Missing value imputation for titanic dataset using mean imputation technique in openmp and plotted a graph for execution time Vs number of threads.

1. Origin: The Titanic dataset is a popular dataset used for data analysis and machine learning tasks. It contains information about the passengers aboard the RMS Titanic, which sank on its maiden voyage in 1912. The dataset is often used as a teaching and learning tool for beginners in data science.
2. Variables: The dataset provides various variables for each passenger, including their age, sex, ticket class (1st, 2nd, or 3rd class), number of siblings/spouses aboard, number of parents/children aboard, fare paid, cabin number, embarkation point (Cherbourg, Queenstown, or Southampton), and whether the passenger survived the disaster (0 = No, 1 = Yes).
3. Size: The Titanic dataset typically consists of 891 rows or instances, representing the known passengers of the Titanic. Each row corresponds to a passenger, and the dataset includes 12 columns or features, providing information about different aspects of the passengers.
4. Missing Data: The Titanic dataset may contain missing values, especially in the "Age" column. Data scientists often employ various techniques to handle missing data, such as imputation or exclusion, depending on the analysis or modeling task at hand.

Link of the dataset: https://github.com/kithinji007/EDA-MissingValues-titanic-dataset/blob/main/titanic.csv

///script for extraction of age from the titanic dataset and removing NaN values.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


#define MAX_LINE_LENGTH 1000

#define MAX_FIELD_LENGTH 100


int main() {

    // Open the Titanic dataset file

    FILE* file = fopen("titanic.csv", "r");

    if (file == NULL) {

        printf("Failed to open the file.\n");

        return 1;

    }


    // Read and discard the header line

    char header[MAX_LINE_LENGTH];

    fgets(header, MAX_LINE_LENGTH, file);
```

```
// Read and process the data
char line[MAX_LINE_LENGTH];
while (fgets(line, MAX_LINE_LENGTH, file) != NULL) {
    // Extract the 1st and 6th columns
    char* field;
    char* token = strtok(line, ",");
    int columnCount = 0;

    while (token != NULL) {
        columnCount++;

        if (columnCount == 7) {
            field = token;
            printf("%s\t", field);  // Print the field value
        }
```


```
// Read and process the data
char line[MAX_LINE_LENGTH];
while (fgets(line, MAX_LINE_LENGTH, file) != NULL) {
    // Extract the 1st and 6th columns
    char* field;
    char* token = strtok(line, ",");
    int columnCount = 0;

    while (token != NULL) {
        columnCount++;

        if (columnCount == 7) {
            field = token;
            printf("%s\t", field);  // Print the field value
        }
```

```
            token = strtok(NULL, ",");
        }

        printf("\n");  // Move to the next line after printing the columns
    }

    // Close the file
    fclose(file);

    return 0;
}
awk -F',' 'NR==1 || (tolower($0) !~ /nan/)' age.csv > age_without_nan.csv
```

This above script saves the age_without_nan.csv file which is not having any NaN values, which makes it easier for mean computation.

Mean imputation:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <omp.h>
#include <time.h>

#define MAX_VALUES 1000
#define NUM_THREADS 16

// Function to calculate the mean
double calculateMean(int values[], int count) {
    int sum = 0;
    for (int i = 0; i < count; i++) {
        sum += values[i];
    }
    return (double)sum / count;
```

```
}

void replaceNaN(double mean) {
    FILE* file = fopen("titanic.csv", "r");
    FILE* output = fopen("titanic_output.csv", "w");

    if (file == NULL || output == NULL) {
        printf("Failed to open the file.\n");
        return;
    }

    char line[100];

    // Copy the header line to the output file
    fgets(line, sizeof(line), file);
    fputs(line, output);

    // Replace NaN values with the mean and write to the output file
    while (fgets(line, sizeof(line), file) != NULL) {
        char* token = strtok(line, ",");

        while (token != NULL) {
            if (strcmp(token, "NaN") == 0) {
                fprintf(output, "%.lf,", mean);
            } else {
                fprintf(output, "%s,", token);
            }

            token = strtok(NULL, ",");
        }

        fseek(output, -1, SEEK_CUR);
        fputs("\n", output);
```

```
    }

    fclose(file);
    fclose(output);
}

int main() {
    // Read the values from the file
    int values[MAX_VALUES];
    int count = 0;

    FILE* file = fopen("age_without_nan.csv", "r");
    if (file == NULL) {
        printf("Failed to open the file.\n");
        return 1;
    }

    char line[100];
    while (fgets(line, sizeof(line), file) != NULL) {
        int value = atoi(line);
        values[count++] = value;
    }

    fclose(file);

    // Calculate the mean
    double mean = calculateMean(values, count);
    printf("Mean: %.lf\n", mean);

    // Record execution time and number of threads
    double start_time = omp_get_wtime();

    // Replace NaN values in titanic.csv with the mean using OpenMP
```

```
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        replaceNaN(mean);
    }

    double end_time = omp_get_wtime();
    double cpu_time = end_time - start_time;

    printf("Mean imputation completed in %f seconds.\n", cpu_time);

    // Generate a graph showing time vs. threads
    FILE* output_file = fopen("time_vs_threads.dat", "w");
    if (output_file == NULL) {
        fprintf(stderr, "Error opening output file.\n");
        return 1;
    }

    fprintf(output_file, "Threads Time\n");
    for (int i = 1; i <= NUM_THREADS; i++) {
        fprintf(output_file, "%d %f\n", i, cpu_time / i);
    }

    fclose(output_file);

    printf("Time vs. Threads saved to time_vs_threads.dat\n");

    return 0;
}
```

The above script takes age_without_nan.csv for calculating mean and imputes it into the titanic.csv creating a new file name titanic_output.csv. It also creates time_vs_threads.dat which contains Number of threads and their execution time.

The time_vs_threads.dat file contents are as below :

```
 GNU nano 2.3.1                                  File: time_vs_threads.dat


1 0.110788
2 0.055394
3 0.036929
4 0.027697
5 0.022158
6 0.018465
7 0.015827
8 0.013848
9 0.012310
10 0.011079
11 0.010072
12 0.009232
13 0.008522
14 0.007913
15 0.007386
16 0.006924
```

Plot_ script:

```
set terminal dumb

set xlabel "Number of Threads"

set ylabel "Execution Time (seconds)"

set title "Execution Time vs. Number of Threads"

plot "time_vs_threads.dat" with lines
```

The above code is a script written in the gnuplot language to generate a simple ASCII plot of the "time_vs_threads.dat" data file. It sets the terminal to ASCII output, defines the x and y labels, and plots the data from the file as lines on a graph to visualize the execution time versus the number of threads used in the program's parallel execution.

Plot for Execution Time vs. Number of Threads:

```
[dmcncourse@ui7 golden]$ ./try
Mean: 30
Mean imputation completed in 0.110788 seconds.
Time vs. Threads saved to time_vs_threads.dat
[dmcncourse@ui7 golden]$ nano time_vs_threads.dat
[dmcncourse@ui7 golden]$ gnuplot plot_script.gnu
```