7144_CourseWork

December 1, 2023

1 Task 1: Population Dynamics in Ecosystems

1.1 1 (a):

To Model this dynamical system using matrix equations by writing the population of rabbits and foxes at the current time step in a 2 × 1 vector 'p',

[1]:
```
##########################################################
#INDIVIDUAL TASK 1a. Population Dynamics in Ecosystems
##########################################################

#Let p1 be the population of rabbits.
#    p2 be the population of Foxes.

#Assume,
p1 = 0.7
p2 = 0.3

#Total population accounts to 1. ie., p1 + p2 = 1

import numpy as np

p = np.array([[p1],[p2]]) #p in 2 x 1 matrix
print('p:\n', p)
```

```
p:
 [[0.7]
 [0.3]]
```

Then, The population change is expressed as,

New Rabbit Population = p1 + 0.05*p1* - *0.05*p2 = 1.05*p1* - *0.05*p2 and

New Fox Population = p2 + 0.05*p1* - *0.08*p2 = 0.05*p1* + *0.92*p2

'A' is the matrix which encapsulates these population changes,

A = [[New Rabbit Population]

[New Fox Population]]

[2]:
```
A = np.array([[1.05,-0.05],[0.05,0.92]]) #A in 2 x 2 matrix
print('A:\n',A)
```

```
A:
 [[ 1.05 -0.05]
 [ 0.05  0.92]]
```

[3]:
```
#So the next time step change in population would be
Next_p = A @ p
print('Next time step change:\n',Next_p)
```

```
Next time step change:
 [[0.72 ]
 [0.311]]
```

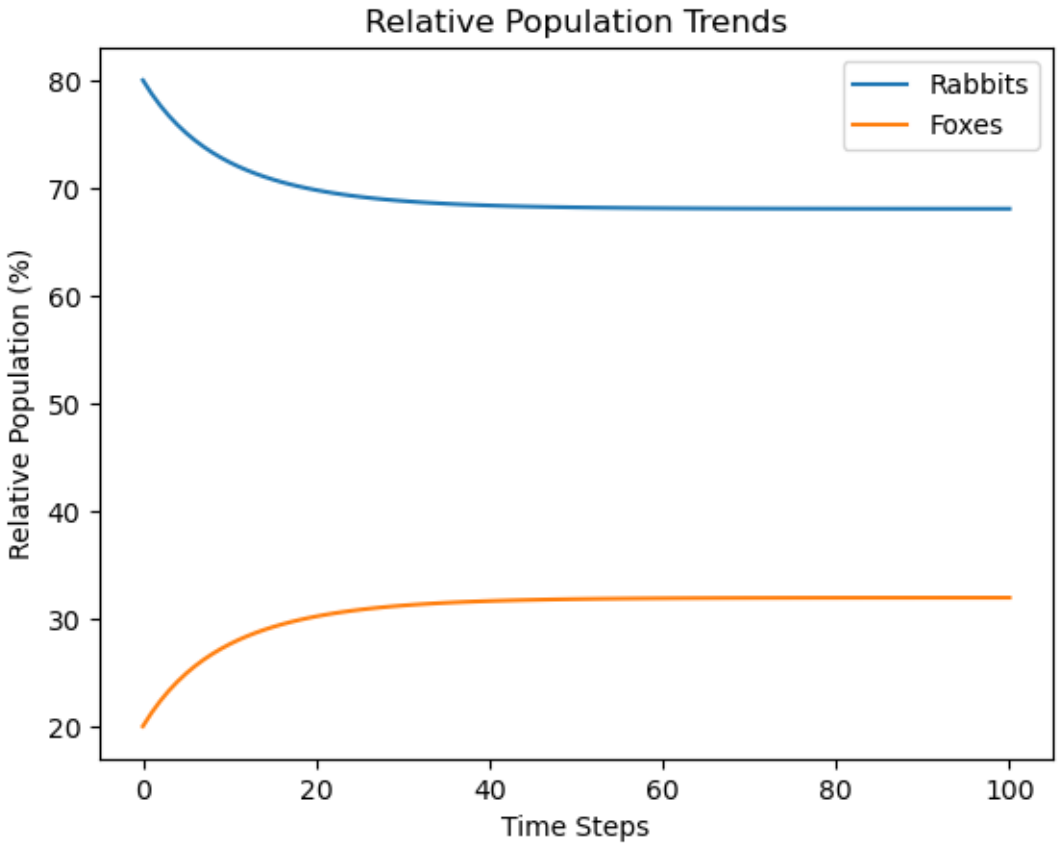
Here, 0.72 is the New Population on Rabbits and 0.311 is the New Population on Foxes in the next time step.

1.2 1 (b):

[4]:
```
p1 = 8000
p2 = 2000
steps = 100

p = np.array([[p1],[p2]])
print('p:\nRabbits:', p[0,0],'\nFoxes:', p[1,0],'\n')

#iterating 100 time steps for the expected population
New_p = p
for i in range(steps):
    New_p = A @ New_p

print('100th time step population:\nRabbits:', round(New_p[0,0]),'\nFoxes:',␣
 ↪round(New_p[1,0]),'\n')

#importing relevant libraries
import matplotlib.pyplot as plt

#Initializing Population over time matrix
Over_time = np.zeros((steps + 1, 2))
step = np.arange(steps + 1)
Over_time[0,:] = p.T
New_p = p

#Storing population data
for i in range (1,steps + 1):
    New_p = A @ New_p
    Over_time[i,:] = New_p.T
```

```
#Relative population Calculation
relativePopulation = Over_time / np.sum(Over_time, axis = 1, keepdims=True) *␣
 ↪100

#Relative populations over time plot
plt.figure()
plt.plot(step, relativePopulation[:, 0], label='Rabbits')
plt.plot(step, relativePopulation[:, 1], label='Foxes')
plt.legend()
plt.title('Relative Population Trends')
plt.xlabel('Time Steps')
plt.ylabel('Relative Population (%)')
plt.show()
```

```
p:
Rabbits: 8000
Foxes: 2000

100th time step population:
Rabbits: 124232
Foxes: 58301
```



The long-term distribution of fox and rabbit populations shows a steady trend of convergence towards an equilibrium state. The simulation shows how the rabbit population dominated over time, with an initial distribution of 80% rabbits and 20% foxes. The relative numbers stabilise in the face of possible changes, suggesting that the two species may cohabit in the ecosystem in a sustainable and balanced manner. This stability is consistent with the dynamics outlined in the matrix, wherein the benefits of fox predation appear to be outweighed by rabbits' faster rate of reproduction. The populations eventually settled into a stable and harmonic balance, demonstrating the endurance and stability of the simulated environment over time. This overall behaviour points to a self-regulating system.

1.3 1 (c):

[5]:
```
#finding Eigen values
eig_values,eig_vectors = np.linalg.eig(A)
print('eigen values:', eig_values)
```

```
eigen values: [1.02653312 0.94346688]
```

The associated population-possibly the rabbit population in this instance-tends to stabilise around an equilibrium value as a result of the dynamics established in the system, according to the eigenvalue 1.0265, which is closer to 1.

A different tendency in the behaviour compared to the other species may be indicated by the eigenvalue 0.9435 (even further from 1), which may indicate a relatively less stable behaviour or influence on the associated population (perhaps the fox population).

These eigenvalues suggest that, in this habitat, the rabbit population may exhibit longer-term behavioural stability than the fox population. The dynamics of both populations are impacted by the interactions shown in matrix A, though, and their stability characteristics across time are reflected in the eigenvalues.

When an eigenvalue approaches 1, it suggests that the factor's impact on population dynamics is more steady. The eigenvalue in this instance, which is closer to 1 (about 1.0265), indicates the more dominant factor influencing population dynamics over the long term.

