Principles of Data Science

Submitted by:

RAMANA Kulanthaivelu

Student ID: 14231913

Table of contents:

1. TASK 1: POPULATION DYNAMICS IN ECOSYSTEM
2. Matrix population ……………………………………………….. 3
3. Calculating and plotting the relative populations ……………….. 5
4. Determining Eigen values……………………………………….. 7
5. Depicting the population trends for various initial populations…. 8
6. TASK 2: REGRESSION BY MATRIX OPERATION
7. Calculating the R2  and plotting the best fitted line for the Regression matrix …………………………………………………………… 10
8. Quadratic regression …………………………………………….. 12
9. Polynomial regression
10. Assembling the dataset……………………………………….. 14
11. Polynomial regression ……………………………………….. 15
12. Further polynomial regression ……………………………….. 17

3. References ………………………………………………… 20

**TASK : 1 POPULATION DYNAMICS IN ECOSYSTEM**

**a) Matrix population**

The population of Rabbits and Foxes at the current time step,

**p =**

**New Rabbit Population = Old Rabbit Population + 0.05 \* Old Rabbit Population - 0.05 \* Old Fox Population**

**New Fox Population = Old Fox Population - 0.08 \* Old Fox Population + 0.05 \* Old Rabbit Population**

=

.

Let’s assume the Old\_Rabbit\_Population and the Old\_Fox\_Population as 100%.

Thus we will get,

*= .*

We can obtain new population using the following equation,

**new\_population = A . p**

The resulting vector represents the predicted rabbit and fox populations for the next time step. The calculated new populations become the current populations for the next iteration in order to predict subsequent time steps. To simulate population dynamics over time, this procedure can be repeated indefinitely.

