Indian Institute of Technology Gandhinagar

Comparative Analysis of Numerical Interpolation and Machine Learning Methods

MA 202 Project Report
24th April 2022

Authors

Aditi Agarwal 20110006
Pavidhar Jain 20110136
Tanvi Dixit 20110212
Yash Khandelwal 20110236

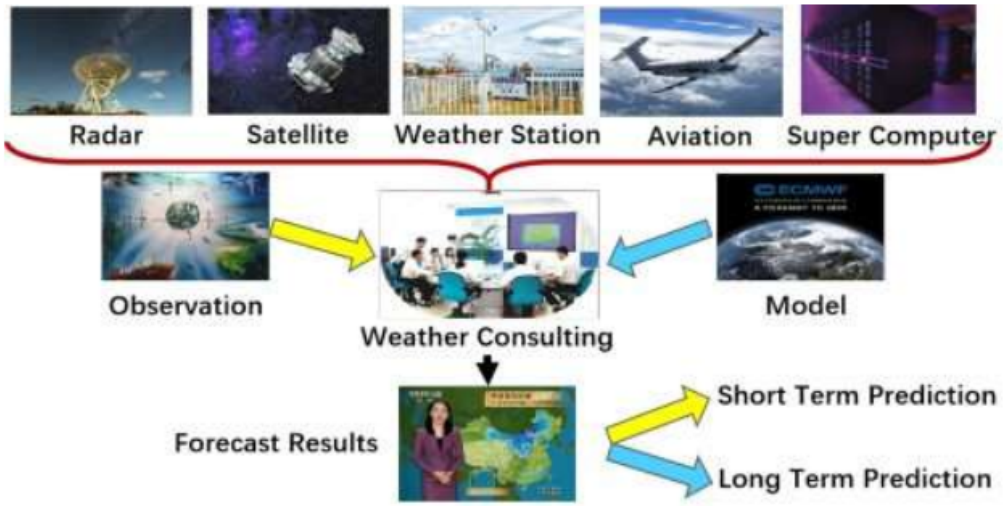
Under the guidance of

Prof. Satyajit Pramanik
Prof. Tanya Kaushal Srivastava
Prof. Uddipta Ghosh

CONTENTS

1. Introduction

Weather forecasting is the use of technology to predict the weather of a place at a particular time. But how is it done? Weather forecasting has been done informally by individuals for ages and has been done formally by computer-based algorithms ever since the 19th century. Weather forecasting is done these days through data on various weather conditions such as temperature, pressure, humidity, wind and cloud movement, sunlight, precipitation, and many more through multiple devices and satellites. This data over some time is used to predict the weather of the next few days using advanced algorithms.



The data given to us in assignments and classrooms are primarily meant to be solved, thus, have the data fabricated accordingly; however, we wanted to use these methods on some real-life data. The data used here is from the University of California, Irvine's repository consisting of the weather data of Chicago in the year 1995. This data varies drastically and unpredictably; thus, we changed our approach and fit our models accordingly. We wanted to check how methods of interpolation that were taught in this course would work on this abrupt data. Thus we have used four linear interpolation methods on certain data intervals and predicted values and checked their accuracy.

As the entire data is quite erratic, we divided the year's data into quarters corresponding to the four seasons. We randomly chose one point in each of these quarters, which the models would predict, and their accuracy will be compared. Furthermore, we decided to select and change the intervals of data used to predict the values for each method. This would help us find a more stable set of points for prediction, and generally, data on weather varies within a short span. The implementation of each

method for predicting the values of this data set helps us understand the accuracy and comparative study of each numerical method with a varied range of errors and input datasets.

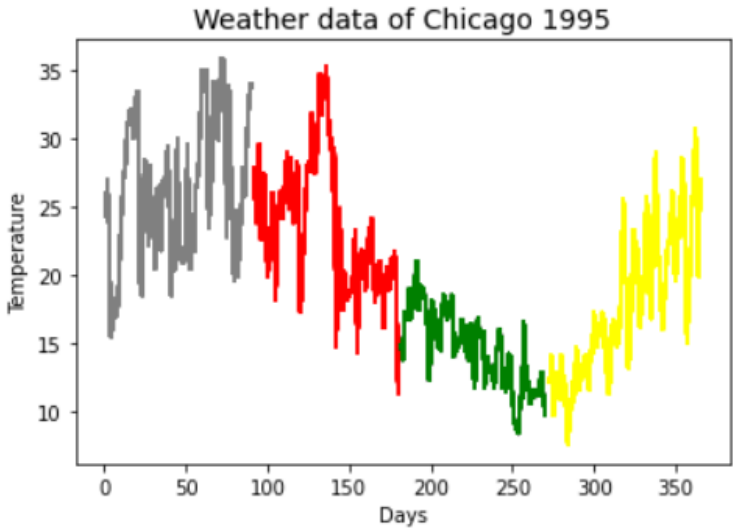
	Day	MaxTemp	Rainfall	Evaporation
0	1	24.3	0.0	3.4
1	2	26.9	3.6	4.4
2	3	23.4	3.6	5.8
3	4	15.5	39.8	7.2
4	5	16.1	2.8	5.6
5	6	16.9	0.0	5.8
6	7	18.2	0.2	4.2
7	8	17.0	0.0	5.6
8	9	19.5	0.0	4.0
9	10	22.8	16.2	5.4
10	11	25.2	0.0	4.2
11	12	27.3	0.2	7.2

Eventually, we conducted a comparative analysis of the accuracy of the methods and the polynomials as predicted. We have also incorporated an aspect of Machine Learning here as it is used extensively in weather forecasting. This was undoubtedly better as it could consider more factors and with due weightage to each factor, unlike our linear models. We have used linear and logistic regression models here, but we have applied the Machine Learning models to the entire dataset. In contrast, the interpolation methods were applied for a specific interval of four days to fifteen days.

2. Mathematical Model

2.1 Bifurcation of the dataset and obtaining points for analysis:

We imported a dataset from the University of California, Irvine's repository consisting of the weather data of Chicago in the year 1995. This entire data was plotted on a graph to get a basic idea of the data trends. On plotting, we obtained a rather irregular and unpredictable graph, as expected while dealing with something as sensitive as the weather. Thus, we decided to divide this data into quarterly segments each representing the prevalent four seasons.



We then ran a code to randomly get a point for each season, which will be predicted and analysed in-depth by our models.

```
#selecting random entries
l  = []
for i in range(10,300,90):
  l.append(random.randint(i,i+90))
x_plot = l
print(l)
```

Points thus obtained,

[46, 109, 214, 308]

We had to divide the data into smaller segments as applying linear interpolation to the entire set was resulting in large amounts of errors for the prediction of the four values. Thus to lower the error due to erratic data and get a more accurate prediction we choose to give intervals as inputs. The observed values when we applied linear interpolation to the entire set were as followed:

```
[1.2852585832200677e+31, 1.1719441580487066e+70, 9.12621348786257e+104, 5.407501374966567e+161]
inf
```

Here, we observe an infinite RMSE as the data is rather fluctuating, thus we will be considering intervals surrounding the points of concern. Further, we used linear models of interpolation like Newton's method and Lagrange's method taught in the course to predict these values. We also learned and included methods like the Cubic Spline method and Least Squares method for comparison and prediction.

2.2 Best Model to incorporate and predict the values based on different methods and intervals:

By using numerical methods of interpolation namely Newton's method, Lagrange's method, Cubic Spline method, and Least Squares method we will predict the values of these days using a varying input dataset. As the data corresponds to the weather of an area it is erratic in nature thus to predict a specific day, it is more sensible to choose an interval of inputs to get much better results.