1.4 1 (d):

The population change is expressed as,

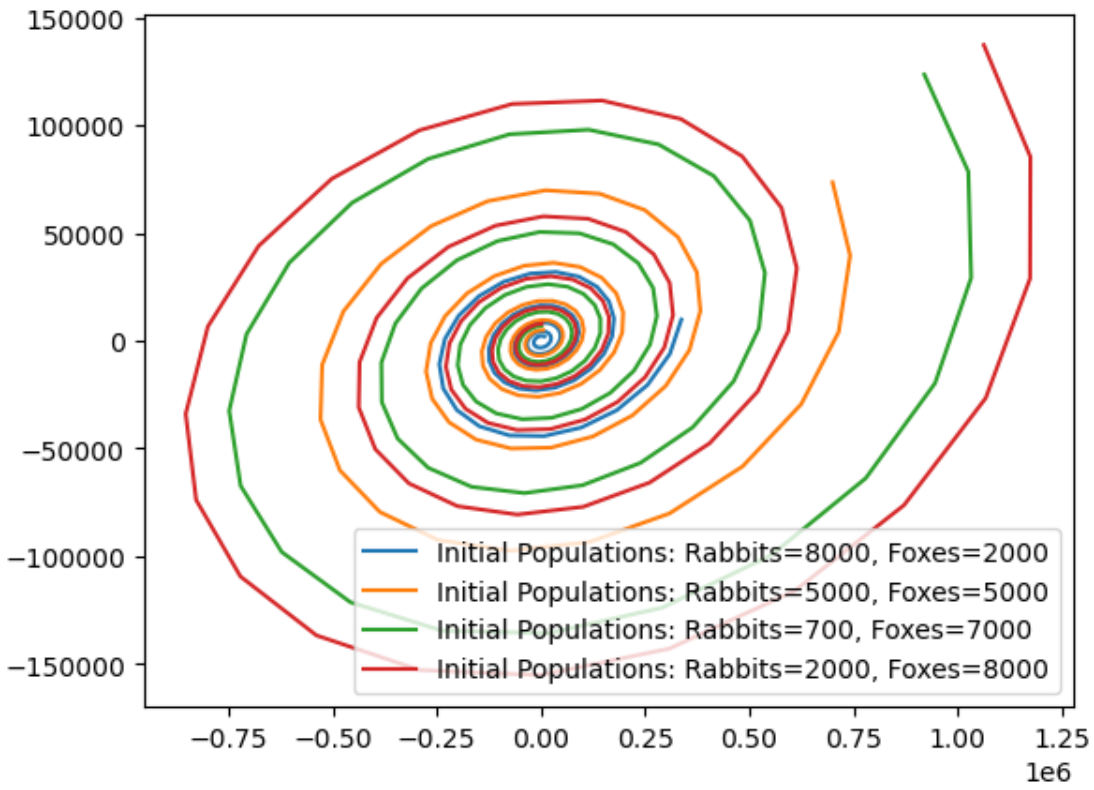
New Rabbit Population = p1 + 0.05$p1$ - 2p2 = 1.05$p1$ - 2p2 and

New Fox Population = p2 + 0.05$p1$ - 0.08p2 = 0.05$p1$ + 0.92p2

[6]:
```
New_A = np.array([[1.05,-2],[0.05,0.92]]) #New A 2 x 2 matrix
print('Modified A:\n',New_A)
```

```
Modified A:
 [[ 1.05 -2.  ]
 [ 0.05  0.92]]
```

[7]:
```
p_list = np.array([[8000,2000],[5000,5000],[700,7000],[2000,8000]]) #Different␣
 ↪types of Initial population
steps = 100

#Storing and plotting the population change for all different type of initial␣
 ↪population
plt.figure()
for j in range (0, p_list.shape[0]):
    newOver_time = np.zeros((steps + 1, 2))
    newOver_time[0,:] = p_list[j]
    Newer_p = p_list[j,:]
    label = "Initial Populations: Rabbits={}, Foxes={}".format(p_list[j,0],␣
 ↪p_list[j,1])
    for i in range(1,steps + 1):
        Newer_p = New_A @ Newer_p
        newOver_time[i,:] = Newer_p

    plt.plot(newOver_time[:,0], newOver_time[:,1], label = label)

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



Derived from the modified A matrix with a 200% decline in the rabbit population owing to predation, the population dynamics matrix raises significant questions and uncovers interesting patterns. At first, there are noticeable shifts in both the rabbit and fox populations, with the fox population growing while the rabbit population sharply declines. However, a deeper look reveals important problems with the model. In the long term, the trajectories imply unstable behaviour and the possibility of limitless development or decrease. Notably, the model's validity is called into question by the occurrence of negative rabbit populations at specific points, which adds a biologically impossible layer. The fact that the populations of foxes and rabbits have diverged over time highlights the lack of a stable equilibrium and points to ongoing expansion or decline. The model evaluation emphasises that in order to guarantee physiologically believable interactions, parameters must be carefully considered. Limitations are introduced by assuming constant percentages for population fluctuations, which calls for parameter adjustment to provide a more realistic picture of ecological dynamics. To sum up, the behaviour that has been observed demands a critical reevaluation of the model, offering opportunities for improvement, different techniques, or more complexity in order to properly represent the complexities of population dynamics in ecological systems.

2 Task 2: Regression by Matrix Operations

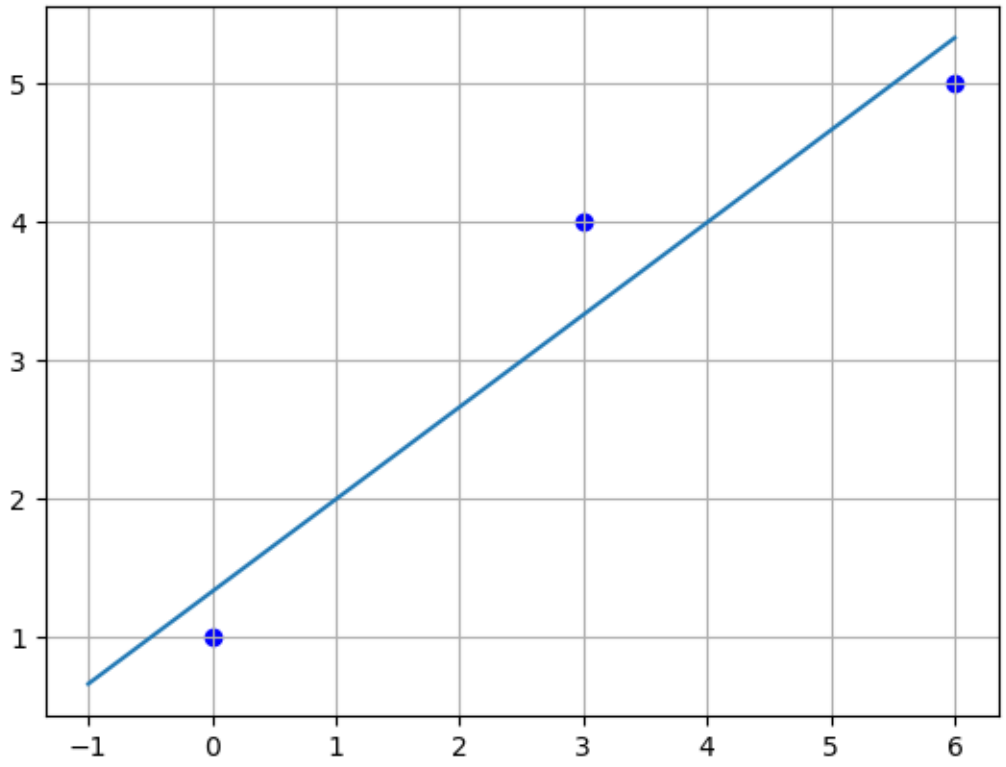
2.1 2 (a):

[8]:
```
import numpy as np
import matplotlib.pyplot as plt

def Regression(points):
    # Create the design matrix X and the response vector y
    X = np.ones((len(points), 2))
    y = np.zeros((len(points), 1))