Whereas , the matrix representing the change in the population,

A =

**Python code to obtain new\_population:**

**import numpy as np**

***# Constructed matrix***

**A = np.array([[1.05, -0.05],[0.05, 0.92]])**

***# The population rabbits and foxes at the current time step***

**p = np.array([[Old\_Rabbit\_population], [Old\_Fox\_population]])**

***# obtaining new populations using matrix multiplication***

**new\_population = A@p**

**b)**

**python code for calculating and plotting the relative populations:**

**import numpy as np**

**import matplotlib.pyplot as plt**

**A = np.array([[1.05, -0.05], [0.05, 0.92]])**

***# Old populations***

**p = np.array([[8000], [2000]])**

**p\_new = p**

***# List for storing new population***

**new\_rabbit\_population =[p\_new[0][0]]**

**new\_fox\_population =[p\_new[1][0]]**

***# 100 time steps of population dynamics model***

**for \_ in range(100):**

**p\_new = A@p\_new**

**new\_rabbit\_population.append(p\_new[0][0])**

**new\_fox\_population.append(p\_new[1][0])**

***# Calculating the total population***

**Total\_population = np.array(new\_rabbit\_population) +np.array(new\_fox\_population)**

***#calculating the percentage for the new populations of Rabbits and foxes***

**percentage\_rabbit\_population = (np.array(new\_rabbit\_population) / total\_population ) \* 100**

**percentage\_fox\_population = (np.array(new\_fox\_population) / Total\_population) \* 100**

***# time steps = 100 steps + initial population***

**time\_steps = range(0,101)**

***# plotting the line graph***

**plt.figure(figsize=(10, 4))**

**plt.plot(time\_steps, percentage\_rabbit\_population, label='Relative Rabbit Population')**

**plt.plot(time\_steps, percentage\_fox\_population, label='Relative Fox Population')**

**plt.title('Rabbits and Foxes Relative Population Trends Over Time')**

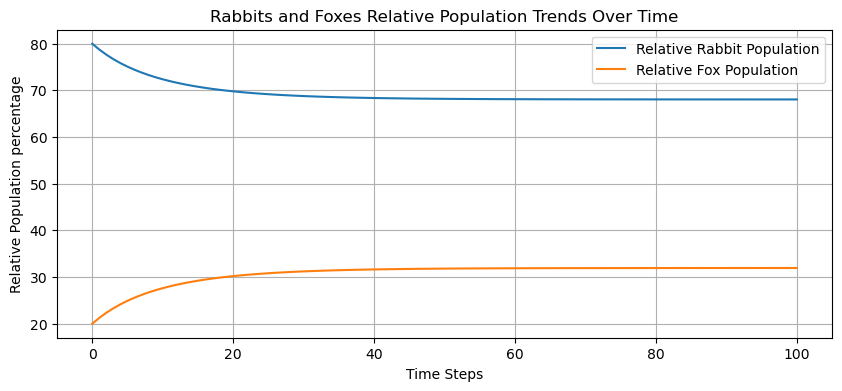
**plt.xlabel('Time Steps')**

**plt.ylabel('Relative Population percentage')**

**plt.legend()**

**plt.grid(True)**

**plt.show()**

****

**Figure 1: Rabbits and Foxes Relative population over time**

Initially, the rabbit population can reach up to 80%. The y-axis shift from 80% to below 70% represents the reduction in the relative percentage of the rabbit population. While the relative population of rabbits declines on the y-axis, the plot suggests that the percentage of rabbits remains consistent over time. This shows that the rabbit population has remained steady despite the decrease in relative share.

The fox population, on the other hand, starts low (about 20%) and steadily grows. The fox line parallels the rabbit line on the x-axis, illustrating that the fox population is steadily increasing in relative percentage over time with no fluctuation.

The rabbit population shrinks slightly at first while keeping a pretty stable total population level, whereas the fox population develops slowly in tandem without having an immediate influence on the rabbit population. This type of pattern may depict a circumstance in which the initial fall in rabbit population percentage has little effect on overall rabbit population size but produces a slow increase in the fox population, influencing the dynamics between the two populations in succeeding time steps.

**C) Determining the Eigen values**

**import numpy as np**

***# Constructed matrix***

**A = np.array([[1.05, -0.05],[0.05, 0.92]])**

**eig\_values,eig\_vectors = np.linalg.eig(A)**

**print(eig\_values)**

**print(eig\_vectors)**

**Output:**

**[1.02653312 0.94346688]**

**[[0.9052544 0.42486994]**

**[0.42486994 0.9052544 ]]**

The magnitude of this eigenvalue, roughly 1.02653312, is more than one but near to one. The closeness to one indicates that the linked factor in population dynamics indicated by this eigenvalue leads to a slower rate of convergence to stable equilibrium and may correlate to the system's dominating behaviour. Despite the fact that it is bigger than one, the system tends to settle at a little slower rate than the other eigenvalue.

The magnitude of this eigenvalue, about 0.94346688, is less than one. The system will almost definitely attain an equilibrium in which both populations are stable.

In conclusion, both eigenvalues suggest that rabbit and fox populations tend to stabilise over time.In summary, the eigenvalues provide a quantitative assessment of the ecosystem's stability, and the dominant eigenvalue determines the overall behaviour of the populations. The eigenvalue signal of instability warns that the dynamics of rabbit and fox populations may someday become unmanageable and unpredictable. Understanding these eigenvalues is crucial for anticipating and regulating predator-prey interactions' ecological repercussions.r time.

**d) Depicting the population trends for various initial populations:**

If the rabbit population is reduced by 200% of the fox population, it signifies that the rabbit population is dropping at double the rate of the fox population. We can update the matrix coefficients to reflect this shift in the interaction between rabbit and fox populations,