For each of the mentioned method, a polynomial and a graph for the chosen interval is found and the values are hence predicted. Using the values as predicted from each model, we then conducted a comparative analysis of the errors and accuracy of each method for the same interval of days.

We will be varying the input dataset from a span of four days to fortnightly. The change in the prediction of these data due to this will be mapped. Using this we can find the optimal range of data that should be provided for each method and for each point.

2.3 Advancing to Machine Learning models:

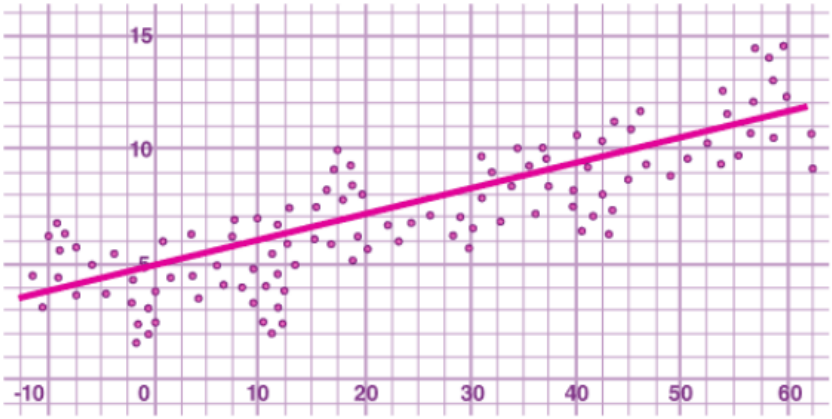
In the aforementioned methods, we are able to take into account at max one of the factors on which the weather of a place might depend. Due to this, we observe great values of errors in prediction with these models. Thus in accordance with today's used methods of weather-forecasting which use a great deal of Machine Learning, we incorporated this into our project. The Machine Learning model would be more accurate as it takes into consideration the majority of the factors and their respective contribution to the final result. Thus, using concepts of linear regression and logistic regression we are able to get a more accurate prediction of the values as compared to the model we built.

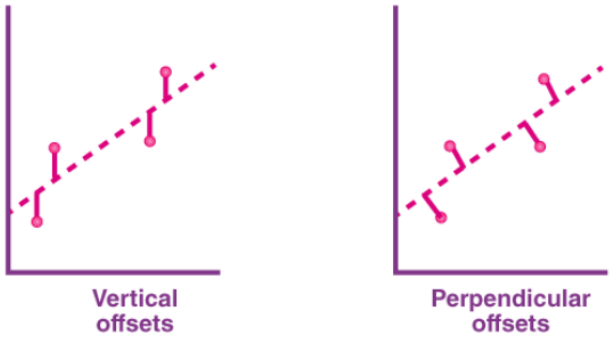
3. Numerical Analysis

3.1 Least Squares fitting method:

The least-square approach involves minimising the sum of the squares of the offsets (residual component) of the points from the curve to get the best-fitting curve or line of best fit for a group of data points. The trend of outcomes is statistically evaluated throughout the process of determining the relationship between two variables. This process is also called regression analysis owing to the approach of curve fitting involved here, which approximates the curves to the provided raw data.

It is practised to find a regression line or a best-fit line for the given pattern and is described by an equation with specific parameters. It defines the solution for the minimization of the sum of squares of deviations or the errors in the result of each equation. Curve fitting for a certain data set isn't always the same. As a result, a curve with the least amount of departure from all of the collected data points must be found. Thus, we have to find the best-fitting curve using the least-squares method.





Let us assume that the given points of data are (x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_n, y_n) in which all x's are independent variables, while all y's are dependent ones. Also, suppose that $f(x)$ is the fitting curve and d represents the error or deviation from each given point.

Now, we can write:

$d_1 = y_1 - f(x_1)$

$d_2 = y_2 - f(x_2)$

.....

$d_n = y_n - f(x_n)$

The least-squares explain that the curve that best fits is represented by the property that the sum of squares of all the deviations from given values must be minimum, i.e:

$$S = \sum_{i=1}^{n} d_i^2$$

$$S = \sum_{i=1}^{n} [y_i - f_{x_i}]^2$$

$$S = d_1^2 + d_2^2 + d_3^2 + \cdots + d_n^2$$

Sum = Minimum Quantity

Suppose when we have to determine the equation of the line of best fit for the given data, then we first use the following formula.

The equation of the least square line is given by $Y = a + bX$

Normal equation for 'a':

$\sum Y = na + b\sum X$

Normal equation for 'b':

$\sum XY = a\sum X + b\sum X^2$

Solving these two normal equations we can get the required trend line equation.

Thus, we can get the line of best fit with the formula $y = ax + b$

3.2 Lagrange Interpolation Method:

The Lagrange interpolation formula is a method for determining a polynomial, known as a Lagrange polynomial, that takes on specific values at random places. Lagrange's interpolation is a polynomial approximation to f(x) of Nth degree.

So what exactly this formula is?

If we have n distinct real values x1,x2,.....xn and n real values y1,y2,....yn, which need not to be distinct, then there is a unique polynomial f with real coefficients that fulfil the following equation, f(xi)=yi for all i1,2.....n, with f's degree less than n. For different orders of polynomials, the formula is as follows:

Lagrange First order Interpolation Formula

The formula for the first order is:

$$f(x) = f(x0) + (x - x0)\frac{f(x0)-f(x1)}{x0-x1}$$

By further simplifying the above equation, we get

$$f(x) = \frac{(x-x1)}{(x0-x1)}f(x0) + \frac{(x-x0)}{(x1-x0)}f(x1)$$

Lagrange Second order Interpolation Formula

The formula for second order is:

$$f(x) = f(x0) + (x - x0)\frac{f(x0)-f(x1)}{x0-x1} + (x - x0)(x - x1)\frac{f(x0,x1)-f(x1,x2)}{x0-x2}$$

Collecting terms for f(x0),f(x1) and f(x2) together, and then simplifying the terms we get,

$$f(x) = \frac{(x-x1)(x-x2)}{(x0-x1)(x0-x2)}f(x0) + \frac{(x-x0)(x-x2)}{(x1-x0)(x1-x2)}f(x1) + \frac{(x-x0)(x-x1)}{(x2-x0)(x2-x1)}f(x2)$$

Lagrange Nth order Interpolation Formula

The formula for Nth order is:

$$f(x) = \frac{(x-x1)(x-x2).....(x-xn)}{(x0-x1)(x0-x2).....(x0-xn)}f(x0) + \frac{(x-x0)(x-x1).....(x-xn)}{(x1-x0)(x1-x2).....(x1-xn)}f(x1) + ... + \frac{(x-x0)(x-x1).....(x-xn-1)}{(xn-x0)(xn-x1).....(xn-xn-1)}f(xn))$$

3.3 Cubic Spline Method of Interpolation:

The method of building a spline $f: [x_1, x_{n+1}] \rightarrow R$ consisting of n polynomials of degree three, referred to as f1 to fn, is known as cubic spline interpolation. A spline is a piecewise polynomial-defined function. The interpolation function, in contrast to regression, covers all n+1 pre-defined nodes of a data collection D. The structure of the generated function is as follows:

$$f(x) = \begin{cases} a_1 x^3 + b_1 x^2 + c_1 x + d_1 & \text{if } x \in [x_1, x_2] \\ a_2 x^3 + b_2 x^2 + c_2 x + d_2 & \text{if } x \in (x_2, x_3] \\ \dots \\ a_n x^3 + b_n x^2 + c_n x + d_n & \text{if } x \in (x_n, x_{n+1}] \end{cases}.$$

Important Note:
The interpolation function is made up of polynomials which are only viable inside a given interval. Interpolation operates just inside the data limits, whereas extrapolation anticipates a development outside the range of the data.

