SafeHarborInsights Report

Ema Čikotić

89221136@student.upr.si

1 Introduction

In the course of my Machine Learning and Data Mining project, I selected a dataset from Kaggle, titled *LA Crime Rates from 2020 to 2023*, which is accessible at the following link: LA Crime Rates Dataset.

This report presents the methodology, results, and challenges encountered while applying the K-means clustering algorithm to identify the safest areas in Los Angeles. The analysis focuses on overall crime rates and the distribution of crimes by gender.

2 Methodology

2.1 Understanding the Dataset

The dataset contains comprehensive records of crime incidents reported in Los Angeles over a three-year period. Key attributes of the dataset include:

- **Date and Time**: The specific date and time when each crime occurred.
- **Area**: The geographic location where the incident took place.
- **Crime Type**: The nature and classification of the offense committed.
- **Victim's Gender**: The gender of the individual who was victimized in each incident.

2.2 Setting Up the Environment

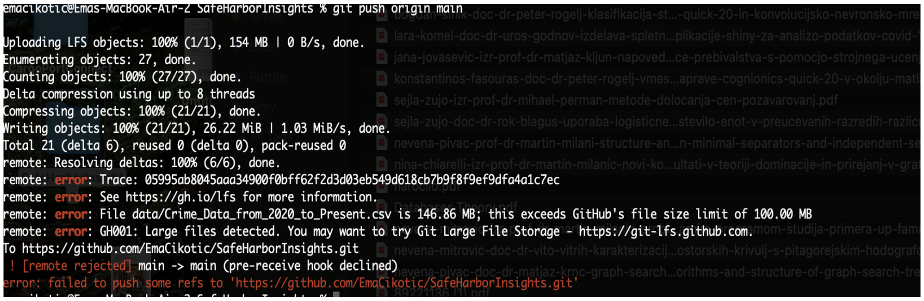
For the implementation, I utilized TypeScript within the Visual Studio Code (VS Code) development environment. Dependency management was facilitated through npm (Node Package Manager). The primary package employed was:

- `csv-parser`: For parsing and reading CSV files.

2.3 Preprocessing and Handling Large Datasets

Preprocessing involved data cleaning and the selection of relevant features. The dataset's substantial size (149MB) made it unsuitable for Git due to its 100MB file size limitation.

To address this challenge:

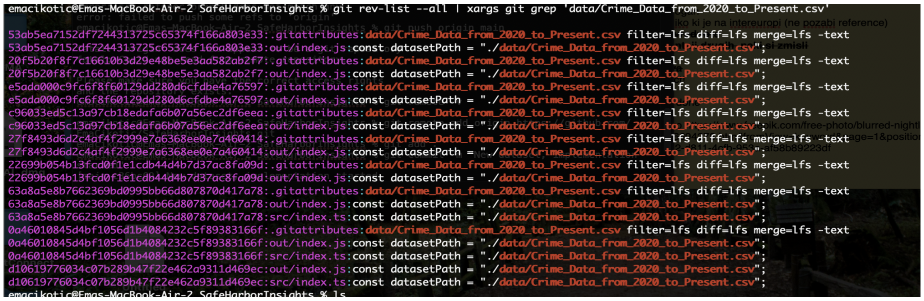


```
emacikotic@Emas-MacBook-Air-2 SafeHarborInsights % git push origin main

Uploading LFS objects: 100% (1/1), 154 MB | 0 B/s, done.
Enumerating objects: 27, done.
Counting objects: 100% (27/27), done.
Delta compression using up to 8 threads
Compressing objects: 100% (21/21), done.
Writing objects: 100% (21/21), 26.22 MiB | 1.03 MiB/s, done.
Total 21 (delta 6), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (6/6), done.
remote: error: Trace: 05995ab8045aaa34900f0bff62f2d3d03eb549d618cb7b9f8f9ef9dfa4a1c7ec
remote: error: See https://gh.io/lfs for more information.
remote: error: File data/Crime_Data_from_2020_to_Present.csv is 146.86 MB; this exceeds GitHub's file size limit of 100.00 MB
remote: error: GH001: Large files detected. You may want to try Git Large File Storage - https://git-lfs.github.com.
To https://github.com/EmaCikotic/SafeHarborInsights.git
 ! [remote rejected] main -> main (pre-receive hook declined)
error: failed to push some refs to 'https://github.com/EmaCikotic/SafeHarborInsights.git'
emacikotic@Emas-MacBook-Air-2 SafeHarborInsights %
```