A =

**import numpy as np**

**import matplotlib.pyplot as plt**

***# Setting different initial population values for rabbits and foxes***

**rabbit\_initial\_population = [7000,5000]**

**fox\_initial\_population = [2000,5000]**

**plt.figure(figsize=(8, 6))**

***# Function to calculate the population changes***

**def population\_changes(rabbit, fox):**

***# Modified matrix (A) based on the fox population***

**A\_modified = np.array([[1.05, -2],**

**[0.05, 0.92]])**

***# Calculating the population changes using matrix multiplication***

**population = np.array([[rabbit], [fox]])**

**next\_population = A\_modified @ population**

**return next\_population.flatten()**

***# Plotting the trajectories for different initial population values***

**for rabbit\_initial in rabbit\_initial\_population:**

**for fox\_initial in fox\_initial\_population:**

**rabbit\_new\_population = [rabbit\_initial]**

**fox\_new\_population = [fox\_initial]**

**for \_ in range(100):**

**rabbit\_new, fox\_new = population\_changes(rabbit\_new\_population[-1], fox\_new\_population[-1])**

**rabbit\_new\_population.append(rabbit\_new)**

**fox\_new\_population.append(fox\_new)**

**plt.plot(rabbit\_new\_population, fox\_new\_population, label=f'Rabbit: {rabbit\_initial}, Fox: {fox\_initial}')**

**plt.xlabel('Rabbit Population')**

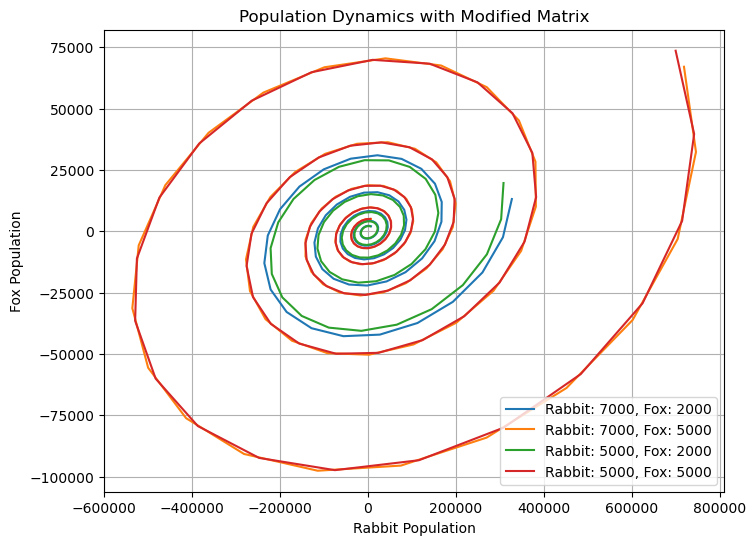
**plt.ylabel('Fox Population')**

**plt.title('Population Dynamics with Modified Matrix')**

**plt.legend()**

**plt.grid(True)**

**plt.show()**

****

**Figure 2: Population Dynamics with modified matrix**

The snail pattern demonstrates cyclical behavior, suggesting that over a lengthy period of time, fox and rabbit populations consistently oscillate. The provided x- and y-axis ranges show the limits of these population fluctuations.The range of the rabbit population on the x-axis, from -600,000 to 800,000, indicates the population's instability. Positive numbers can suggest a significant growth or abundance in the rabbit population, whereas negative values might suggest a sharp decline. Additionally, the range of the fox population on the y-axis from -100,000 to 75,000 illustrates the variability in the fox population. Population growth or rise is shown by positive figures, whereas population decline is indicated by negative values. When the initial numbers of foxes and rabbits are set at 7000 and 5000, respectively, then 5000 and 5000. This suggests that population dynamics have a more prominent and significant cyclic behavior. This may indicate that fox and rabbit population oscillations are growing more noticeable and consequential over time. The cycle may fluctuate for longer periods of time or with greater amplitude changes. A greater initial population difference may lead to more pronounced or prolonged cyclic behavior, which would result in higher variations over time because of the initial imbalance in predator-prey relationships. The cyclic behavior may be lower in scale or duration when the beginning populations are closer together (7000, 2000) and (5000, 2000), which may suggest a more stable or balanced interaction between the two species. These boundaries show a dynamic system in which populations swing between high and low values, presumably suggesting periods of abundance, scarcity, recuperation, and predator-prey dynamics. This is suggested by the boundaries' wide range and cyclical character.