    # Fill in the values of X and y from the input points
    for i in range(0, len(points)):
        X[i, 1] = points[i][0]
        y[i, 0] = points[i][1]

    # Display the design matrix X and response vector y
    print('X:\n', X)
    print('y:\n', y)

    # Calculate the product of the transpose of X and X (P matrix)
    P = X.T @ X

    # Calculate the inverse of the P matrix (Q matrix)
    Q = np.linalg.inv(P)

    # Calculate the product of the transpose of X and y (R matrix)
    R = X.T @ y

    # Calculate the regression coefficients  using the normal equation
```

```
      = Q @ R
    print(' :\n',  )

    # Display the predicted values X
    print('X :\n', X @  )

    return

# Sample data points
points = [[0, 1], [3, 4], [6, 5]]
  = Regression(points)

# Plot the data points and the regression line
plt.figure()
x = [-1, 0, 2, 3, 6]
Y = []
for i in range(0, len(points)):
    plt.scatter(points[i][0], points[i][1], color='blue')

for j in range(0, len(x)):
    g =  [0, 0] +  [1, 0] * x[j]
    Y.append(g)
plt.plot(x, Y)
plt.grid()
plt.show()

# Calculate R-squared value
Actual_value = np.array([])
Response_value = np.array([])

for i in range(len(points)):
    Actual_value = np.append(Actual_value, points[i][1])

Predictor = [0, 3, 6]
for j in range(len(Predictor)):
    Response_value = np.append(Response_value,  [0, 0] +  [1, 0] * Predictor[j])

# Calculate mean of actual values
mean = np.mean(Actual_value)

# Calculate R-squared using the formula
R_squared = 1 - ((np.sum((Actual_value - Response_value)**2)) / (np.
  ↪sum((Actual_value - mean)**2)))

print('R_Squared:', R_squared)
```

```
X:
 [[1. 0.]
```

```
 [1. 3.]
 [1. 6.]]
y:
 [[1.]
 [4.]
 [5.]]
 :
 [[1.33333333]
 [0.66666667]]
X :
 [[1.33333333]
 [3.33333333]
 [5.33333333]]
```




```
R_Squared: 0.9230769230769231
```

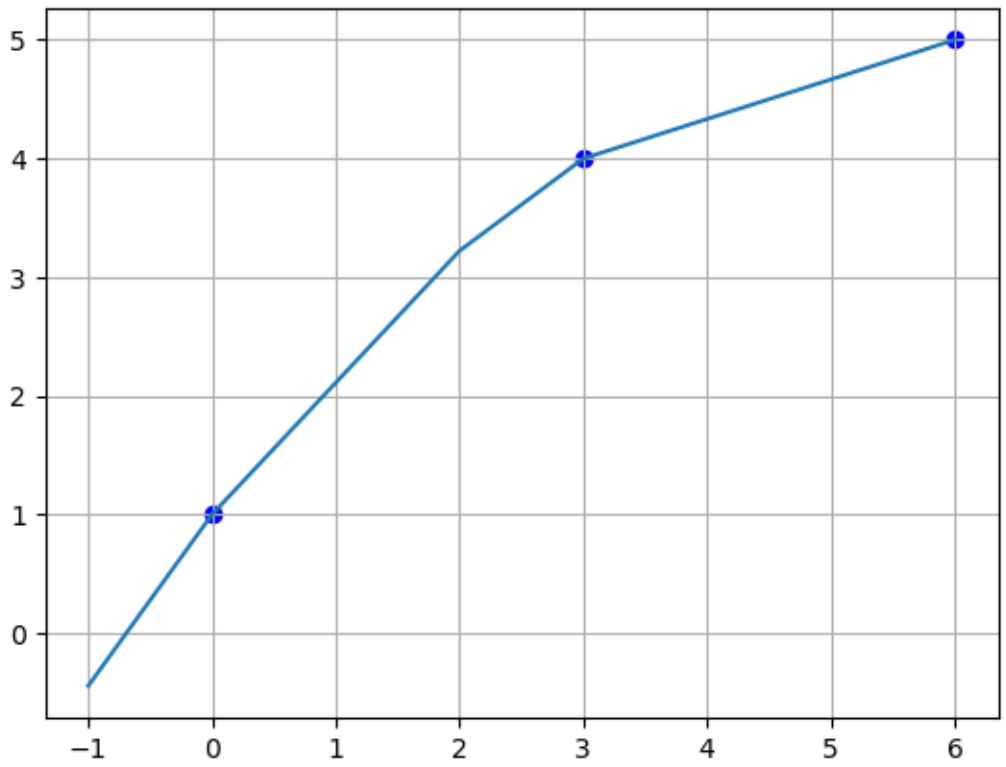
2.2 2 (b):

[9]:
```
import numpy as np
import matplotlib.pyplot as plt

def Regression(points):
```

```
    # Create the design matrix X with columns for the constant term, linear␣
 ↪term, and quadratic term
    X = np.ones((len(points), 3))
    y = np.zeros((len(points), 1))

    # Fill in the values of X and y from the input points
    for i in range(0, len(points)):
        X[i, 1] = points[i][0]  # Linear term
        X[i, 2] = (points[i][0])**2  # Quadratic term
        y[i, 0] = points[i][1]

    # Display the design matrix X and response vector y
    print('X:\n', X)
    print('y:\n', y)

    # Calculate the product of the transpose of X and X (P matrix)
    P = X.T @ X

    # Calculate the inverse of the P matrix (Q matrix)
    Q = np.linalg.inv(P)

    # Calculate the product of the transpose of X and y (R matrix)
    R = X.T @ y

    # Calculate the regression coefficients  using the normal equation
      = Q @ R
    print(' :\n',  )

    # Display the predicted values X
    print('X :\n', X @  )

    return

# Sample data points
points = [[0, 1], [3, 4], [6, 5]]
  = Regression(points)

# Plot the data points and the regression curve
plt.figure()
x = [-1, 0, 2, 3, 6]
Y = []
for i in range(0, len(points)):
    plt.scatter(points[i][0], points[i][1], color='blue')

for j in range(0, len(x)):
    g =  [0, 0] +  [1, 0] * x[j] +  [2, 0] * x[j]**2
    Y.append(g)
```

```
plt.plot(x, Y)
plt.grid()
plt.show()

# Calculate R-squared value
Actual_value = np.array([])
Response_value = np.array([])

for i in range(0, len(points)):
    Actual_value = np.append(Actual_value, points[i][1])

Predictor = [0, 3, 6]
for j in range(0, len(Predictor)):
    Response_value = np.append(Response_value,  [0, 0] +  [1, 0] * Predictor[j]␣
 ↪+  [2, 0] * Predictor[j]**2)

# Calculate mean of actual values
mean = np.mean(Actual_value)

# Calculate R-squared using the formula
R_squared = 1 - ((np.sum((Actual_value - Response_value)**2)) / (np.
 ↪sum((Actual_value - mean)**2)))

print('R_Squared:', R_squared)
```

```
X:
 [[ 1.  0.  0.]
 [ 1.  3.  9.]
 [ 1.  6. 36.]]
y:
 [[1.]
 [4.]
 [5.]]
:
 [[ 1.        ]
 [ 1.33333333]
 [-0.11111111]]
X :
 [[1.]
 [4.]
 [5.]]
```




```
R_Squared: 1.0
```

2.3 2 (c):

[10]:
```
# QUADRATIC MODEL

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

def Regression():
    # Load dataset from CSV file
    dataset = np.array(pd.read_csv('Daylength.csv')) # Day length in this␣
 ↪dataset is converted to minutes
    print(dataset)

    # Initialize design matrix X and response vector y
    X = np.ones((dataset.shape[0], 3))
    y = np.zeros((dataset.shape[0], 1))