Table of Notations:

Symbol	**Description**
D	n+1 data points, a set of ordered pairs (x,y)
(x_i,y_i)	ith element in D under the condition xi<xi+1
$f:[x_1,x_{n+1}] \rightarrow R$	It is the interpolation function which predicts new points and consists of n polynomials.
$f_i : [x_1,x_{i+1}] \rightarrow R$	ith polynomial of third-degree: $a_i x^3 + b_1 x^2 + c_i x + d_i$

Properties of a Cubic Spline:
To determine the 4n coefficients of all the polynomials in the spline, we must state all the equations. And every polynomial should pass from exactly two points. The equations, therefore, are

$$
\begin{aligned}
f_1(x_1) &= y_1 \\
f_1(x_2) &= y_2 \\
f_2(x_2) &= y_2 \\
f_2(x_3) &= y_3 \\
&\dots \\
f_n(x_n) &= y_n \\
f_n(x_{n+1}) &= y_{n+1}\ ,
\end{aligned}
$$

written-out as

$$
\begin{aligned}
a_1x_1^3 + b_1x_1^2 + c_1x_1 + d_1 &= y_1 \\
a_1x_2^3 + b_1x_2^2 + c_1x_2 + d_1 &= y_2 \\
a_2x_2^3 + b_2x_2^2 + c_2x_2 + d_2 &= y_2 \\
a_2x_3^3 + b_2x_3^2 + c_2x_3 + d_2 &= y_3 \\
&\dots \\
a_nx_n^3 + b_nx_n^2 + c_nx_n + d_n &= y_n \\
a_nx_{n+1}^3 + b_nx_{n+1}^2 + c_nx_{n+1} + d_n &= y_{n+1}\ ,
\end{aligned}
$$

When $x=x_1$ is fed in the first polynomial, it yields y_1 as the output, and at $x=x_2$ the second polynomial yields y_2. For the point, where the n-1th polynomial ends the nth polynomial starts.

Also, the first and second derivatives of all polynomials are equal at the point where they touch/intersect their neighbouring polynomial.

The first and second derivatives of the third-degree equation can be written as:

$$\frac{d}{dx}f_i(x) = 3a_i x^2 + 2b_i x + c_i$$

$$\frac{d^2}{dx^2}f_i(x) = 3a_i x^2 + 2b_i x + c_i$$

f_1 touches the polynomial f_2 at x_2

$$\begin{aligned}
\frac{d}{dx} f_1(x) &= \frac{d}{dx} f_2(x) \qquad |_{x=x_2} \\
\frac{d}{dx} f_2(x) &= \frac{d}{dx} f_3(x) \qquad |_{x=x_3} \\
&\cdots \\
\frac{d}{dx} f_{n-1}(x) &= \frac{d}{dx} f_n(x) \qquad |_{x=x_n} .
\end{aligned}$$

Evaluated at the corresponding x values, that gives

$$\begin{aligned}
3a_1 x_2^2 + 2b_1 x_2 + c_1 &= 3a_2 x_2^2 + 2b_2 x_2 + c_2 \\
3a_2 x_3^2 + 2b_2 x_3 + c_2 &= 3a_3 x_3^2 + 2b_3 x_3 + c_3 \\
&\cdots \\
3a_{n-1} x_n^2 + 2b_{n-1} x_n + c_{n-1} &= 3a_n x_n^2 + 2b_n x_n + c_n .
\end{aligned}$$

Equivalently, for the second derivative, the same is done by stating

$$\begin{aligned}
\frac{d^2}{dx^2} f_1(x) &= \frac{d^2}{dx^2} f_2(x) \qquad |_{x=x_2} \\
\frac{d^2}{dx^2} f_2(x) &= \frac{d^2}{dx^2} f_3(x) \qquad |_{x=x_3} \\
&\cdots \\
\frac{d^2}{dx^2} f_{n-1}(x) &= \frac{d^2}{dx^2} f_n(x) \qquad |_{x=x_n} ,
\end{aligned}$$

written-out as

$$\begin{aligned}
6a_1 x_2 + 2b_1 &= 6a_2 x_2 + 2b_2 \\
6a_2 x_3 + 2b_2 &= 6a_3 x_3 + 2b_3 \\
&\cdots \\
6a_{n-1} x_n + 2b_{n-1} &= 6a_n x_n + 2b_n .
\end{aligned}$$

This adds another 2n-2 equation.

Boundary Conditions:

Two more pieces of information are necessary in order to solve the system of equations. It's possible to utilise arbitrary limitations, such as setting the third derivative in the (say) fourth point to zero. The most typical way, however, is to choose a boundary condition that consists of a pair of equations. There are four different conditions namely: "natural spline," "not-a-knot spline," "periodic spline," and "quadratic spline." However, in our model, we have used the “natural spline” condition.

Natural Spline:

In the boundary points of the interpolation function, the natural spline is defined by putting the second derivative of the first and final polynomials to zero.

$$6a_1x_1 + 2b_1 = 0$$

$$6a_{n+1}x_{,n+1} + 2b_{n+1} = 0$$

The polynomial can be visually interpreted as the it's change of the steepness approaches to zero at the boundary points or the first and the last point.

3.4 Newton's Polynomial Interpolation:

This is another popular way to exactly fit data points of a set, the general for of a (n-1) order Newton's polynomial going through n points is:

$$f(x)=a_0+a_1(x-x_0)+a_2(x-x_0)(x-x1)+\cdots+a_n(x-x_0)(x-x_1)...(x-x_1)$$

Which can be also written as

$$f(x)=\sum_{i=0}^{n} a_i n_i(x)$$

where $n_i(x) = \prod_{j=0}^{i-1} (x-x_j)$

In newton's polynomial we can determine the coefficients a_i using the simple mathematical procedure. Since the polynomial goes through each data point, therefore for a data point (x_i,y_i), it'll be $f(x_i)=y_i$, thus,

$$f(x_0)= a_0=y_0$$

And $f(x_1)=a_0+a_1(x_1-x_0)=y_1$,

rearranging it to get a_1:

$$a_1=\frac{y1-y0}{x1-x0}$$

Inserting (x_2,y_2), to find a_2:

$$a_2 = \frac{\frac{y2-y1}{x2-x1} - \frac{y1-y0}{x1-x0}}{x2-x0}$$

Finding a_3:

$$a_3 = \frac{\frac{\frac{y2-y1}{x2-x1} - \frac{y1-y0}{x1-x0}}{x2-x0} - \frac{\frac{y2-y1}{x2-x1} - \frac{y1-y0}{x1-x0}}{x2-x0}}{x3-x0}$$

The patterns hence observed are called **divided differences** if we define:

$f[x_1,x_0] = \frac{y1-y0}{x1-x0}$

$f[x_2,x_1,x_0] = \frac{\frac{y1-y0}{x1-x0} - \frac{y1-y0}{x1-x0}}{x2-x0} = \frac{f[x2,x1]-f[x1,x0]}{x2-x1}$

Continuing the same, we will obtain:

$f[x_k,x_{k-1},\ldots\ldots,x_1,x_0] = \frac{f[xk,xk-1,\ldots\ldots,x1,x0] - f[xk,xk-1,\ldots\ldots,x1,x0]}{xk-x0}$

As we can observe, once the coefficients are determined, adding new data points will not change the calculated ones.
For example,

$$\begin{array}{llllll}
x_0 & y_0 & & & & \\
 & & f[x_1,x_0] & & & \\
x_1 & y_1 & & f[x_2,x_1,x_0] & & \\
 & & f[x_2,x_1] & & f[x_3,x_2,x_1,x_0] & \\
x_2 & y_2 & & f[x_3,x_2,x_1] & & f[x_4,x_3,x_2,x_1,x_0] \\
 & & f[x_3,x_2] & & f[x_4,x_3,x_2,x_1] & \\
x_3 & y_3 & & f[x_4,x_3,x_2] & & \\
 & & f[x_4,x_3] & & & \\
x_4 & y_4 & & & &
\end{array}$$

Each element in the table can be calculated using the two previous elements. Basically, storing the calculated elements in a diagonal matrix.