**TASK : 2 REGRESSION BY MATRIX OPERATIONS**

1. **Calculating the R2  and plotting the best fitted line for the Regression matrix**

**import numpy as np**

**import matplotlib.pyplot as plt**

***# Given array***

**x\_array = np.array([[0, 1], [3, 4], [6, 5]])**

**x\_value = x\_array[:, 0]**

**y\_value = x\_array[:, 1]**

***# Creating the matrix X and vector y***

**x = np.hstack((np.ones((len(data\_points), 1)), x\_value.reshape(-1, 1)))**

**y = y\_value.reshape(-1, 1)**

***# Calculating (X^T \* X)^-1***

**X\_tra\_inverse = np.linalg.inv(X.T@X)**

***# Calculating X^T \* y***

**X\_tra\_y = X.T@y**

***# Calculating beta***

**beta = X\_tra\_inverse@ X\_tra\_y**

***# Calculating the fitted predicted values***

**fitted\_values = X@beta**

***#calculating the r\_squared value***

**correlation\_coeff = np.corrcoef(y.flatten(), fitted\_values.flatten())**

**r\_squared = correlation\_coeff[0, 1]\*\*2**

**print("R-squared value:", r\_squared)**

***# Plotting the scatterplot and the fitted line***

**plt.scatter(x\_value, y\_value)**

**plt.plot(x\_value, fitted\_values, label='Fitted Line',color='g')**

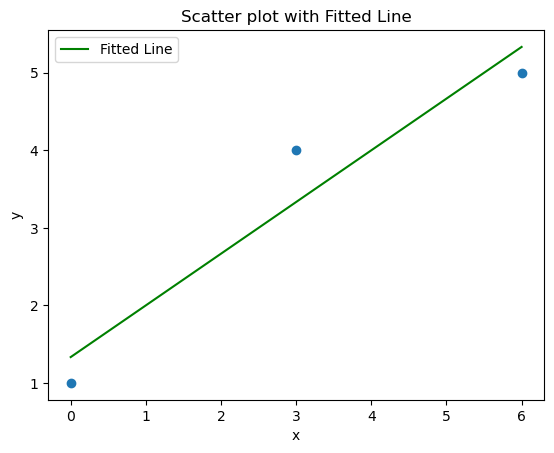
**plt.xlabel('x')**

**plt.ylabel('y')**

**plt.title('Scatter plot with Fitted Line')**

**plt.legend()**

**plt.show()**

****

**Figure 3: Scatter plot with Fitted line**

**R-squared value: 0.9230769230769227**

‘R-Squared (R² or the coefficient of determination) is a statistical measure in a regression model that determines the proportion of variance in the dependent variable that can be explained by the [independent variable](https://corporatefinanceinstitute.com/resources/financial-modeling/independent-variable/). In other words, r-squared shows how well the data fit the regression model (the goodness of fit)’. (Taylor, 2023)

The R2  value, calculated using NumPy's corrcoef function, indicates the level of how accurately it fit. When the value is close to 0.92, the expected and actual results are well correlated. The scatterplot and fitted line from Matplotlib show how effectively the linear regression model captured the underlying trend in the data. The intercept represents the expected response when the predictor variable is zero, and the slope represents the rate of change in the answer when the predictor variable changes by one unit.