    # Fill in the values of X and y from the dataset
    for i in range(0, dataset.shape[0]):
```

```
        X[i, 1] = dataset[i][0]  # Linear term
        X[i, 2] = (dataset[i][0])**2  # Quadratic term
        y[i, 0] = dataset[i][2]  # Response variable

    # Display the design matrix X and response vector y
    print('X:\n', X)
    print('y:\n', y)

    # Calculate the product of the transpose of X and X (P matrix)
    P = X.T @ X

    # Calculate the inverse of the P matrix (Q matrix)
    Q = np.linalg.inv(P)

    # Calculate the product of the transpose of X and y (R matrix)
    R = X.T @ y

    # Calculate the regression coefficients  using the normal equation
      = Q @ R
    print(' :\n',  )

    # Display the predicted values X
    print('X :\n', X @  )

    return

# Perform quadratic regression
  = Regression()

# Load dataset again for plotting and evaluation
dataset = np.array(pd.read_csv('Daylength.csv'))

print('\nQuadratic Model\n')

# Plot the data points and the quadratic regression curve
plt.figure()
Q_x = np.array(dataset[:, 0])
Q_Y = np.array([])
for i in range(0, dataset.shape[0]):
    plt.scatter(dataset[i][0], dataset[i][2], color='blue')

for j in range(0, len(Q_x)):
    Q_Y = np.append(Q_Y,  [0, 0] +  [1, 0] * Q_x[j] +  [2, 0] * Q_x[j]**2)

plt.plot(Q_x, Q_Y)
plt.xlabel('Predictor (Day)')
plt.ylabel('Response (Day Length)')
```

```
plt.grid()
plt.show()

# Evaluate R-squared value for quadratic regression
Q_Actual_value = np.array(dataset[:, 2])
Q_Response_value = np.array([])

Q_Predictor = np.array(dataset[:, 0])
for j in range(0, Q_Predictor.shape[0]):
    Q_Response_value = np.append(Q_Response_value,  [0, 0] +  [1, 0] *␣
 ↪Q_Predictor[j] +  [2, 0] * Q_Predictor[j]**2)

Q_mean = np.mean(Q_Actual_value)

Q_R_squared = 1 - ((np.sum((Q_Actual_value - Q_Response_value)**2)) / (np.
 ↪sum((Q_Actual_value - Q_mean)**2)))

print('R_Squared:', Q_R_squared)
```

```
[[1 '01-01-2022' 471.2833333]
 [2 '02-01-2022' 472.55]
 [32 '01-02-2022' 471.2833333]
 [33 '02-02-2022' 472.55]
 [60 '01-03-2022' 656.1833333]
 [61 '02-03-2022' 660.25]
 [91 '01-04-2022' 783.05]
 [92 '02-04-2022' 787.1333333]
 [121 '01-05-2022' 900.8833333]
 [122 '02-05-2022' 904.5333333]
 [152 '01-06-2022' 992.9833333]
 [153 '02-06-2022' 994.9166667]
 [182 '01-07-2022' 1007.933333]
 [183 '02-07-2022' 1006.816667]
 [213 '01-08-2022' 935.7333333]
 [214 '02-08-2022' 932.4333333]
 [244 '01-09-2022' 820.8666667]
 [245 '02-09-2022' 816.9166667]
 [274 '01-10-2022' 700.1666667]
 [275 '02-10-2022' 696.1333333]
 [305 '01-11-2022' 578.5833333]
 [306 '02-11-2022' 574.9166667]
 [335 '01-12-2022' 487.1166667]
 [336 '02-12-2022' 485.05]
 [366 '01-01-2023' 471.0166667]
 [367 '02-01-2023' 472.2166667]
 [397 '01-02-2023' 548.1166667]
 [398 '02-02-2023' 551.5833333]
```

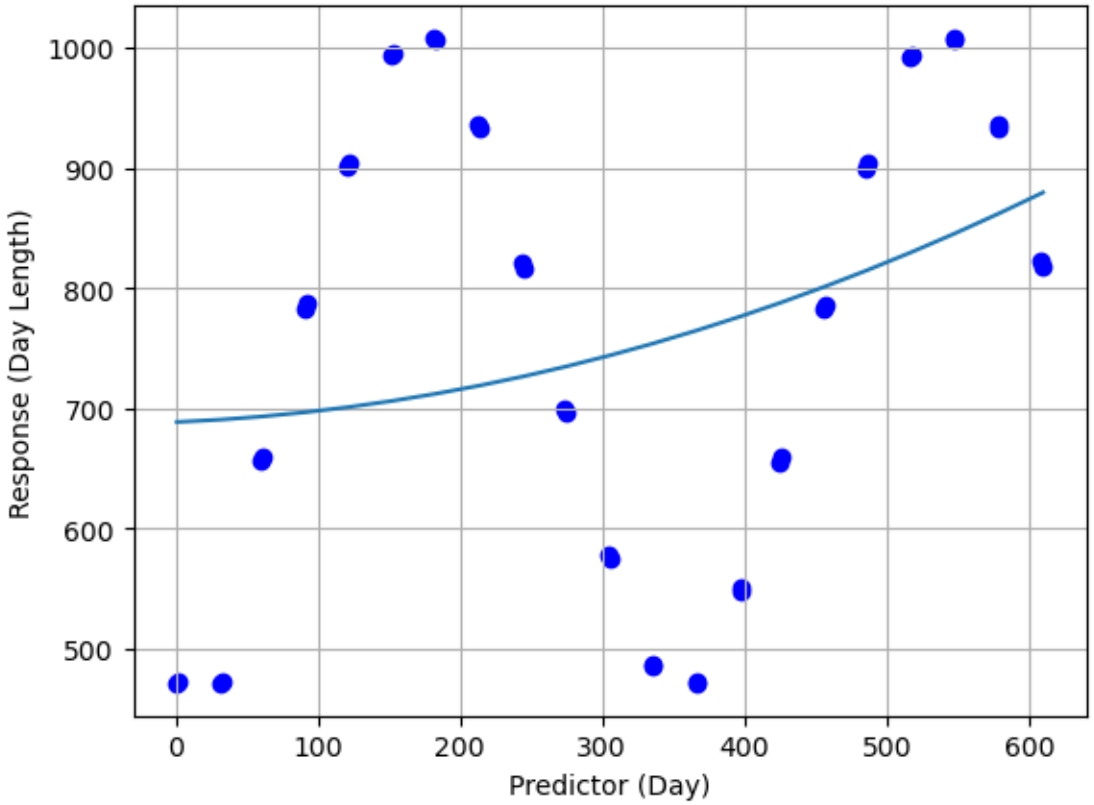
```
 [425 '01-03-2023' 655.2]
 [426 '02-03-2023' 659.25]
 [456 '01-04-2023' 782.0833333]
 [457 '02-04-2023' 786.15]
 [486 '01-05-2023' 899.9833333]
 [487 '02-05-2023' 903.65]
 [517 '01-06-2023' 992.5166667]
 [518 '02-06-2023' 994.4666667]
 [547 '01-07-2023' 1008.183333]
 [548 '02-07-2023' 1007.1]
 [578 '01-08-2023' 936.5333333]
 [579 '02-08-2023' 933.2333333]
 [609 '01-09-2023' 821.8333333]
 [610 '02-09-2023' 817.8833333]]
X:
 [[1.00000e+00 1.00000e+00 1.00000e+00]
 [1.00000e+00 2.00000e+00 4.00000e+00]
 [1.00000e+00 3.20000e+01 1.02400e+03]
 [1.00000e+00 3.30000e+01 1.08900e+03]
 [1.00000e+00 6.00000e+01 3.60000e+03]
 [1.00000e+00 6.10000e+01 3.72100e+03]
 [1.00000e+00 9.10000e+01 8.28100e+03]
 [1.00000e+00 9.20000e+01 8.46400e+03]
 [1.00000e+00 1.21000e+02 1.46410e+04]
 [1.00000e+00 1.22000e+02 1.48840e+04]
 [1.00000e+00 1.52000e+02 2.31040e+04]
 [1.00000e+00 1.53000e+02 2.34090e+04]
 [1.00000e+00 1.82000e+02 3.31240e+04]
 [1.00000e+00 1.83000e+02 3.34890e+04]
 [1.00000e+00 2.13000e+02 4.53690e+04]
 [1.00000e+00 2.14000e+02 4.57960e+04]
 [1.00000e+00 2.44000e+02 5.95360e+04]
 [1.00000e+00 2.45000e+02 6.00250e+04]
 [1.00000e+00 2.74000e+02 7.50760e+04]
 [1.00000e+00 2.75000e+02 7.56250e+04]
 [1.00000e+00 3.05000e+02 9.30250e+04]
 [1.00000e+00 3.06000e+02 9.36360e+04]
 [1.00000e+00 3.35000e+02 1.12225e+05]
 [1.00000e+00 3.36000e+02 1.12896e+05]
 [1.00000e+00 3.66000e+02 1.33956e+05]
 [1.00000e+00 3.67000e+02 1.34689e+05]
 [1.00000e+00 3.97000e+02 1.57609e+05]
 [1.00000e+00 3.98000e+02 1.58404e+05]
 [1.00000e+00 4.25000e+02 1.80625e+05]
 [1.00000e+00 4.26000e+02 1.81476e+05]
 [1.00000e+00 4.56000e+02 2.07936e+05]
 [1.00000e+00 4.57000e+02 2.08849e+05]
 [1.00000e+00 4.86000e+02 2.36196e+05]
```