Thus, the coefficient matrix for a_0,a_1,a_2,a_3,a_4 will respectively be:

y_0	$f[x_1, x_0]$	$f[x_2, x_1, x_0]$	$f[x_3, x_2, x_1, x_0]$	$f[x_4, x_3, x_2, x_1, x_0]$
y_1	$f[x_2, x_1]$	$f[x_3, x_2, x_1]$	$f[x_4, x_3, x_2, x_1]$	0
y_2	$f[x_3, x_2]$	$f[x_4, x_3, x_2]$	0	0
y_3	$f[x_4, x_3]$	0	0	0
y_4	0	0	0	0

3.5 Machine Learning Method:

Machine Learning models like Logistic Regression, Decision Tree Classifier, K Neighbors Classifier, Linear Discriminant Analysis, SVC, and GaussianNB were used to predict these values. This model was applied to the **entire** dataset and took into account factors like rainfall, temperature and humidity for the entire year to predict the same four values.
The same Machine Learning methods were used with taking only Temperature as the factor as done in the other interpolation methods.

4. Algorithm and Python Code

4.1 Newton Interpolation

- The primary approach in this method was to create a coefficient matrix of the divided differences. For this purpose, a top to down approach was followed, and each subsequent entry was computed recursively using the entry of the previous column.
- On the computation of this matrix, the polynomial was computed using another function in which a for loop was run and the temperature of the given day was calculated by inputting it in the polynomial computed using divided differences.

```
def divided_diff(x, y):
    n = len(y)
    coef = np.zeros([n, n])
    coef[:,0] = y
    for j in range(1,n):
        for i in range(n-j):
            coef[i][j] = (coef[i+1][j-1] - coef[i][j-1]) / (x[i+j]-x[i])
    return coef
```

```
def newton_poly(coef, x_data, x):
    n = len(x_data) - 1
    p = coef[n]
    for k in range(1,n+1):
        p = coef[n-k] + (x -x_data[n-k])*p
    return p
```

4.2 Lagrange Interpolation

- This method involved running nested a loop for the square of the number of values run in the input data. After this for each iteration of the j loop wrt a single iteration in the i loop, the value of the polynomial was computed as a product of the interpolation at a given point. The value of the temperature was then computed for each polynomial value and added to in each subsequent iteration. Thus the error was minimized after n*n iterations.

```
def Lagrange_Interpolation(x,y,x_find):
  n = len(x)
  y_find = 0
  for i in range(n):
    p = 1
    for j in range(n):
      if i != j:
        p = p * (x_find - x[j])/(x[i] - x[j])
    y_find = y_find + p*y[i]
  return y_find
```

4.3 Cubic Spline Interpolation

- To implement this model, the matrices A, B, C were first calculated which are the coefficient matrix of the polynomial. These matrix coefficients were computed by writing the boundary conditions and inverting the matrix to yield the result using the delta of the dataset.
- When these matrices were set in the Ax = b form, they were sent to be computed using the Jacobian method. The results of these matrices were then obtained and the polynomial was written by multiplying the coefficient by the respective degree term.

```
def cubic_spline(x, y, tol = 1e-100):
    x = np.array(x)
    y = np.array(y)
    ### check if sorted
    if np.any(np.diff(x) < 0):
        idx = np.argsort(x)
        x = x[idx]
        y = y[idx]

    size = len(x)
    delta_x = np.diff(x)
    delta_y = np.diff(y)

    ### Get matrix A
    A = np.zeros(shape = (size,size))
    b = np.zeros(shape=(size,1))
    A[0,0] = 1
    A[-1,-1] = 1

    for i in range(1,size-1):
        A[i, i-1] = delta_x[i-1]
        A[i, i+1] = delta_x[i]
        A[i,i] = 2*(delta_x[i-1]+delta_x[i])
    ### Get matrix b
        b[i,0] = 3*(delta_y[i]/delta_x[i] - delta_y[i-1]/delta_x[i-1])

    ### Solves for c in Ac = b
    c = jacobi(A, b, np.zeros(len(A)), tol = tol, n_iterations=1000)

    ### Solves for d and b
    d = np.zeros(shape = (size-1,1))
    b = np.zeros(shape = (size-1,1))
    for i in range(0,len(d)):
        d[i] = (c[i+1] - c[i]) / (3*delta_x[i])
        b[i] = (delta_y[i]/delta_x[i]) - (delta_x[i]/3)*(2*c[i] + c[i+1])

    return b.squeeze(), c.squeeze(), d.squeeze()
```

```
def jacobi(A, b, x0, tol, n_iterations=300):
    n = A.shape[0]
    x = x0.copy()
    x_prev = x0.copy()
    counter = 0
    x_diff = tol+1

    while (x_diff > tol) and (counter < n_iterations): #iteration level
        for i in range(0, n): #element wise level for x
            s = 0
            for j in range(0,n): #summation for i !=j
                if i != j:
                    s += A[i,j] * x_prev[j]

            x[i] = (b[i] - s) / A[i,i]
        #update values
        counter += 1
        x_diff = (np.sum((x-x_prev)**2))**0.5
        x_prev = x.copy() #use new x for next iteration
    return x
```

```
def polynomial(x,y,x1,p,a):
  l = cubic_spline(x, y, tol = 1e-100)
  y1 = a + l[0][p-1]*(x1-x[p-1]) + l[1][p-1]*(x1-x[p-1])**2 + l[2][p-1]*(x1-x[p-1])**3
  return y1
```

4.4 Least Square Fitting

- For this method, the slope of the line was calculated as the product of the difference from the mean for both x and y, which was divided by the square of the difference between the mean and x. The value of the Temperature for the day was then calculated using the obtained slope and intercept.

```
def LeastSq(x,y,x1):
  mean_x = np.mean(x)
  mean_y = np.mean(y)
  n = len(x)
  numer = 0
  denom = 0
  for i in range(n):
    numer += (x[i] - mean_x) * (y[i] - mean_y)
    denom += (x[i] - mean_x) ** 2
    m = numer / denom
    c = mean_y - (m * mean_x)
  y = m*x1 + c
  return y
```