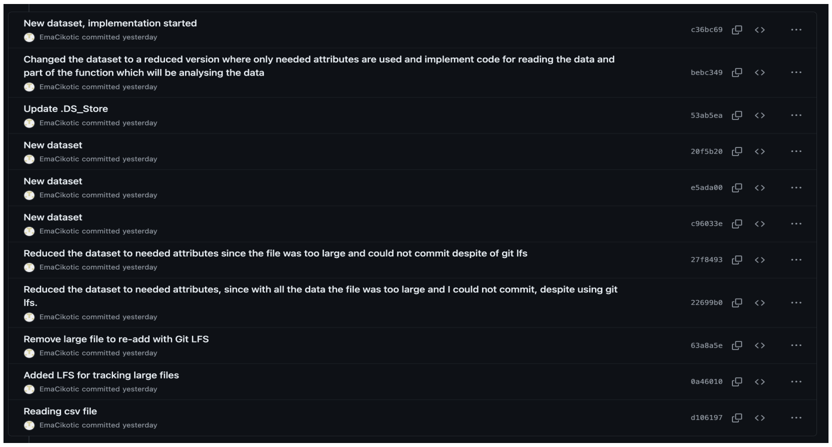
Figure 1: File size exceeds Git limitations

Attempts to utilize Git Large File Storage were unsuccessful. As a result, my Git commit history became cluttered with multiple similar commits due to repeated attempts to commit the dataset.



```
emacikotic@Emas-MacBook-Air-2 SafeHarborInsights % git rev-list --all | xargs git grep 'data/Crime_Data_from_2020_to_Present.csv'
53ab5ea7152df7244313725c65374f166a803e33:.gitattributes:data/Crime_Data_from_2020_to_Present.csv filter=lfs diff=lfs merge=lfs -text
53ab5ea7152df7244313725c65374f166a803e33:out/index.js:const datasetPath = "./data/Crime_Data_from_2020_to_Present.csv";
20f5b20f8f7c16610b3d29e48be5e3aa582ab2f7:.gitattributes:data/Crime_Data_from_2020_to_Present.csv filter=lfs diff=lfs merge=lfs -text
20f5b20f8f7c16610b3d29e48be5e3aa582ab2f7:out/index.js:const datasetPath = "./data/Crime_Data_from_2020_to_Present.csv";
e5ada000c9fc6f8f60129dd280d6cfdbe4a76597:.gitattributes:data/Crime_Data_from_2020_to_Present.csv filter=lfs diff=lfs merge=lfs -text
e5ada000c9fc6f8f60129dd280d6cfdbe4a76597:out/index.js:const datasetPath = "./data/Crime_Data_from_2020_to_Present.csv";
c96033ed5c13a97cb18edafa6b07a56ec2df6eea:.gitattributes:data/Crime_Data_from_2020_to_Present.csv filter=lfs diff=lfs merge=lfs -text
c96033ed5c13a97cb18edafa6b07a56ec2df6eea:out/index.js:const datasetPath = "./data/Crime_Data_from_2020_to_Present.csv";
27f8493d6d2c4af4f2999e7a6368ee0e7a460414:.gitattributes:data/Crime_Data_from_2020_to_Present.csv filter=lfs diff=lfs merge=lfs -text
27f8493d6d2c4af4f2999e7a6368ee0e7a460414:out/index.js:const datasetPath = "./data/Crime_Data_from_2020_to_Present.csv";
22699b054b13fcd0f1e1cdb44d4b7d37ac8fa09d:.gitattributes:data/Crime_Data_from_2020_to_Present.csv filter=lfs diff=lfs merge=lfs -text
22699b054b13fcd0f1e1cdb44d4b7d37ac8fa09d:out/index.js:const datasetPath = "./data/Crime_Data_from_2020_to_Present.csv";
63a8a5e8b7662369bd0995bb66d807870d417a78:.gitattributes:data/Crime_Data_from_2020_to_Present.csv filter=lfs diff=lfs merge=lfs -text
63a8a5e8b7662369bd0995bb66d807870d417a78:out/index.js:const datasetPath = "./data/Crime_Data_from_2020_to_Present.csv";
63a8a5e8b7662369bd0995bb66d807870d417a78:src/index.ts:const datasetPath = "./data/Crime_Data_from_2020_to_Present.csv";
0a46010845d4bf1056d1b4084232c5f89383166f:.gitattributes:data/Crime_Data_from_2020_to_Present.csv filter=lfs diff=lfs merge=lfs -text
0a46010845d4bf1056d1b4084232c5f89383166f:out/index.js:const datasetPath = "./data/Crime_Data_from_2020_to_Present.csv";
0a46010845d4bf1056d1b4084232c5f89383166f:src/index.ts:const datasetPath = "./data/Crime_Data_from_2020_to_Present.csv";
d10619776034c07b289b47f22e462a9311d469ec:out/index.js:const datasetPath = "./data/Crime_Data_from_2020_to_Present.csv";
d10619776034c07b289b47f22e462a9311d469ec:src/index.ts:const datasetPath = "./data/Crime_Data_from_2020_to_Present.csv";
emacikotic@Emas-MacBook-Air-2 SafeHarborInsights % ls
```