```
 [1.00000e+00 4.87000e+02 2.37169e+05]
 [1.00000e+00 5.17000e+02 2.67289e+05]
 [1.00000e+00 5.18000e+02 2.68324e+05]
 [1.00000e+00 5.47000e+02 2.99209e+05]
 [1.00000e+00 5.48000e+02 3.00304e+05]
 [1.00000e+00 5.78000e+02 3.34084e+05]
 [1.00000e+00 5.79000e+02 3.35241e+05]
 [1.00000e+00 6.09000e+02 3.70881e+05]
 [1.00000e+00 6.10000e+02 3.72100e+05]]
y:
 [[ 471.2833333]
 [ 472.55     ]
 [ 471.2833333]
 [ 472.55     ]
 [ 656.1833333]
 [ 660.25     ]
 [ 783.05     ]
 [ 787.1333333]
 [ 900.8833333]
 [ 904.5333333]
 [ 992.9833333]
 [ 994.9166667]
 [1007.933333 ]
 [1006.816667 ]
 [ 935.7333333]
 [ 932.4333333]
 [ 820.8666667]
 [ 816.9166667]
 [ 700.1666667]
 [ 696.1333333]
 [ 578.5833333]
 [ 574.9166667]
 [ 487.1166667]
 [ 485.05     ]
 [ 471.0166667]
 [ 472.2166667]
 [ 548.1166667]
 [ 551.5833333]
 [ 655.2      ]
 [ 659.25     ]
 [ 782.0833333]
 [ 786.15     ]
 [ 899.9833333]
 [ 903.65     ]
 [ 992.5166667]
 [ 994.4666667]
 [1008.183333 ]
 [1007.1      ]
```

```
 [ 936.5333333]
 [ 933.2333333]
 [ 821.8333333]
 [ 817.8833333]]
:
 [[6.88458363e+02]
 [4.95712907e-02]
 [4.32066810e-04]]
X :
 [[688.50836656]
 [688.55923405]
 [690.48708092]
 [690.56473655]
 [692.98808116]
 [693.08993254]
 [696.54729591]
 [696.67593543]
 [700.78237955]
 [700.93694307]
 [705.97567097]
 [706.15702264]
 [711.79211913]
 [711.99939481]
 [718.61948724]
 [718.85355105]
 [726.27728775]
 [726.53813971]
 [734.4787447 ]
 [734.76552067]
 [743.77062189]
 [744.084186  ]
 [753.55344336]
 [753.89293148]
 [764.47939722]
 [764.84567349]
 [776.2357835 ]
 [776.6288479 ]
 [787.56822934]
 [787.98548948]
 [800.905116  ]
 [801.34916429]
 [814.60246278]
 [815.07243507]
 [829.57342612]
 [830.07018656]
 [844.85213742]
 [845.37482187]
 [861.45717743]
```

```
 [862.00665002]
 [878.89264986]
 [879.46891059]]
```

```
Quadratic Model
```




```
R_Squared: 0.09544526310757495
```

[11]:
```
# CUBIC MODEL

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

def Regression():
    # Load dataset from CSV file
    dataset = np.array(pd.read_csv('Daylength.csv')) # Day length in this␣
 ↪dataset is converted to minutes

    # Initialize design matrix X and response vector y
```

```
    X = np.ones((dataset.shape[0], 4))
    y = np.zeros((dataset.shape[0], 1))

    # Fill in the values of X and y from the dataset
    for i in range(0, dataset.shape[0]):
        X[i, 1] = dataset[i][0]  # Linear term
        X[i, 2] = (dataset[i][0])**2  # Quadratic term
        X[i, 3] = (dataset[i][0])**3  # Cubic term
        y[i, 0] = dataset[i][2]  # Response variable

    # Display the design matrix X and response vector y
    print('X:\n', X)
    print('y:\n', y)

    # Calculate the product of the transpose of X and X (P matrix)
    P = X.T @ X

    # Calculate the inverse of the P matrix (Q matrix)
    Q = np.linalg.inv(P)

    # Calculate the product of the transpose of X and y (R matrix)
    R = X.T @ y

    # Calculate the regression coefficients  using the normal equation
      = Q @ R
    print(' :\n',  )

    # Display the predicted values X
    print('X :\n', X @  )

    return

# Perform cubic regression
  = Regression()

# Load dataset again for plotting and evaluation
dataset = np.array(pd.read_csv('Daylength.csv'))

print('\nCubic Model\n')

# Plot the data points and the cubic regression curve
plt.figure()
C_x = np.array(dataset[:, 0])
C_Y = np.array([])
for i in range(0, dataset.shape[0]):
    plt.scatter(dataset[i][0], dataset[i][2], color='blue')
```

```
for j in range(0, len(C_x)):
    C_Y = np.append(C_Y, [0, 0] + [1, 0] * C_x[j] + [2, 0] * C_x[j]**2 + [3,␣
 ↪0] * C_x[j]**3)

plt.plot(C_x, C_Y)
plt.xlabel('Predictor (Day)')
plt.ylabel('Response (Day Length)')
plt.grid()
plt.show()

# Evaluate R-squared value for cubic regression
C_Actual_value = np.array(dataset[:, 2])
C_Response_value = np.array([])

C_Predictor = np.array(dataset[:, 0])
for j in range(0, C_Predictor.shape[0]):
    C_Response_value = np.append(C_Response_value, [0, 0] + [1, 0] *␣
 ↪C_Predictor[j] + [2, 0] * C_Predictor[j]**2 + [3, 0] * C_Predictor[j]**3)

C_mean = np.mean(C_Actual_value)