4.5 Machine Learning Implementation

1. **Taking temp as the only factor the data depends on:**

```
#error when taking only Max Temperature and Days
clf = LogisticRegression()
clf.fit(X_train, Y_train.astype(int))
print("LogisticRegression")
y_pred= clf.predict(X_test)
print(mean_squared_error(Y_test, y_pred, squared=False))

clf = DecisionTreeClassifier()
clf.fit(X_train, Y_train.astype(int))
print("DecisionTreeClassifier")
y_pred= clf.predict(X_test)
print(mean_squared_error(Y_test, y_pred, squared=False))

clf = KNeighborsClassifier()
clf.fit(X_train, Y_train.astype(int))
print("KNeighborsClassifier")
y_pred = clf.predict(X_test)
print(mean_squared_error(Y_test, y_pred, squared=False))

clf = LinearDiscriminantAnalysis()
clf.fit(X_train, Y_train.astype(int))
print("LinearDiscriminantAnalysis")
y_pred = clf.predict(X_test)
print(mean_squared_error(Y_test, y_pred, squared=False))

clf = GaussianNB()
clf.fit(X_train, Y_train.astype(int))
print("GaussianNB")
y_pred = clf.predict(X_test)
print(mean_squared_error(Y_test, y_pred, squared=False))

clf = SVC()
clf.fit(X_train, Y_train.astype(int))
print("SVC")
y_pred = clf.predict(X_test)
print(mean_squared_error(Y_test, y_pred, squared=False))
```

2. **Taking into account the other factors:**

- The data was one hot encoded in the Rainfall and Humidity columns, this data was then run in popular machine learning models, and their RMSE was computed.

```
one_hot_encoded_data = pd.get_dummies(df, columns = ['Rainfall', 'Evaporation'])
df = one_hot_encoded_data
df.insert(0, 'MaxTemp', df.pop('MaxTemp'))
X = df.iloc[:,1:4]
Y = df.iloc[:,0]
df
```

```
clf = LogisticRegression()
clf.fit(X_train, Y_train.astype(int))
print("LogisticRegression")
y_pred= clf.predict(X_test)
print(mean_squared_error(Y_test, y_pred, squared=False))

clf = DecisionTreeClassifier()
clf.fit(X_train, Y_train.astype(int))
print("DecisionTreeClassifier")
y_pred= clf.predict(X_test)
print(mean_squared_error(Y_test, y_pred, squared=False))

clf = KNeighborsClassifier()
clf.fit(X_train, Y_train.astype(int))
print("KNeighborsClassifier")
y_pred = clf.predict(X_test)
print(mean_squared_error(Y_test, y_pred, squared=False))

clf = LinearDiscriminantAnalysis()
clf.fit(X_train, Y_train.astype(int))
print("LinearDiscriminantAnalysis")
y_pred = clf.predict(X_test)
print(mean_squared_error(Y_test, y_pred, squared=False))

clf = GaussianNB()
clf.fit(X_train, Y_train.astype(int))
print("GaussianNB")
y_pred = clf.predict(X_test)
print(mean_squared_error(Y_test, y_pred, squared=False))

clf = SVC()
clf.fit(X_train, Y_train.astype(int))
print("SVC")
y_pred = clf.predict(X_test)
print(mean_squared_error(Y_test, y_pred, squared=False))
```

The dataset and codes for all the models implemented here can be found here.

5. Results and Discussion

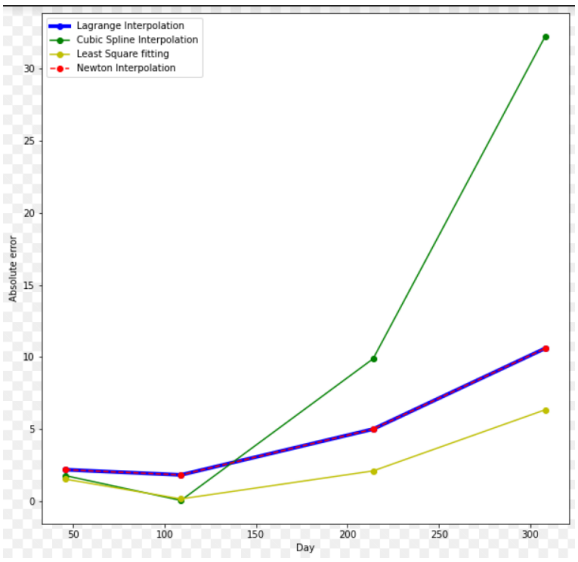
5.1 Error Analysis:

i) We selected four random points from the given dataset, using the random function in python. These four data points are: [46,109,214,308].

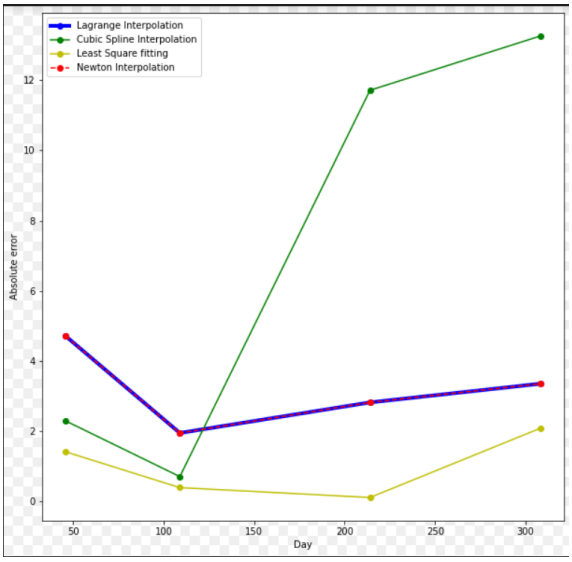
ii) We took different intervals of days near these points and calculated the temperature at these points using different interpolation techniques. The different interpolation techniques were: “Newton Interpolation”, “Lagrange Interpolation”, “Cubic Spline Interpolation”, and “Least Square fitting”.

iii) We then calculated the error for every interpolation in different intervals for the whole dataset.

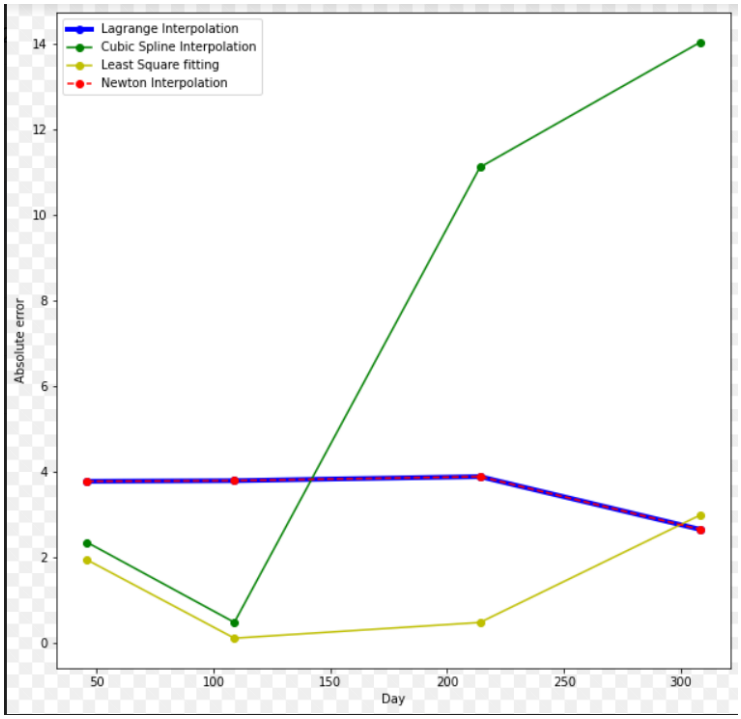
iv) The error in the 4-day interval was:



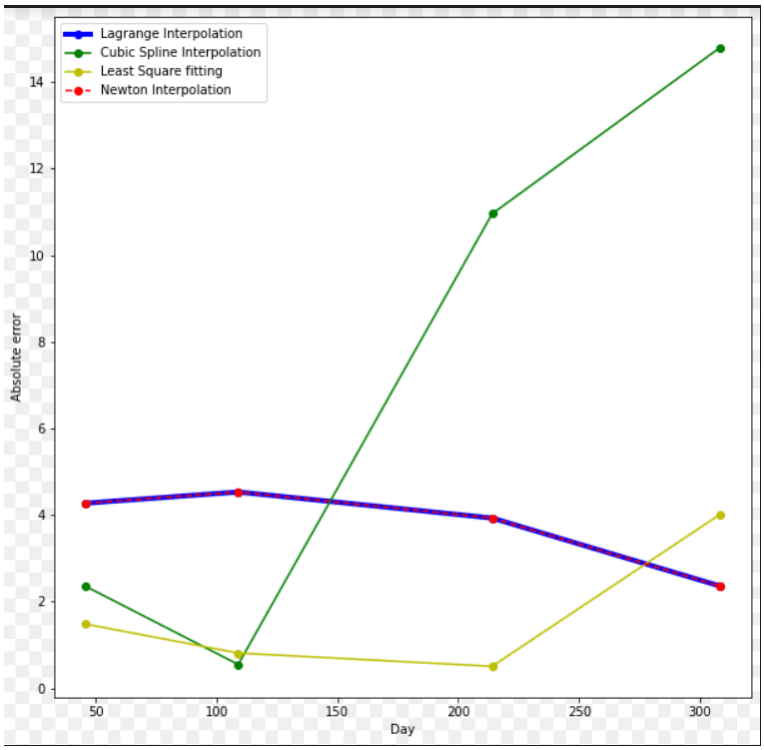
v) The error in the 7-day interval was:



vi) The error in the 10-day interval was:



vii) The error in the fortnight interval was:



viii) From the above four graphs, it is quite clear that the method which has the maximum error is the Cubic Spline method. This is because of the reason that, in the cubic spline method, it converts the given dataset into a smooth cubic curve. But the dataset that we have is real-time weather, which is very erratic so it can't be converted into a smooth cubic curve. So, therefore it is not correct to journalize such erratic data into a smooth cubic curve.

ix) The error in the Newton method and Lagrange method is the same as the difference in both the method only lies in the computational aspect. The use of nested multiplication and the relative ease of adding more data points for higher-order interpolating polynomials is the only advantage of Newton's interpolation over the Lagrange method.

x) For such fluctuating data, making trends is not good, as it leaves us with a lot of errors in the data. Therefore it is best to plot a line, as the square of the distances of points from the line is the minimum. Hence the least square fitting is the best method.

xi) RMSE error values of different methods

Method	**4-day interval**	**7-day interval**	**10-day interval**	**14-day interval**
Newton Interpolation	15.623239900865638	9.25665207889713	9.374716099673819	6.986638949359213
Lagrange Interpolation	15.623239900865629	9.25665207889713	9.374716099673826	6.986638949359211
Cubic Spline Interpolation	27.56979532326306	50.41442592842062	52.19874928468699	52.121808484764685
Least Squares Fitting	2.4341765411887177	2.0989710269994846	2.859426736855427	3.464424463129617

xii) Root Mean Square for different ML Models:

1. **Taking temp as the only factor the data depends on**

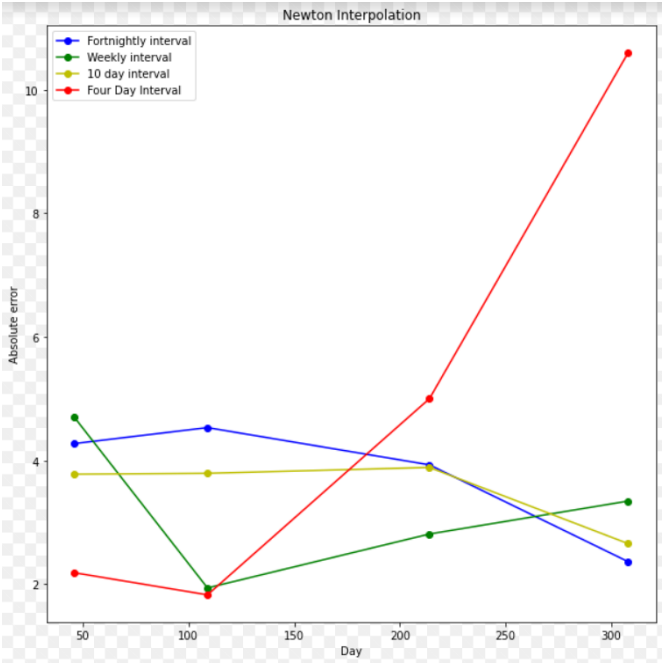
Method	**RMSE**
Logistic Regression	5.458207308908945
DecisionTreeClassifier	3.2470568794492003
K Neighbors Classifier	4.415895734178417
Linear Discriminant Analysis	5.458207308908945
SVC	5.672849038729677
Gaussian NB	5.184970483900456

2. **Taking other factors into account**

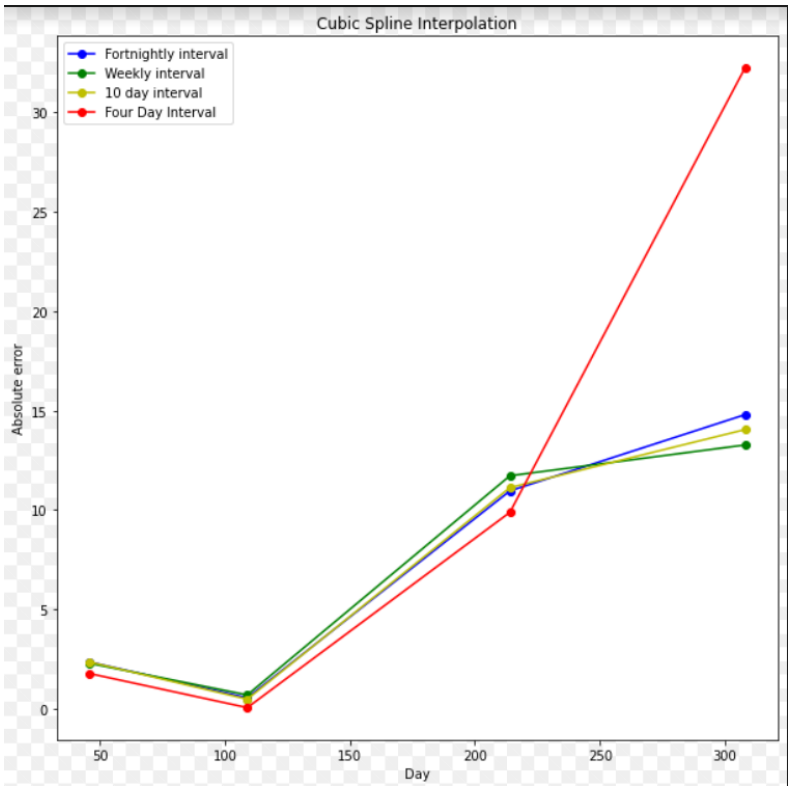
Method	**RMSE**
Logistic Regression	6.025407914866888
DecisionTreeClassifier	3.6495279220817114
K Neighbors Classifier	4.450341620655715
Linear Discriminant Analysis	5.611775746323709
SVC	5.531983174037846
Gaussian NB	6.896150512560751

If we don't feed the other factors into consideration in the ML model yet we observed low values of error compared to the interpolation methods.