Figure 2: Cluttered Git commit history



New dataset, implementation started
EmaCikotic committed yesterday
c36bc69

Changed the dataset to a reduced version where only needed attributes are used and implement code for reading the data and part of the function which will be analysing the data
EmaCikotic committed yesterday
bebc349

Update .DS_Store
EmaCikotic committed yesterday
53ab5ea

New dataset
EmaCikotic committed yesterday
20f5b20

New dataset
EmaCikotic committed yesterday
e5ada00

New dataset
EmaCikotic committed yesterday
c96033e

Reduced the dataset to needed attributes since the file was too large and could not commit despite of git lfs
EmaCikotic committed yesterday
27f8493

Reduced the dataset to needed attributes, since with all the data the file was too large and I could not commit, despite using git lfs.
EmaCikotic committed yesterday
22699b0

Remove large file to re-add with Git LFS
EmaCikotic committed yesterday
63a8a5e

Added LFS for tracking large files
EmaCikotic committed yesterday
0a46010

Reading csv file
EmaCikotic committed yesterday
d106197

Figure 3: Multiple commits

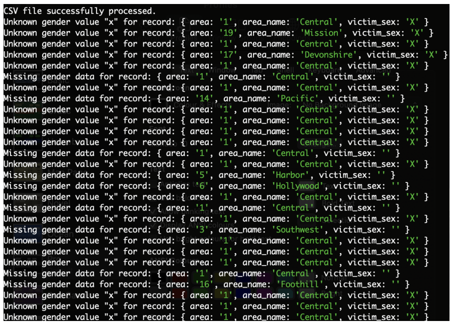
To resolve the issue of the oversized file in my Git commit history, I executed the following command:

```
git filter-repo --path data/Crime_Data_from_2020_to_Present.csv --invert-paths
```

This command allowed me to remove the large file from the history and proceed with the project without unnecessary complications.

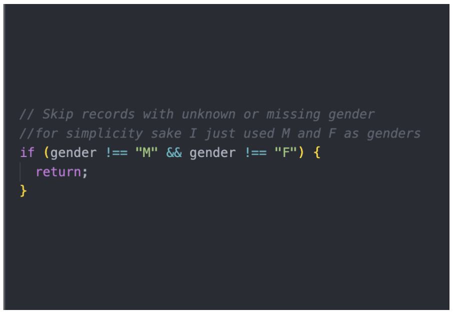
To effectively manage the large dataset, I took the following steps:

- **Feature Selection:** I concentrated on the essential attributes for analysis, specifically the area and victim's gender.
- **Gender Attribute Filtering:** The dataset contained multiple categories for gender (e.g., "M" for Male, "F" for Female, along with "X", "H", blank entries, and "-"). I filtered these to retain only "M" and "F" to maintain relevance in the gender-based safety analysis.



```
CSV file successfully processed.
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
Unknown gender value "x" for record: { area: '19', area_name: 'Mission', victim_sex: 'X' }
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
Unknown gender value "x" for record: { area: '17', area_name: 'Devonshire', victim_sex: 'X' }
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
Missing gender data for record: { area: '1', area_name: 'Central', victim_sex: '' }
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
Missing gender data for record: { area: '14', area_name: 'Pacific', victim_sex: '' }
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
Missing gender data for record: { area: '1', area_name: 'Central', victim_sex: '' }
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
Missing gender data for record: { area: '5', area_name: 'Harbor', victim_sex: '' }
Missing gender data for record: { area: '6', area_name: 'Hollywood', victim_sex: '' }
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
Missing gender data for record: { area: '1', area_name: 'Central', victim_sex: '' }
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
Missing gender data for record: { area: '3', area_name: 'Southwest', victim_sex: '' }
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
Missing gender data for record: { area: '1', area_name: 'Central', victim_sex: '' }
Missing gender data for record: { area: '16', area_name: 'Foothill', victim_sex: '' }
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
Unknown gender value "x" for record: { area: '1', area_name: 'Central', victim_sex: 'X' }
```

Figure 4: Issues before filtering



```
// Skip records with unknown or missing gender
//for simplicity sake I just used M and F as genders
if (gender !== "M" && gender !== "F") {
  return;
}
```

Figure 5: Skipping record if gender is not male or female

3 Code Explanation

The following section explains the TypeScript code used to preprocess the dataset, perform K-means clustering, and analyze the results.

3.1 Reading and Parsing the CSV File

We start by importing necessary modules and defining an interface for the structure of each crime record:

```
import * as fs from "fs";
import csv from "csv-parser";

interface CrimeRecord {
  area: string;
  area_name: string;
  victim_sex: string;
}

const datasetPath = "./data/Crime_Data_from_2020_to_Present_reduced.csv";
const crimeData: CrimeRecord[] = [];

// Read the CSV file
fs.createReadStream(datasetPath)
  .pipe(csv())
  .on("data", (row: CrimeRecord) => {
    crimeData.push(row);
  })
  .on("end", () => {
    console.log("CSV file successfully processed.");
    dataAnalysis();
  });
```

Explanation: This segment imports the necessary libraries for file handling and CSV parsing. An interface, `CrimeRecord`, defines the structure of each crime record, specifying that it contains the area, area name, and victim's gender. The dataset path is defined, and an empty array `crimeData` is initialized to store the parsed records. The code reads the CSV file, processes each row, and stores the data in the `crimeData` array. Once all data is read, the `dataAnalysis()` function is invoked.

3.2 Data Analysis

In the `dataAnalysis` function, we process the data to count crimes by area and by gender:

```
function dataAnalysis() {
  const crimeByArea: { [area: string]: number } = {};
  const crimeByAreaAndGender: {
    [area: string]: { male: number; female: number };
  } = {};

  crimeData.forEach((record) => {
    const area = record.area;
    const gender = record.victim_sex ? record.victim_sex.toUpperCase() : null;

    if (gender !== "M" && gender !== "F") {
      return;
    }

    if (!crimeByArea[area]) {
      crimeByArea[area] = 0;
    }
    crimeByArea[area]++;

    if (!crimeByAreaAndGender[area]) {
      crimeByAreaAndGender[area] = { male: 0, female: 0 };
    }

    if (gender === "M") {
      crimeByAreaAndGender[area].male++;
    } else if (gender === "F") {
      crimeByAreaAndGender[area].female++;
    }
  });

  const areaNames = Object.keys(crimeByArea);
  const crimeCountsGender = areaNames.map((area) => [
    crimeByAreaAndGender[area].male || 0,
    crimeByAreaAndGender[area].female || 0,
  ]);

  console.log("Crime counts for gender before clustering:", crimeCountsGender);

  const k = 3;
  const genderClusters = kMeans(crimeCountsGender, k);

  console.log("Generated Gender Clusters (Before Processing):", genderClusters);
  console.log("Gender-Based Safety Clusters:");
  genderClusters.forEach((cluster, index) => {
    console.log(‘Cluster ${index + 1}:‘);

    let totalMale = 0;
    let totalFemale = 0;
    const areasInCluster: string[] = [];

    cluster.forEach((point) => {
      const areaIndex = crimeCountsGender.findIndex(
        (crimeDataPoint) =>
          crimeDataPoint[0] === point[0] && crimeDataPoint[1] === point[1]
      );

      if (areaIndex >= 0 && areaIndex < crimeCountsGender.length) {
        const maleCount = crimeCountsGender[areaIndex][0] || 0;
        const femaleCount = crimeCountsGender[areaIndex][1] || 0;
        totalMale += maleCount;
        totalFemale += femaleCount;
        areasInCluster.push(areaNames[areaIndex]);
      } else {
        console.log(‘  Invalid area index: ${areaIndex}‘);
      }
    });

    const areaCount = areasInCluster.length;
    const averageMale = areaCount > 0 ? (totalMale / areaCount).toFixed(2) : 0;
    const averageFemale =
      areaCount > 0 ? (totalFemale / areaCount).toFixed(2) : 0;

    console.log(‘  Areas: ${areasInCluster.join(", ")}‘);
    console.log(‘  Total Male Crimes: ${totalMale}‘);
    console.log(‘  Total Female Crimes: ${totalFemale}‘);
    console.log(‘  Average Male Crimes per Area: ${averageMale}‘);
    console.log(‘  Average Female Crimes per Area: ${averageFemale}‘);
  });
}
```