1. **Quadratic Regression**

**import numpy as np**

**import matplotlib.pyplot as plt**

***# Given data points***

**x\_array = np.array([[0, 1], [3, 4], [6, 5]])**

**x\_value = x\_array[:, 0]**

**y\_value = x\_array[:, 1]**

***# Assemble X and Y matrix for quadratic regression***

**X = np.hstack((np.ones((len(data\_points), 1)), x\_value.reshape(-1, 1),**

**(x\_value\*\*2).reshape(-1, 1)))**

**y = y\_value.reshape(-1, 1)**

***# Calculating (X.T\*X)^-1***

**X\_tra\_inverse = np.linalg.inv(X.T@X)**

***# Calculate X^T y***

**X\_tra\_y = X.T@y**

***# Calculating beta***

**beta = X\_tra\_inverse@XTy**

***# quadratic of best fit: y = a + bx + cx^2***

**a, b, c = beta[0], beta[1], beta[2]**

**quadratic\_of\_best\_fit = f'y = {a} + {b}x + {c}x^2'**

***# Calculating fitted values***

**fitted\_values = X@beta**

***# Calculating R^2 value***

**correlation\_coeff = np.corrcoef(y\_value, fitted\_values[:, 0])**

**r\_squared = correlation\_coeff[0, 1]\*\*2**

**print("Quadratic coefficient of Best Fit line:", quadratic\_of\_best\_fit)**

**print("R squared Value:", r\_squared)**

***# Plotting the scatterplot and the best fitted quadratic curve***

**plt.scatter(x\_value, y\_value)**

**plt.plot(x\_values, fitted\_values, label='Best Fitted Quadratic Curve', color='g')**

**plt.xlabel('x')**

**plt.ylabel('y')**

**plt.legend()**

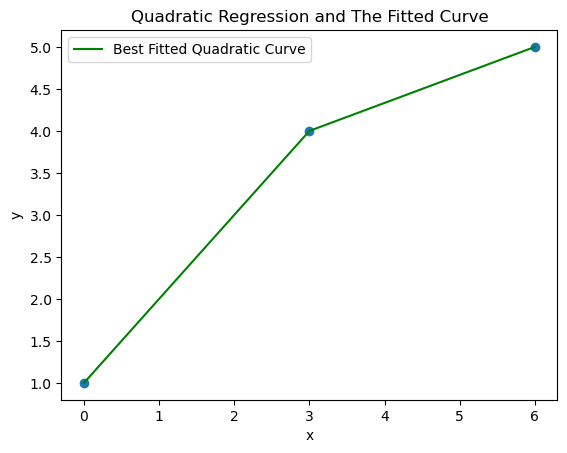
**plt.title('Quadratic Regression and The Fitted Curve')**

**plt.show()**

**Output:**

**Quadratic coefficient of Best Fit line: y = [1.] + [1.33333333]x + [-0.11111111]x^2**

**R squared Value: 1.0**

****

**Figure 4: Quadratic regression and the fitted curve**

In quadratic regression, R-squared is the coefficient of the determination and it illustrates the degree to which the variation in y can be explained by x-variables. The r-squared value, therefore, allows us to evaluate how the differences in one variable can be explained by a difference in the second variable. R-squared can take values between 0 to 1, where 0 reflects 0% variation and 1 reflects a 100% variation. (*Quadratic Regression - Voxco*, n.d.)

The quadratic regression equation y=1+1.33333333x−0.11111111x2 suggests a quadratic relationship between the predictor variable x and the response variable y for the given dataset.

When the quadratic model's R2 value is 1, it indicates that it fits the observed data points perfectly, which means that it can use the provided x and x2 terms to fully explain and capture the variability in y.

This perfect fit indicates that, for these particular data points, the quadratic equation correctly describes the relationship between x and y, providing an exact prediction of y based on the given x values.

**C)**

**1) Assembling the dataset:**

The data set has been taken ( <https://sunrise-sunset.org/gb/coventry/2023/9> ) from the given source and the Hour : Minute : Second format has been changed into minutes by using the pandas timedelta function ***'(Leem, 2021)'.*** For removing the decimals the calculated minutes are divided by 60 and got the Quotient and the new coloum is created.