C_R_squared = 1 - ((np.sum((C_Actual_value - C_Response_value)**2)) / (np.
 ↪sum((C_Actual_value - C_mean)**2)))

print('R_Squared:', C_R_squared)
```

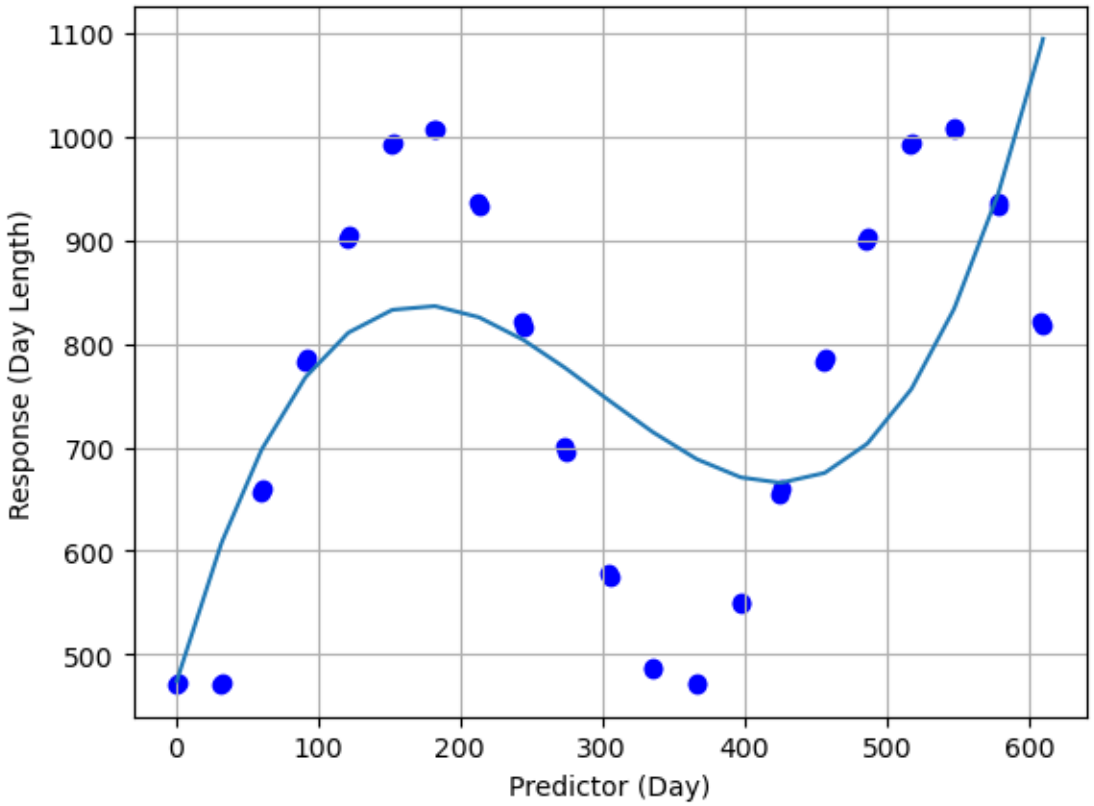
```
X:
 [[1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00]
 [1.00000000e+00 2.00000000e+00 4.00000000e+00 8.00000000e+00]
 [1.00000000e+00 3.20000000e+01 1.02400000e+03 3.27680000e+04]
 [1.00000000e+00 3.30000000e+01 1.08900000e+03 3.59370000e+04]
 [1.00000000e+00 6.00000000e+01 3.60000000e+03 2.16000000e+05]
 [1.00000000e+00 6.10000000e+01 3.72100000e+03 2.26981000e+05]
 [1.00000000e+00 9.10000000e+01 8.28100000e+03 7.53571000e+05]
 [1.00000000e+00 9.20000000e+01 8.46400000e+03 7.78688000e+05]
 [1.00000000e+00 1.21000000e+02 1.46410000e+04 1.77156100e+06]
 [1.00000000e+00 1.22000000e+02 1.48840000e+04 1.81584800e+06]
 [1.00000000e+00 1.52000000e+02 2.31040000e+04 3.51180800e+06]
 [1.00000000e+00 1.53000000e+02 2.34090000e+04 3.58157700e+06]
 [1.00000000e+00 1.82000000e+02 3.31240000e+04 6.02856800e+06]
 [1.00000000e+00 1.83000000e+02 3.34890000e+04 6.12848700e+06]
 [1.00000000e+00 2.13000000e+02 4.53690000e+04 9.66359700e+06]
 [1.00000000e+00 2.14000000e+02 4.57960000e+04 9.80034400e+06]
 [1.00000000e+00 2.44000000e+02 5.95360000e+04 1.45267840e+07]
 [1.00000000e+00 2.45000000e+02 6.00250000e+04 1.47061250e+07]
 [1.00000000e+00 2.74000000e+02 7.50760000e+04 2.05708240e+07]
 [1.00000000e+00 2.75000000e+02 7.56250000e+04 2.07968750e+07]
```

```
 [1.00000000e+00 3.05000000e+02 9.30250000e+04 2.83726250e+07]
 [1.00000000e+00 3.06000000e+02 9.36360000e+04 2.86526160e+07]
 [1.00000000e+00 3.35000000e+02 1.12225000e+05 3.75953750e+07]
 [1.00000000e+00 3.36000000e+02 1.12896000e+05 3.79330560e+07]
 [1.00000000e+00 3.66000000e+02 1.33956000e+05 4.90278960e+07]
 [1.00000000e+00 3.67000000e+02 1.34689000e+05 4.94308630e+07]
 [1.00000000e+00 3.97000000e+02 1.57609000e+05 6.25707730e+07]
 [1.00000000e+00 3.98000000e+02 1.58404000e+05 6.30447920e+07]
 [1.00000000e+00 4.25000000e+02 1.80625000e+05 7.67656250e+07]
 [1.00000000e+00 4.26000000e+02 1.81476000e+05 7.73087760e+07]
 [1.00000000e+00 4.56000000e+02 2.07936000e+05 9.48188160e+07]
 [1.00000000e+00 4.57000000e+02 2.08849000e+05 9.54439930e+07]
 [1.00000000e+00 4.86000000e+02 2.36196000e+05 1.14791256e+08]
 [1.00000000e+00 4.87000000e+02 2.37169000e+05 1.15501303e+08]
 [1.00000000e+00 5.17000000e+02 2.67289000e+05 1.38188413e+08]
 [1.00000000e+00 5.18000000e+02 2.68324000e+05 1.38991832e+08]
 [1.00000000e+00 5.47000000e+02 2.99209000e+05 1.63667323e+08]
 [1.00000000e+00 5.48000000e+02 3.00304000e+05 1.64566592e+08]
 [1.00000000e+00 5.78000000e+02 3.34084000e+05 1.93100552e+08]
 [1.00000000e+00 5.79000000e+02 3.35241000e+05 1.94104539e+08]
 [1.00000000e+00 6.09000000e+02 3.70881000e+05 2.25866529e+08]
 [1.00000000e+00 6.10000000e+02 3.72100000e+05 2.26981000e+08]]
y:
 [[ 471.2833333]
 [ 472.55     ]
 [ 471.2833333]
 [ 472.55     ]
 [ 656.1833333]
 [ 660.25     ]
 [ 783.05     ]
 [ 787.1333333]
 [ 900.8833333]
 [ 904.5333333]
 [ 992.9833333]
 [ 994.9166667]
 [1007.933333 ]
 [1006.816667 ]
 [ 935.7333333]
 [ 932.4333333]
 [ 820.8666667]
 [ 816.9166667]
 [ 700.1666667]
 [ 696.1333333]
 [ 578.5833333]
 [ 574.9166667]
 [ 487.1166667]
 [ 485.05     ]
 [ 471.0166667]
```