Explanation: The `dataAnalysis()` function processes crime data to analyze crime distribution by area and gender and then applies clustering to group areas based on crime rates. It initializes two objects, **crimeByArea** for storing total crimes per area and **crimeByAreaAndGender** for storing gender-specific crime counts. It loops through each record in crimeData, extracting the area and gender, and skips records with invalid genders. For each valid record, it increments the crime count in both objects, ensuring each area is initialized as needed. After processing all records, it prepares the data for clustering by creating arrays of gender-specific crime counts for each area. The function then applies the k-means clustering algorithm to group areas based on these counts, generating clusters that are analyzed and printed, showing the total and average crimes per gender in each cluster, thus providing insights into gender-based safety in different areas.

3.3 K-means Clustering Implementation

K-means clustering is implemented in the following functions:

3.3.1 K-means Clustering Function

```
function kMeans(data: number[][], k: number): number[][][] {
  const centroids: number[][] = initializeCentroids(data, k);

  let clusters: number[][][] = [];
  for (let i = 0; i < k; i++) {
    clusters.push([]);
  }

  let iterations = 0;
  let prevCentroids: number[][] = [];

  while (!arraysEqual(prevCentroids, centroids) && iterations < 100) {
    clusters = assignClusters(data, centroids);
    prevCentroids = centroids.map((centroid) => centroid.slice());
    updateCentroids(clusters, centroids);
    iterations++;
  }

  return clusters;
}
```

Explanation: The `kMeans()` function encapsulates the K-means clustering logic. It initializes centroids and clusters, iterates to assign data points to the nearest centroid, and updates the centroids based on the current cluster memberships. The process continues until the centroids stabilize or a maximum of 100 iterations is reached, ensuring efficient convergence.

3.3.2 Initialize Centroids

```
function initializeCentroids(data: number[][], k: number): number[][] {
  const centroids: number[][] = [];
  const usedIndices: Set<number> = new Set();
  while (centroids.length < k) {
    const index = Math.floor(Math.random() * data.length);
    if (!usedIndices.has(index)) {
      centroids.push(data[index]);
      usedIndices.add(index);
    }
  }
  return centroids;
}
```

Explanation: This function randomly selects K unique data points as initial centroids for the clustering process. The use of a set ensures that each centroid is unique, which is critical for the algorithm's performance.

3.3.3 Assign Clusters

```
function assignClusters(data: number[][], centroids: number[][]): number[][][] {
  const clusters: number[][][] = [];
  for (let i = 0; i < centroids.length; i++) {
    const currentCluster: number[][] = [];
    clusters.push(currentCluster);
  }

  data.forEach((point, index) => {
    let minDist = Infinity;
    let closestCentroid = 0;

    centroids.forEach((centroid, centroidIndex) => {
      const distance = euclideanDistance(point, centroid);
      if (distance < minDist) {
        minDist = distance;
        closestCentroid = centroidIndex;
      }
    });

    const pointWithIndex = new Array(point.length + 1);
    for (let i = 0; i < point.length; i++) {
      pointWithIndex[i] = point[i];
    }
    pointWithIndex[point.length] = index;

    clusters[closestCentroid].push(pointWithIndex);
  });

  return clusters;
}
```

Explanation: The assignClusters() function categorizes each data point into the closest cluster based on the calculated Euclidean distance to the centroids. Each cluster is represented as an array of points, and the function ensures that points are added to the correct cluster.