**import pandas as pd**

**data = pd.read\_csv("C:/Users/Admin/Desktop/Daylight time coventry final.csv")**

**data.head(41)**

***# converting the hour:minute:second format to minutes '(Leem, 2021)'***

**data['Daylight'] = pd.to\_timedelta(data['Daylight'])**

**data['TimeInMinutes'] = data['Daylight'].dt.total\_seconds() // 60 *#'(Leem, 2021)'***

**data = data.drop('Daylight', axis=1)**

**data.head(5)**

|  | **Days** | **TimeInMinutes** |
| --- | --- | --- |
| **0** | Sun, Jan 30 | 542.0 |
| **1** | Mon, Jan 31 | 545.0 |
| **2** | Fri, Feb 25 | 640.0 |
| **3** | Sat, Feb 26 | 644.0 |
| **4** | Thu, Mar 10 | 692.0 |

Sample of the obtained dataset is given above.

**2) Polynomial regression**

The Python code that goes with it tries to fit cubic and quadratic regression models to a dataset that shows a day's duration over time. The dataset is loaded using pandas, and the 'Day length' column is converted to seconds for handling. The predictor variable (x) is created using the range of data, and the response variable (y) is set to the day duration values. Next, quadratic and cubic regression are carried out using NumPy's ‘polyfit’ function, and the R-squared values for each model are calculated. (Hanoon et al., 2023)

The extracted data set used in this code <https://github.com/Ramana-Kulanthaivelu/Data-for-Daylight-coventry>