```
 [ 472.2166667]
 [ 548.1166667]
 [ 551.5833333]
 [ 655.2      ]
 [ 659.25     ]
 [ 782.0833333]
 [ 786.15     ]
 [ 899.9833333]
 [ 903.65     ]
 [ 992.5166667]
 [ 994.4666667]
 [1008.183333 ]
 [1007.1      ]
 [ 936.5333333]
 [ 933.2333333]
 [ 821.8333333]
 [ 817.8833333]]
:
 [[ 4.69278852e+02]
 [ 4.88627138e+00]
 [-1.97806278e-02]
 [ 2.20502081e-05]]
X :
 [[ 474.14536475]
 [ 478.9724486 ]
 [ 606.10671429]
 [ 609.77712196]
 [ 696.00771922]
 [ 698.74266796]
 [ 766.74256537]
 [ 768.56281693]
 [ 809.97280494]
 [ 811.02892132]
 [ 832.41657271]
 [ 832.80817356]
 [ 836.29790483]
 [ 836.16748179]
 [ 825.71167557]
 [ 825.16691867]
 [ 804.18821914]
 [ 803.35626987]
 [ 776.65774373]
 [ 775.66892202]
 [ 745.12100313]
 [ 744.09517073]
 [ 715.28464694]
 [ 714.34405337]
 [ 688.99570354]
```

```
 [ 688.26828093]
 [ 671.22218249]
 [ 670.83507234]
 [ 665.76629169]
 [ 665.79584137]
 [ 675.08859646]
 [ 675.70043759]
 [ 703.07265396]
 [ 704.36905859]
 [ 755.42018683]
 [ 757.54906457]
 [ 832.42595605]
 [ 835.48150856]
 [ 943.05979838]
 [ 947.19800566]
 [1089.16306107]
 [1094.51106462]]
```

Cubic Model



R_Squared: 0.4060425652814097

A better fit is typically indicated by a greater R-squared value. Thus, the cubic model (second model) against the quadratic model (first model) seems to offer a superior fit to the data based solely on the R-squared values. Moreover, the cubic model's fitted line nearly matches the data points when we look at the scatter plots, whereas the quadratic model's simple curve is seen. These fitted curves/lines also have an impact on the residuals and the response value, which indicates that the quadratic model is less predictive than the cubic model.

[12]:
```
#QUARTIC MODEL

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

def Regression():
    # Load dataset from CSV file
    dataset = np.array(pd.read_csv('Daylength.csv')) # Day length in this␣
 ↪dataset is converted to minutes

    # Initialize design matrix X and response vector y
    X = np.ones((dataset.shape[0], 5))
    y = np.zeros((dataset.shape[0], 1))

    # Fill in the values of X and y from the dataset
    for i in range(0, dataset.shape[0]):
        X[i, 1] = dataset[i][0]  # Linear term
        X[i, 2] = (dataset[i][0])**2  # Quadratic term
        X[i, 3] = (dataset[i][0])**3  # Cubic term
        X[i, 4] = (dataset[i][0])**4  # Quartic term
        y[i, 0] = dataset[i][2]  # Response variable

    # Display the design matrix X and response vector y
    print('X:\n', X)
    print('y:\n', y)

    # Calculate the product of the transpose of X and X (P matrix)
    P = X.T @ X

    # Calculate the inverse of the P matrix (Q matrix)
    Q = np.linalg.inv(P)

    # Calculate the product of the transpose of X and y (R matrix)
    R = X.T @ y

    # Calculate the regression coefficients  using the normal equation
      = Q @ R
    print(' :\n',  )
```

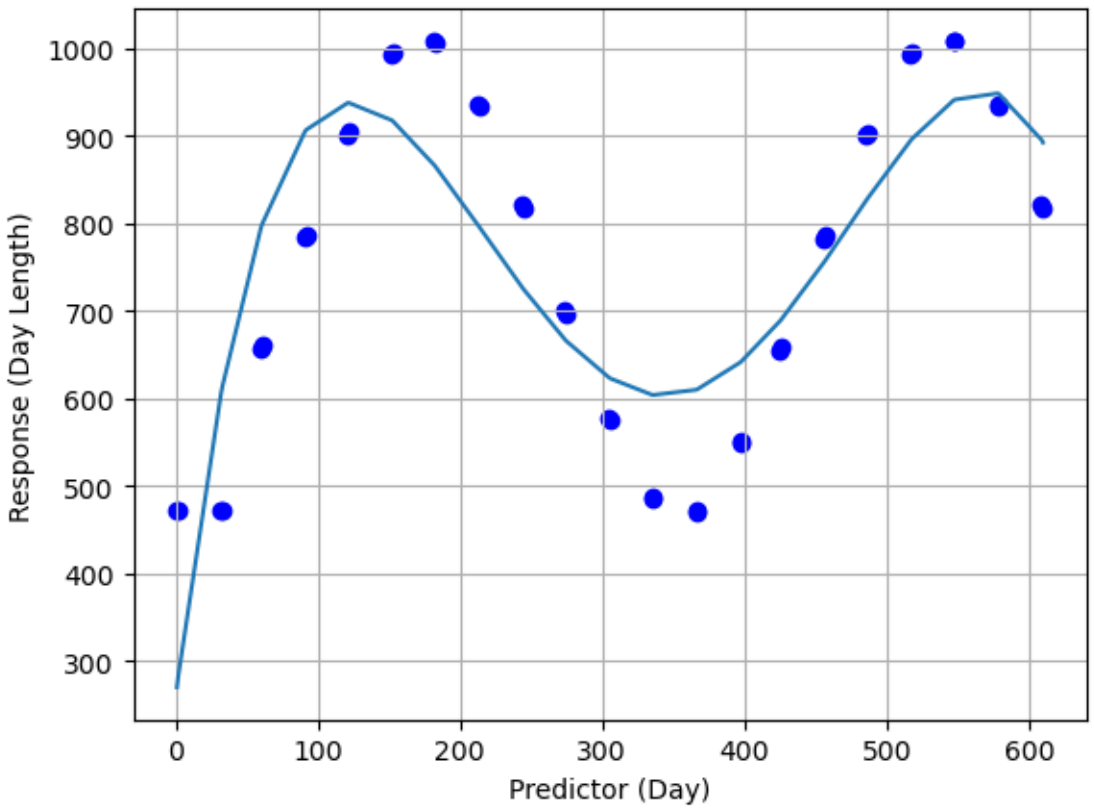
```
    # Display the predicted values X
    print('X :\n', X @  )

    return

# Perform cubic regression
 = Regression()

# Load dataset again for plotting and evaluation
dataset = np.array(pd.read_csv('Daylength.csv'))

print('\nQuartic Model\n')

# Plot the data points and the cubic regression curve
plt.figure()
q_x = np.array(dataset[:, 0])
q_Y = np.array([])
for i in range(0, dataset.shape[0]):
    plt.scatter(dataset[i][0], dataset[i][2], color='blue')

for j in range(0, len(q_x)):
    q_Y = np.append(q_Y,  [0, 0] +  [1, 0] * q_x[j] +  [2, 0] * q_x[j]**2 +  [3,␣
 ↪0] * q_x[j]**3 +  [4, 0] * q_x[j]**4)

plt.plot(q_x, q_Y)
plt.xlabel('Predictor (Day)')
plt.ylabel('Response (Day Length)')
plt.grid()
plt.show()