3.3.4 Update Centroids

```
function updateCentroids(clusters: number[][][], centroids: number[][]): void {
  clusters.forEach((cluster, index) => {
    if (cluster.length === 0) return;

    const newCentroid: number[] = [];
    for (let i = 0; i < cluster[0].length - 1; i++) {
      newCentroid[i] = 0;
    }

    cluster.forEach((point) => {
      point.slice(0, -1).forEach((value, i) => {
        newCentroid[i] += value;
      });
    });

    newCentroid.forEach((sum, i) => {
      newCentroid[i] = sum / cluster.length;
    });

    centroids[index] = newCentroid;
  });
}
```

Explanation: This function recalculates the centroids of each cluster by averaging the points assigned to it. If a cluster is empty, it skips the update for that cluster. The new centroid for each cluster is computed and stored for the next iteration.

3.3.5 Euclidean Distance

```
function euclideanDistance(point1: number[], point2: number[]): number {
  let sumOfSquares = 0;
  for (let i = 0; i < point1.length; i++) {
    const difference = point1[i] - point2[i];
    const multOfDifference = difference * difference;
    sumOfSquares += multOfDifference;
  }
  return Math.sqrt(sumOfSquares);
}
```

Explanation: The `euclideanDistance()` function calculates the Euclidean distance between two points . It takes two arrays of numbers, point1 and point2, as input and initializes a variable sumOfSquares to 0. It then iterates through each dimension, calculating the squared difference between corresponding elements of the two arrays and adding these squared differences to sumOfSquares. Finally, it returns the square root of sumOfSquares, which is the Euclidean distance between the two points.

3.3.6 Arrays Equal Check

```
function arraysEqual(arr1: number[][], arr2: number[][]): boolean {
  if (arr1.length !== arr2.length) return false;

  for (let i = 0; i < arr1.length; i++) {
    if (arr1[i].length !== arr2[i].length) return false;
    for (let j = 0; j < arr1[i].length; j++) {
      if (arr1[i][j] !== arr2[i][j]) return false;
    }
  }

  return true;
}
```

Explanation: The `arraysEqual()` function checks whether two 2D arrays are the same, which is important for the k-means clustering process. It compares arrays that represent clusters of crime data points to see if they match in size and content. If the arrays have different lengths or any values that don't line up, it returns false. If everything matches, it returns true. This check helps the k-means algorithm determine when the centroids have stabilized, allowing it to stop iterating when the clusters are no longer changing.

4 Results and Discussion

The analysis began with the execution of the clustering algorithm, which first processed the crime data by compiling counts of male and female victims across various areas. This preliminary analysis provided individual counts, revealing the distribution of crime based on gender.

Following this, the data was grouped into three clusters based on gender-related safety. Each cluster's analysis includes the total and average number of crimes for both male and female victims. These findings have been logged for verification and to provide valuable insights into crime patterns across different areas.

The results indicate that the CSV file was successfully processed, with the following counts for male and female victims before clustering:

```
    CSV file successfully processed.
Crime counts for gender before clustering: [
  [ 28992, 18552 ], [ 15609, 14237 ],
  [ 16319, 19721 ], [ 12115, 11351 ],
  [ 12044, 13168 ], [ 19206, 15282 ],
  [ 17173, 14219 ], [ 16862, 13501 ],
  [ 15713, 13726 ], [ 18400, 13022 ],
  [ 14124, 12696 ], [ 17555, 23714 ],
  [ 17660, 15454 ], [ 20240, 16246 ],
  [ 21845, 14617 ], [ 12212, 10861 ],
  [ 13056, 12199 ], [ 13392, 19573 ],
  [ 14062, 13240 ], [ 16369, 15338 ],
  [ 18414, 12751 ]
]
```