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

***# Loading the dataset***

**data = pd.read\_csv("C:/Users/Admin/Documents/Polynaomial regression final.csv")**

***# converting the hour:minute:second format to minutes '(Leem, 2021)'***

**data['Daylight'] = pd.to\_timedelta(data['Daylight'])**

**data['TimeInMinutes'] = data['Daylight'].dt.total\_seconds() // 60 *#'(Leem, 2021)'***

**df = data.drop('Daylight', axis=1)**

**x = np.arange(len(df))**

**y = df['TimeInMinutes'].values**

***# Quadratic regression (Zach, 2020)***

**coeff\_quadratic = np.polyfit(x, y, 2) #** (Hanoon et al., 2023)

**y\_quadratic = np.polyval(coeff\_quadratic, x)**

**ssreg = np.sum((y - y\_quadratic)\*\*2)**

**sstot = np.sum((y - np.mean(y))\*\*2)**

**r\_squared\_quadratic = 1 - (ssreg / sstot) *#(Zach, 2020)***

***# Cubic regression (Zach, 2022)***

**coeff\_cubic = np.polyfit(x, y, 3)**

**y\_cubic = np.polyval(coeff\_cubic, x)**

**ccreg = np.sum((y - y\_cubic)\*\*2)**

**cctot = np.sum((y - np.mean(y))\*\*2)**

**r\_squared\_cubic = 1 - ccreg / cctot *#(Zach, 2022)***

***# Print coefficients and R-squared values***

**print("Quadratic Model Coefficients:", coeff\_quad)**

**print("Equation for the Quadratic model= {:.4f}x^2 + {:.4f}x + {:.4f}".format(\*coeff\_quad))**

**print("R-squared for this Quadratic Model:", r\_squared\_quad)**

**print("\nCubic Model Coefficients:", coeff\_cubic)**

**print("Equation for this Cubic Model = {:.4f}x^3 + {:.4f}x^2 + {:.4f}x + {:.4f}".format(\*coeff\_cubic))**

**print("R-squared for Cubic Model:", r\_squared\_cubic)**

***# Plotting the data and fitted models***

**plt.scatter(x, y, label='Actual Data')**

**plt.plot(x, y\_quad, label='Quadratic Model', color='red')**

**plt.plot(x, y\_cubic, label='Cubic Model', color='green')**

**plt.xlabel('Observation')**

**plt.ylabel('Minutes for the Daytime')**

**plt.legend()**

**plt.title('Quadratic and Cubic Regression Models for the Daytime data')**

**plt.show()**

**Output:**

**Quadratic Model Coefficients: [-5.24959300e-02 5.31277394e+00 6.86123981e+02]**

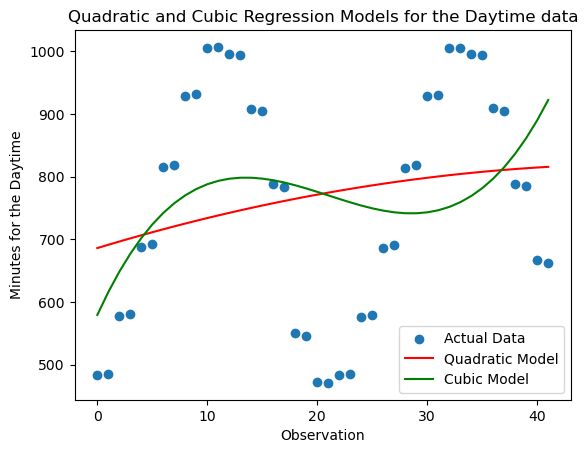
**Equation for the Quadratic model= -0.0525x^2 + 5.3128x + 686.1240**

**R-squared for this Quadratic Model: 0.045205572659048676**

**Cubic Model Coefficients: [ 3.33567139e-02 -2.10393383e+00 3.85527393e+01 5.79449210e+02]**

**Equation for this Cubic Model = 0.0334x^3 + -2.1039x^2 + 38.5527x + 579.4492**

**R-squared for Cubic Model: 0.1097769000158173**

****

When compared to the quadratic model, the cubic model shows a noticeably better fit to the data. The cubic model's higher R-squared value of 0.1098 indicates that it accounts for a larger percentage of the data's variability than the quadratic model (R-squared = 0.0452).The superior fit of the cubic model implies that it is better able to capture the underlying trend in the data, especially when the values are extreme. When predicting daytime data from a variety of observations, it offers more precise results. Nonetheless, it's critical to recognise that the R-squared values for both models are rather low. This suggests that neither model can fully account for the variation found in the dataset. This might be due to noise in the data or the impact of other unreported factors influencing the duration of daytime data.

In conclusion, the cubic model explains variability better than the quadratic model; however, both models have limited explanatory power due to their relatively low R-squared values."

This interpretation stresses the relative performance of the cubic and quadratic models, highlighting the better fit of the cubic model, despite the constraints indicated by the low R-squared values in explaining the variability in the dataset.

**3) Further polynomial regression**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from sklearn.metrics import r2\_score #** (*Sklearn.metrics.r2\_score — Scikit-Learn 0.24.1 Documentation*, n.d.)