# Evaluate R-squared value for cubic regression
q_Actual_value = np.array(dataset[:, 2])
q_Response_value = np.array([])

q_Predictor = np.array(dataset[:, 0])
for j in range(0, q_Predictor.shape[0]):
    q_Response_value = np.append(q_Response_value,  [0, 0] +  [1, 0] *␣
 ↪q_Predictor[j] +  [2, 0] * q_Predictor[j]**2 +  [3, 0] * q_Predictor[j]**3 +␣
 ↪ [4, 0] * q_Predictor[j]**4)

q_mean = np.mean(q_Actual_value)

q_R_squared = 1 - ((np.sum((q_Actual_value - q_Response_value)**2)) / (np.
 ↪sum((q_Actual_value - q_mean)**2)))

print('R_Squared:', q_R_squared)
```

```
X:
 [[1.00000000e+00 1.00000000e+00 1.00000000e+00 1.00000000e+00
  1.00000000e+00]
 [1.00000000e+00 2.00000000e+00 4.00000000e+00 8.00000000e+00
  1.60000000e+01]
 [1.00000000e+00 3.20000000e+01 1.02400000e+03 3.27680000e+04
  1.04857600e+06]
 [1.00000000e+00 3.30000000e+01 1.08900000e+03 3.59370000e+04
  1.18592100e+06]
 [1.00000000e+00 6.00000000e+01 3.60000000e+03 2.16000000e+05
  1.29600000e+07]
 [1.00000000e+00 6.10000000e+01 3.72100000e+03 2.26981000e+05
  1.38458410e+07]
 [1.00000000e+00 9.10000000e+01 8.28100000e+03 7.53571000e+05
  6.85749610e+07]
 [1.00000000e+00 9.20000000e+01 8.46400000e+03 7.78688000e+05
  7.16392960e+07]
 [1.00000000e+00 1.21000000e+02 1.46410000e+04 1.77156100e+06
  2.14358881e+08]
 [1.00000000e+00 1.22000000e+02 1.48840000e+04 1.81584800e+06
  2.21533456e+08]
 [1.00000000e+00 1.52000000e+02 2.31040000e+04 3.51180800e+06
  5.33794816e+08]
 [1.00000000e+00 1.53000000e+02 2.34090000e+04 3.58157700e+06
  5.47981281e+08]
 [1.00000000e+00 1.82000000e+02 3.31240000e+04 6.02856800e+06
  1.09719938e+09]
 [1.00000000e+00 1.83000000e+02 3.34890000e+04 6.12848700e+06
  1.12151312e+09]
 [1.00000000e+00 2.13000000e+02 4.53690000e+04 9.66359700e+06
  2.05834616e+09]
 [1.00000000e+00 2.14000000e+02 4.57960000e+04 9.80034400e+06
  2.09727362e+09]
 [1.00000000e+00 2.44000000e+02 5.95360000e+04 1.45267840e+07
  3.54453530e+09]
 [1.00000000e+00 2.45000000e+02 6.00250000e+04 1.47061250e+07
  3.60300062e+09]
 [1.00000000e+00 2.74000000e+02 7.50760000e+04 2.05708240e+07
  5.63640578e+09]
 [1.00000000e+00 2.75000000e+02 7.56250000e+04 2.07968750e+07
  5.71914062e+09]
 [1.00000000e+00 3.05000000e+02 9.30250000e+04 2.83726250e+07
  8.65365062e+09]
 [1.00000000e+00 3.06000000e+02 9.36360000e+04 2.86526160e+07
  8.76770050e+09]
 [1.00000000e+00 3.35000000e+02 1.12225000e+05 3.75953750e+07
  1.25944506e+10]
 [1.00000000e+00 3.36000000e+02 1.12896000e+05 3.79330560e+07
```

```
  1.27455068e+10]
 [1.00000000e+00 3.66000000e+02 1.33956000e+05 4.90278960e+07
  1.79442099e+10]
 [1.00000000e+00 3.67000000e+02 1.34689000e+05 4.94308630e+07
  1.81411267e+10]
 [1.00000000e+00 3.97000000e+02 1.57609000e+05 6.25707730e+07
  2.48405969e+10]
 [1.00000000e+00 3.98000000e+02 1.58404000e+05 6.30447920e+07
  2.50918272e+10]
 [1.00000000e+00 4.25000000e+02 1.80625000e+05 7.67656250e+07
  3.26253906e+10]
 [1.00000000e+00 4.26000000e+02 1.81476000e+05 7.73087760e+07
  3.29335386e+10]
 [1.00000000e+00 4.56000000e+02 2.07936000e+05 9.48188160e+07
  4.32373801e+10]
 [1.00000000e+00 4.57000000e+02 2.08849000e+05 9.54439930e+07
  4.36179048e+10]
 [1.00000000e+00 4.86000000e+02 2.36196000e+05 1.14791256e+08
  5.57885504e+10]
 [1.00000000e+00 4.87000000e+02 2.37169000e+05 1.15501303e+08
  5.62491346e+10]
 [1.00000000e+00 5.17000000e+02 2.67289000e+05 1.38188413e+08
  7.14434095e+10]
 [1.00000000e+00 5.18000000e+02 2.68324000e+05 1.38991832e+08
  7.19977690e+10]
 [1.00000000e+00 5.47000000e+02 2.99209000e+05 1.63667323e+08
  8.95260257e+10]
 [1.00000000e+00 5.48000000e+02 3.00304000e+05 1.64566592e+08
  9.01824924e+10]
 [1.00000000e+00 5.78000000e+02 3.34084000e+05 1.93100552e+08
  1.11612119e+11]
 [1.00000000e+00 5.79000000e+02 3.35241000e+05 1.94104539e+08
  1.12386528e+11]
 [1.00000000e+00 6.09000000e+02 3.70881000e+05 2.25866529e+08
  1.37552716e+11]
 [1.00000000e+00 6.10000000e+02 3.72100000e+05 2.26981000e+08
  1.38458410e+11]]
y:
 [[ 471.2833333]
 [ 472.55     ]
 [ 471.2833333]
 [ 472.55     ]
 [ 656.1833333]
 [ 660.25     ]
 [ 783.05     ]
 [ 787.1333333]
 [ 900.8833333]
 [ 904.5333333]
```

```
 [ 992.9833333]
 [ 994.9166667]
 [1007.933333 ]
 [1006.816667 ]
 [ 935.7333333]
 [ 932.4333333]
 [ 820.8666667]
 [ 816.9166667]
 [ 700.1666667]
 [ 696.1333333]
 [ 578.5833333]
 [ 574.9166667]
 [ 487.1166667]
 [ 485.05     ]
 [ 471.0166667]
 [ 472.2166667]
 [ 548.1166667]
 [ 551.5833333]
 [ 655.2      ]
 [ 659.25     ]
 [ 782.0833333]
 [ 786.15     ]
 [ 899.9833333]
 [ 903.65     ]
 [ 992.5166667]
 [ 994.4666667]
 [1008.183333 ]
 [1007.1      ]
 [ 936.5333333]
 [ 933.2333333]
 [ 821.8333333]
 [ 817.8833333]]
:
 [[ 2.56880718e+02]
 [ 1.34471727e+01]
 [-8.59380167e-02]
 [ 1.92726605e-04]
 [-1.39646420e-07]]
X :
 [[270.24214558]
 [283.43285118]
 [605.35854994]
 [613.81132239]
 [794.15334696]
 [799.19464593]
 [904.57764674]
 [906.71095307]
 [937.26259426]
```

```
 [937.3602083 ]
 [917.61532255]
 [916.31665357]
 [866.30075526]
 [864.24227401]
 [797.19818343]
 [794.86852827]
 [726.30118065]
 [724.12397131]
 [666.96466647]
 [665.24428426]
 [623.59273484]
 [622.56683796]
 [604.14867785]
 [603.9170977 ]
 [609.76821225]
 [610.38655643]
 [640.95561375]
 [642.35471882]
 [688.1341134 ]
 [690.09592389]
 [755.34716402]
 [757.6822554 ]
 [826.64854505]
 [829.00404822]
 [894.5497507 ]
 [896.47697909]
 [940.11465108]
 [941.09952746]
 [948.2111007 ]
 [947.57954613]
 [895.17624347]
 [892.17628384]]

Quartic Model
```



`R_Squared: 0.7167175524665357`

Upon examining the Quadratic and Cubic Scatter Plots, I saw a pattern of increased R_squared value and better fit to the data points. Thus, the Quartic Model appears to fit even better than the preceding model, the Cubic Model, and its R-squared value is higher as well. It is evident that the fitted line has a closer relationship with the datapoints in the obtained scatter plot. Since the Quartic Model fits the data the best and is the best predictor when compared to the other models, I thus suggest it.