p_value = 1 - stats.chi2(1).cdf(chi2_diff)

print("p value corresponding to the preferred model is ",p_value)
```

```
p value corresponding to the preferred model is  0.178132756953164
```

Q2.

In [14]:
```
#inserting data of JVDP's blog
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

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):
    """Gaussian log-likelihood of the model at theta"""
    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):
    theta_0 = (degree + 1) * [0]
    neg_logL = lambda theta: -logL(theta, model)
    return optimize.fmin_bfgs(neg_logL, theta_0, disp=False)

# Now we can calculate AIC and BIC as follows:
def compute_AIC(degree):
    k = (degree+1)
    AIC = -2*logL(best_theta(degree)) + 2*k
    return AIC

def compute_BIC(degree):
    k = (degree+1)
    BIC = (-2*logL(best_theta(degree)) + k*math.log(20))
    return BIC

print('AIC values')
print("- linear model: ", compute_AIC(1))
print("- quadratic model: ", compute_AIC(2))
print("- cubic model: ", compute_AIC(3))

print('\nBIC values')
print("- linear model: ", compute_BIC(1))
print("- quadratic model: ", compute_BIC(2))
print("- cubic model: ", compute_BIC(3))

# AIC strength of evidence test:
AIClist = [compute_AIC(1),compute_AIC(2),compute_AIC(3)]
AIC_min = min(AIClist)
```

```python
def delta_AIC(degree):
    return AIClist[degree-1]-AIC_min

print('\nDelta AIC values')
print("- linear model: ", delta_AIC(1))
print("- quadratic model: ", delta_AIC(2))
print("- cubic model: ", delta_AIC(3))

# BIC strength of evidence test:
BIClist = [compute_BIC(1),compute_BIC(2),compute_BIC(3)]
BIC_min = min(BIClist)
def delta_BIC(degree):
    return BIClist[degree-1]-BIC_min

print('\nDelta BIC values')
print("- linear model: ", delta_BIC(1))
print("- quadratic model: ", delta_BIC(2))
print("- cubic model: ", delta_BIC(3))
```

```
AIC values
- linear model:  -40.021734013225085
- quadratic model:  -39.883027173008
- cubic model:  -38.352254021188415

BIC values
- linear model:  -38.030269466117105
- quadratic model:  -36.89583035234603
- cubic model:  -34.36932492697245

Delta AIC values
- linear model:  0.0
- quadratic model:  0.13870684021708257
- cubic model:  1.6694799920366705

Delta BIC values
- linear model:  0.0
- quadratic model:  1.134439113771073
- cubic model:  3.6609445391446513
```

The AIC and BIC suggest that linear model fits the data best which is same as frequentist model comparison results.Thus the obtained results agree with the result shown on the blog.

From the strength of evidence tests for AIC/BIC, we find that the quadratic model is also a fairly good fit.

Q3.

Paper used for reference: https://www.researchgate.net/publication/225692214_Kolmogorov-Smirnov_test_for_spatially_correlated_data

The Kolmogorov-Smirnov test is based on the maximum difference between empirical and a hypothetical cumulative distribution. This test is used as a test of goodness of fit and is ideal when the size of the sample is small. The null hypothesis assumes no difference between the observed and theoretical distribution and the value of test statistic is calculated. Its critical value is found from the K-S table values for one sample test.

When instead of one, there are two independent samples then K-S two sample test can be used to test agreement between the two cumulative distributions. The null hypothesis states that there is no difference between the two distributions.

This non-parametric tests helps a researcher to test the significance of one results when the characteristic of the target population are unknown or no assumptions are made about them.

This test is a convenient metjos for investigating whether two underlying univariate distributions can be regarged as undistinguishable from each other or whether an underlying distribution differes from a hypothesized distribution. Sample needs to be unbiased ans the outcomes need to be independednt and identically distributed for this test to be applied. A generalised form of the bootstrap methos is used here for the purpose of modeling the distribution. The generalization consists of preparing resamplings with the same spatial correlation as the empirical sample.

The generalization is accomplished by reading the value of unconditional stochastic realizations at the sampling locations, realizations that are simulated annealing. This new approachn was tested by two empirical samples taken from an exhaustive sample closely following a lognormal distribution. One sample was a regular, unbiased sample while the other one was a clustered, preferential sample that had to be preprocessed.

The paper's results show that the p-value of the statistic in the absence of spatical correlation, which is in agreement with the fact that the information content of an uncorrelated sample is larger than the one for a spatially correlated sample of the same size.

Checking if the KS-test has been correctly applied: The empirical data has been obtained from experiments, and they have not derived the model from the dataset. The KS-test has been applied only to a one-dimensional dataset.

Therefore, the KS-test has been correctly applied in this paper

Q4.

In [16]:

```
p_value = 1.7*math.pow(10,-9)
print('No. of sigmas = ', stats.norm.isf(p_value))
```

```
No. of sigmas =  5.911017938341624
```

In [17]:

```
LIGO_p_value = 2*math.pow(10,-7)
print('No. of sigmas = ',stats.norm.isf(LIGO_p_value))
```

```
No. of sigmas =  5.068957749717791
```

In [18]:

```
DOF = 67
chi2 = 65.2

print('Chi square GOF : ',(1 - stats.chi2(DOF).cdf(chi2)))
```

```
Chi square GOF :  0.5394901931099038
```