***# Loading the dataset***

**data = pd.read\_csv("C:/Users/Admin/Documents/Polynaomial regression final.csv")**

***# converting the hour:minute:second format to minutes '(Leem, 2021)'***

**data['Daylight'] = pd.to\_timedelta(data['Daylight'])**

**data['TimeInMinutes'] = data['Daylight'].dt.total\_seconds() // 60 *#'(Leem, 2021)'***

**df = data.drop('Daylight', axis=1)**

**x = np.arange(len(df))**

**y = df['TimeInMinutes'].values**

***# Quartic regression***

**X\_quatratic = np.vander(x,5)**

**beta\_quatratic = np.linalg.inv(X\_quatratic.T @ X\_quatratic) @ X\_quatratic.T @ y**

**day\_length\_quatratic = X\_quatratic @ beta\_quatratic**

**r2\_quatratic = r2\_score(y, day\_length\_quatratic)**

**print('Value of R squared',r2\_quatratic)**

**plt.scatter(x, y, label='Actual Data')**

**plt.plot(x, day\_length\_quatratic, label='Fourth-Degree Polynomial Model', color='purple')**

**plt.xlabel('Observation')**

**plt.ylabel('Minutes for the Daytime')**

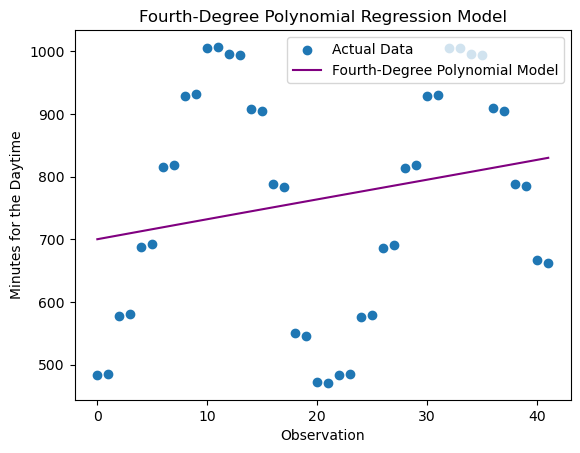
**plt.legend()**

**plt.title('Fourth-Degree Polynomial Regression Model')**

**plt.show()**

**output:**

Value of R squared 0.04383258654893363



Finding a more suitable polynomial regression model for the day length dataset is crucial, as the R-squared value of 0.0438 for the fourth-degree polynomial model indicates a poor fit. The fourth-degree polynomial may be too complex or not sufficient to capture the underlying trend, as indicated by the low R-squared value. Perhaps a simpler model, like a quadratic or cubic model, would fit the data more accurately. Keeping the dataset in mind, it would be wise to use a quadratic or cubic regression model to enhance model performance. These models strike a compromise between the amount of information they contain and their capacity to identify nonlinear patterns. A quadratic or cubic model would be suitable to predict day length based on the day of the year. These models manage to avoid being unduly complex while still capturing the nonlinear relationship between day of the year and day length. They avoid undue complexity that could cause over fitting and provide a fair amount of flexibility to account for any potential curvature in the provided dataset. The best model to balance model complexity and predictive accuracy on unobserved data will be determined by evaluating each model's performance using cross-validation or other evaluation metrics. Further supporting the chosen model's validity will be the analysis of residual plots and diagnostic metrics.

**References**

Hanoon, S. K., Abdullah, A. F., Shafri, H. Z. M., & Wayayok, A. (2023). Urban Growth Forecast Using Machine Learning Algorithms and GIS-Based Novel Techniques: A Case Study Focusing on Nasiriyah City, Southern Iraq. *ISPRS International Journal of Geo-Information*, *12*(2), 76. <https://doi.org/10.3390/ijgi12020076>

*How to Calculate Residual Sum of Squares in Python*. (2022, February 17). GeeksforGeeks. <https://www.geeksforgeeks.org/how-to-calculate-residual-sum-of-squares-in-python/>

Leem, D. (2021, February 15). *Converting Time to Seconds or Minutes for an Entire Python Dataframe / CSV Column*. The Perpetual Student. <https://blog.duaneleem.com/converting-time-to-seconds-or-minutes-for-an-entire-python-dataframe-csv-column/>

*numpy.vander — NumPy v1.23 Manual*. (n.d.). Numpy.org. <https://numpy.org/doc/stable/reference/generated/numpy.vander.html>

*Quadratic Regression - Voxco*. (n.d.). [https://www.voxco.com/blog/quadratic-regression/#:~:text=In%20quadratic%20regression%2C%20R%2Dsquared](https://www.voxco.com/blog/quadratic-regression/%23:~:text=In%20quadratic%20regression%2C%20R%2Dsquared)

*sklearn.metrics.r2\_score — scikit-learn 0.24.1 documentation*. (n.d.). Scikit-Learn.org[. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2\_score.html](.%20https:/scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html)

Taylor, S. (2023). *R-Squared*. Corporate Finance Institute. <https://corporatefinanceinstitute.com/resources/data-science/r-squared/>

*What is markov chain periodic?* (n.d.). Www.collimator.ai. Retrieved December 1, 2023, from <https://www.collimator.ai/reference-guides/what-is-markov-chain-periodic>

Zach. (2020, September 2). *How to Perform Quadratic Regression in Python*. Statology. <https://www.statology.org/quadratic-regression-python/>

Zach. (2022, November 23). *How to Perform Cubic Regression in Python*. Statology. <https://www.statology.org/cubic-regression-python/>