The clustering analysis resulted in three distinct gender-based safety clusters:

```
Gender-Based Safety Clusters:
Cluster 1:
  Areas: 4, 5, 11, 16, 17, 19
  Total Male Crimes: 77613
  Total Female Crimes: 73515
  Average Male Crimes per Area: 12935.50
  Average Female Crimes per Area: 12252.50

Cluster 2:
  Areas: 2, 6, 7, 8, 9, 10, 13, 14, 15, 20, 21
  Total Male Crimes: 197491
  Total Female Crimes: 158393
  Average Male Crimes per Area: 17953.73
  Average Female Crimes per Area: 14399.36

Cluster 3:
  Areas: 1, 3, 12, 18
  Total Male Crimes: 76258
  Total Female Crimes: 81560
  Average Male Crimes per Area: 19064.50
  Average Female Crimes per Area: 20390.00
```

These findings highlight the varying levels of gender-based crime across different areas. **Cluster 1** shows a balanced but significant level of crime for both males and females, indicating a need for intervention. **Cluster 2** reveals areas with the highest crime rates for both genders, suggesting urgent safety measures are required. In textbfCluster 3, the higher incidence of female crimes compared to male crimes points to a concerning gender disparity, which necessitates targeted approaches to enhance safety for female victims. Overall, these results provide critical insights for developing effective crime prevention strategies tailored to the unique characteristics of each cluster.

4.1 Variability in Results

An important point to note is that the results of the K-means clustering vary with each run due to the random initialization of centroids in the `initializeCentroids` function. Since `Math.random()` is used to select the initial centroids, this can lead to different cluster formations each time the algorithm is executed, resulting in inconsistent results.

```
const index = Math.floor(Math.random() * data.length);
```

4.2 Mistakes in Implementation and Incorrect Results

The incorrect results were mainly due to invalid area indices that occurred when assigning data points to clusters. This mismatch caused errors in the output. To fix this, the logic in the `assignClusters` function was updated to ensure that each data point was properly linked to its corresponding index. Stricter checks were added as well in the `dataAnalysis` function to filter out any invalid entries.

```
    CSV file successfully processed.
Gender-Based Safety Clusters:
Cluster 1:
Invalid area index: 28992
Cluster 2:
Invalid area index: 15609
Invalid area index: 12115
Invalid area index: 12044
Invalid area index: 15713
Invalid area index: 14124
Invalid area index: 12212
Invalid area index: 13056
Invalid area index: 14062
Cluster 3:
Invalid area index: 16319
Invalid area index: 19206
Invalid area index: 17173
Invalid area index: 16862
Invalid area index: 18400
Invalid area index: 17555
Invalid area index: 17660
Invalid area index: 20240
Invalid area index: 21845
Invalid area index: 13392
Invalid area index: 16369
```

5 Conclusion

This project demonstrated the application of K-means clustering to crime data to identify and analyze safety patterns in Los Angeles. By focusing on overall and gender-specific crime distributions, the analysis provides valuable insights into the relative safety of different areas.

The comprehensive breakdown of the TypeScript implementation illustrates the step-by-step approach to reading, preprocessing, and clustering the data, offering a detailed understanding of the methodology and results.

6 Compilation and Execution

To compile and run the project, please follow the steps outlined below:

6.1 Prerequisites

Ensure that you have the following software installed on your machine:

- **Node.js** - A JavaScript runtime for executing TypeScript.
- **npm** (Node Package Manager) - Comes with Node.js, used for managing project dependencies.
- **TypeScript** - Install globally via npm using the command:

  ```
  npm install -g typescript
  ```

6.2 Project Setup

1. Clone the repository from GitHub:

```
git clone https://github.com/EmaCikotic/SafeHarborInsights.git
```

2. Navigate to the project directory:

```
cd SafeHarborInsights
```

3. Install the required dependencies:

```
npm install
```

6.3 Compiling the TypeScript Code

To compile the TypeScript code, run the following command:

```
npx tsc
```

6.4 Running the Project

After successful compilation, run the project using the following command:

```
node out/index.js
```

Make sure to replace `out/index.js` with the correct path to your main JavaScript file if it differs.

6.5 Output

Upon running the project, the console will display the processed results and the generated clusters. Ensure that the CSV dataset is present in the specified path in the code for successful execution.

7 References

- GeeksforGeeks: K-means Clustering Introduction
- Kaggle: LA Crime Rates from 2020 to 2023
- K-means Clustering Algorithm in Machine Learning
- Importing Dataset using TypeScript
- Java K-means Clustering Algorithm
- Git-filter
- Initialization of centroids
- Github Repository