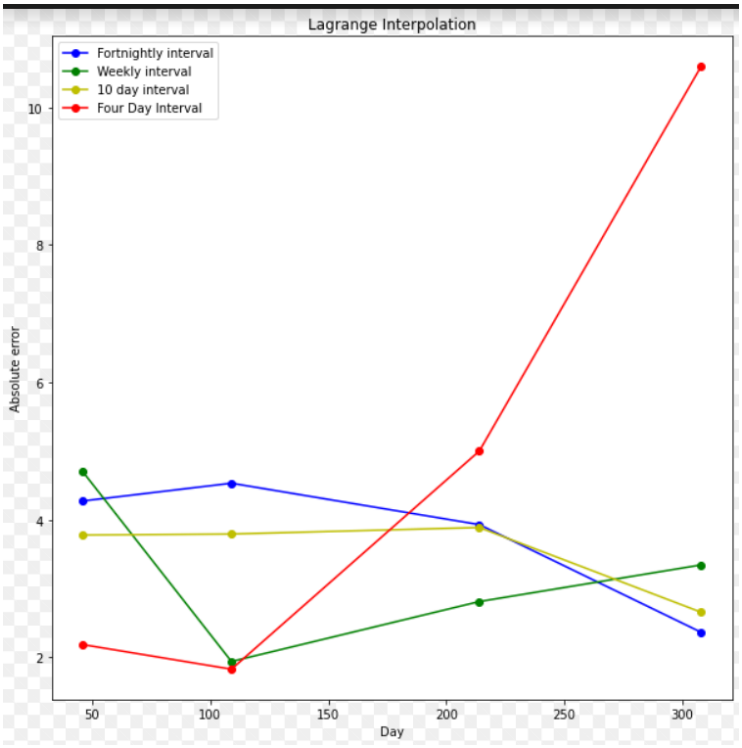
xiii) The error in the Newton method in different intervals is:



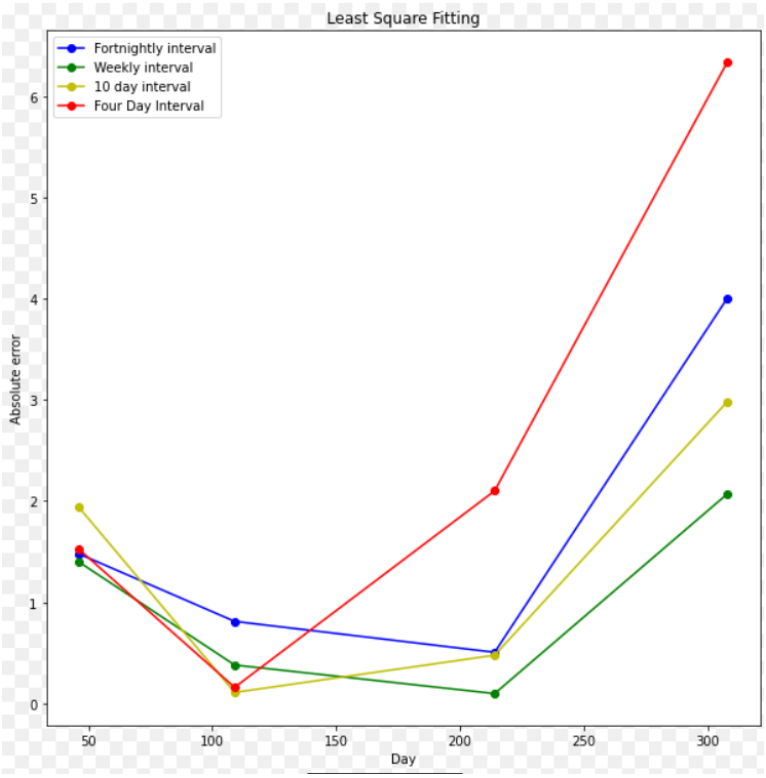
xiv) The error in the Cubic Spline method in different intervals is:



xv) The error in the Lagrange method in different intervals is:

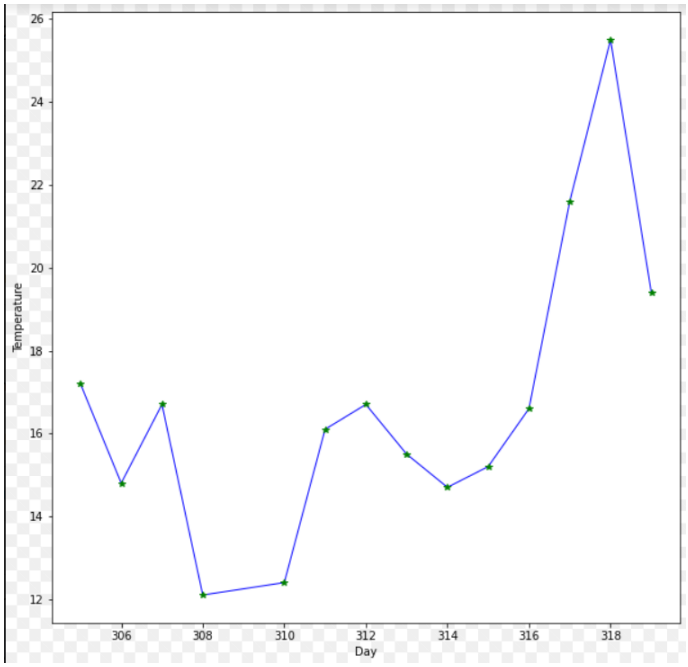


xvi) The error in the least-squares method in different intervals is:

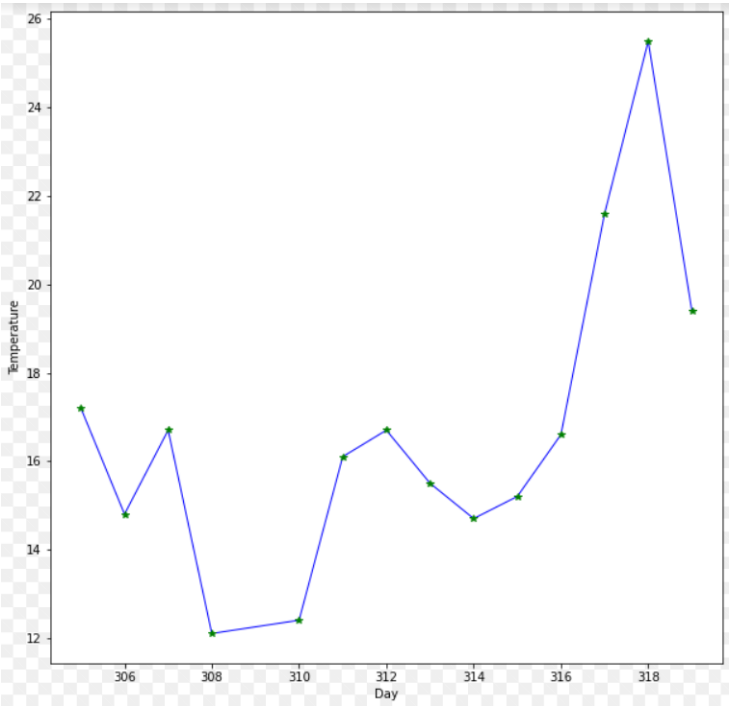


5.2 Polynomial Analysis:

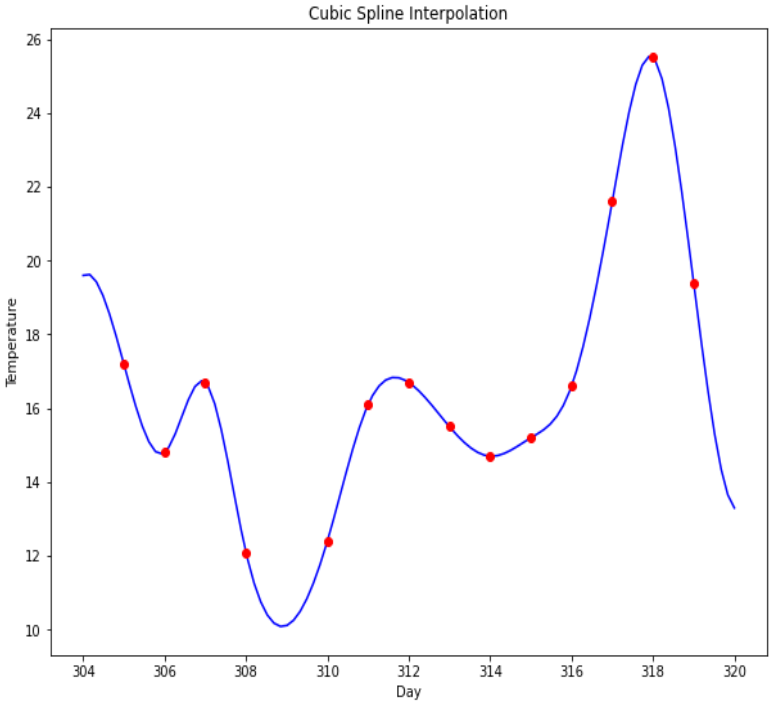
i) The polynomial we obtained by using Newton Method looks like this because in the Newton method we predict the values between the two points by connecting them with lines.



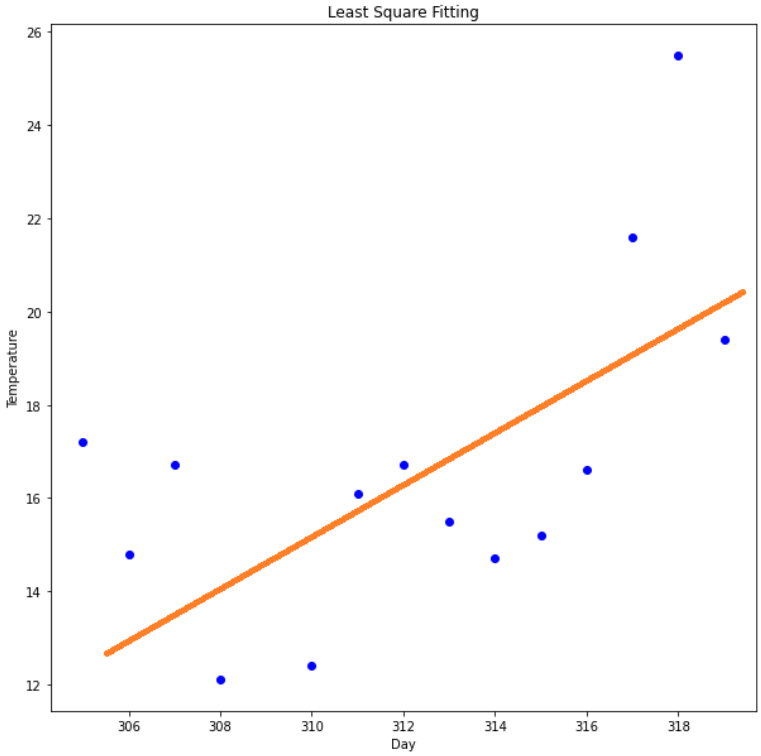
ii) The polynomial we obtained by using Lagrange Method looks like this because in the Lagrange method we predict the values between the two points by connecting them with lines.



iii) The polynomial we obtained by using a cubic spline looks like this because in this we join two points by a cubic function. At the ends, the first and second derivatives of the function are also along with its value.



iv) The graph that we obtained while performing the least square method looks like this because in it the square of the distances of all the points from a single line is minimum. Which in this case is the orange line.



6. Conclusion

In this project we used a real-life data set and implemented interpolation models on it, through which we could get a comparison of errors and the polynomials predicted by each of them. The data-set used here was the weather data of Chicago for the year 1995, from which random four points were predicted and then compared by the actual values. The range of input data varied from four days to fourteen days thus, we also got to see how this affected the predictions.

According to our analysis, we found out that in linear interpolation methods error was most in cubic spline and least in the least square method. This is major because the data inputted wasn't linear rather was very irregular thus, led to a huge error in cubic spline which is mostly used for non-linear inputs. The graph for Newton's and Lagrange had points linearly connected while the cubic spine method connected points with cubical functions and the least square method predicted the best fit curve for the data.

We can also observe that the error obtained via the Machine Learning model is greater than the least square methods. This is because we have applied the Machine learning model to the entire data set of 365 days while the least square method and other interpolation methods are applied to a set of data ranging from four to fourteen days. Thus, in the long run, the Machine Learning model will provide more accurate results thus, it's used extensively for weather-forecasting purposes due to its ability to take all factors into consideration simultaneously.

On changing the intervals of consideration we expected to see less error with an increasing set of input. This is observed to some extent in Newton's and Lagrange's method, but as we are dealing with data on weather it is not only erratic and unpredictable but also varies on various factors that the linear interpolation doesn't take into account. If our ML model take only temperature as the factor we still observe a comparatively less error this is mainly due to the presence of patterns and trends in the weather data over the year. As the model is able to get a general sense of the pattern in an interval and can be applied more accurately ahead, which is not possible with the input set of 4 days or 14 days.

The dataset and codes for all the models implemented here can be found here.

7. Bibliography (in Chicago style)

- Investopedia. 2022. *Understanding the Least Squares Method*. [online] Available at: <https://www.investopedia.com/terms/l/least-squares-method.asp> [Accessed 24 April 2022].
- Dmpeli.math.mcmaster.ca. 2022. *Lecture 2-2: Newton divided difference polynomials*. [online] Available at: <https://dmpeli.math.mcmaster.ca/Matlab/Math4Q3/NumMethods/Lecture2-2.html#:~:text=The%20difference%20between%20Newton%20and,for%20higher%2Dorder%20interpolating%20polynomials.> [Accessed 24 April 2022].
- Denk, T., 2022. *Cubic Spline Interpolation – Timo Denk's Blog*. [online] Timodenk.com. Available at: <https://timodenk.com/blog/cubic-spline-interpolation/#:~:text=Cubic%20spline%20interpolation%20is%20a,of%20multiple%20cubic%20piecewise%20polynomials.> [Accessed 24 April 2022].
- Medium. 2022. Natural Cubic Splines Implementation with Python. [online] Available at: <https://medium.com/eatpredlove/natural-cubic-splines-implementation-with-python-edf68feb57aa> [Accessed 24 April 2022].
- *Creating a Model for Weather Forecasting Using Linear Regression*. [online] Available at: <https://medium.com/swlh/creating-a-model-for-weather-forecasting-using-linear-regression-b18c1590e8d7> [Accessed 24 April 2022].
- How to Interpolate Data with Scipy. [online] Available at: <https://medium.com/productive-data-science/how-to-interpolate-data-with-scipy-d314143285bc> [Accessed 24 April 2022].