INTRODUCTION TO BIG DATA

PROJECT MILESTONE 2
CHICAGO CRIME DATA ANALYSIS

BY - VISHESH RAJU - vr432
SCHOOL ID - 31695318

Contents

INTRODUCTION

Crime rates and trends have long been a subject of concern for city planners and law enforcement agencies, impacting public safety and urban planning. In a bustling city like Chicago, understanding crime patterns is crucial for allocating resources effectively and devising preventive measures. The Chicago Crime Dataset offers a wealth of information, with detailed records of crimes including their types, locations, times, and arrest statuses.

With the power of big data technologies, analyzing this dataset can uncover meaningful insights that are otherwise hidden in the sheer volume of data. By leveraging tools like Hadoop and Oozie, this project aims to systematically process and analyze the dataset to address critical questions such as:

1. What are the most common types of crimes in Chicago?
2. Which areas are hotspots for criminal activities?
3. How do crime rates vary by time, such as by year, month, or hour?

Through this analysis, the project not only highlights crime patterns but also demonstrates the scalability and efficiency of big data frameworks in processing and extracting actionable insights from large datasets. Ultimately, the findings can guide city officials in making data-driven decisions to enhance public safety and resource allocation.

SOFTWARE SPECIFICATIONS

In this section we will see the Software that will be used in the creation of this project.

1. Windows 11 Home OS

2. Amazon EC2

3. Hadoop

4. Oozie

5. Java

6. Python

DATASET

The **Chicago Crime Dataset** is a publicly available dataset maintained by the City of Chicago. It contains detailed records of reported crimes, spanning several years, and is updated regularly it contains data from the year 2001 to Present minus the seven most recent days. The dataset is approximately **1.7 GB** in size and comprises over **8 million records**. The link to the dataset https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2/data

Features:

1. **ID**: Unique identifier for each crime record.
2. **Case Number**: Unique case number assigned by the police department.
3. **Date**: Date and time when the crime occurred.
4. **Block**: Street address or block location where the crime occurred.

5. **IUCR**: Illinois Uniform Crime Reporting code that categorizes the crime.
6. **Primary Type**: General classification of the crime (e.g., Theft, Robbery).
7. **Description**: Specific details about the crime within the primary type.
8. **Location Description**: Type of location where the crime took place (e.g., Alley, Residence).
9. **Arrest**: Indicates if an arrest was made (True/False).
10. **Domestic**: Indicates if the crime was domestic in nature (True/False).
11. **Beat**: Smallest police geographic area for patrol and crime tracking.
12. **District**: Larger police geographic division encompassing multiple beats.
13. **Ward**: Political ward of Chicago where the crime occurred.
14. **Community Area**: Numeric code representing one of Chicago's 77 community areas.
15. **FBI Code**: FBI classification code for categorizing crimes.
16. **X Coordinate**: X-coordinate for the crime's location in the Chicago spatial coordinate system.
17. **Y Coordinate**: Y-coordinate for the crime's location in the Chicago spatial coordinate system.
18. **Year**: Year when the crime occurred.
19. **Updated On**: Date when the record was last updated.
20. **Latitude**: Geographical latitude of the crime location.
21. **Longitude**: Geographical longitude of the crime location.
22. **Location**: Combined representation of latitude and longitude.

This dataset offers rich information for both spatial and temporal analysis, as well as for understanding crime patterns and trends across different locations and time frames.

ALGORITHMS

A. First Map-Reduce: Crime Frequency by Type

1. The **Mapper** starts first.
2. The Mapper reads the dataset line by line but skips rows with missing or invalid values in the Primary Type column. For each valid record, it emits the Primary Type as the key and the value 1 as output.
3. Now, the **Reducer** starts.
4. The Reducer groups data by Primary Type and sums up all the 1s received from the Mapper to calculate the total count of each crime type.
5. The Reducer then sorts the crime types by their frequencies in descending order.
6. If the dataset is empty or all values are invalid, the output will be a statement indicating the operation cannot be performed.

B. Second Map-Reduce: High-Crime Areas

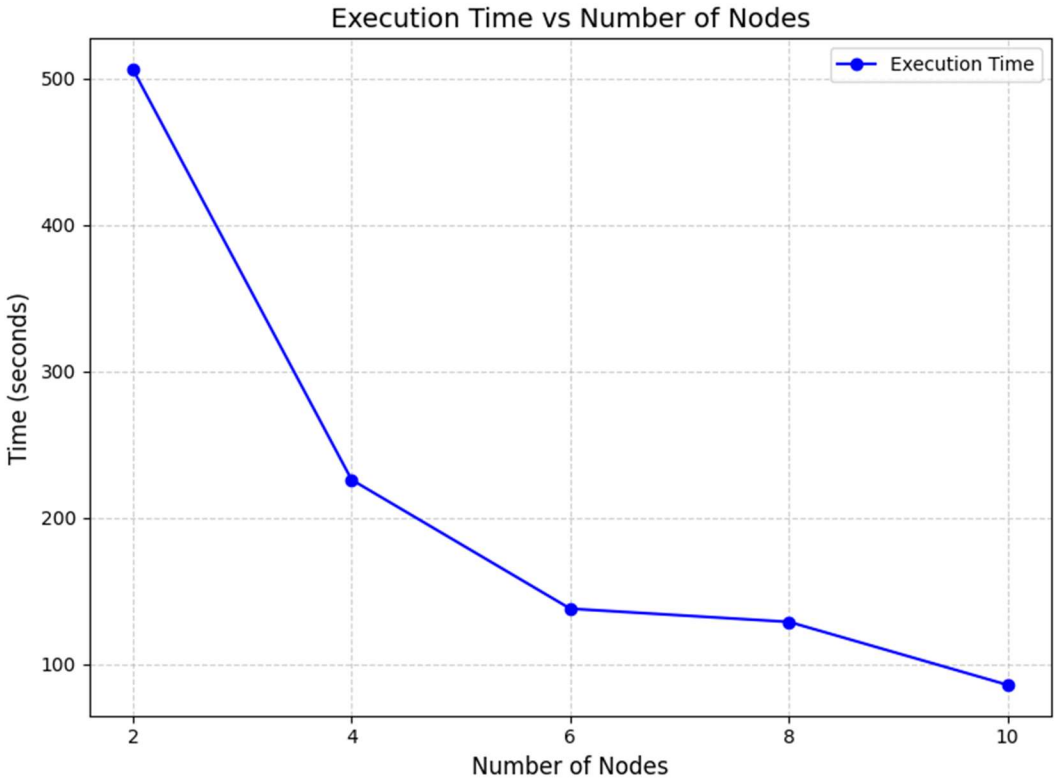
1. The **Mapper** starts first.
2. The Mapper reads the dataset line by line and skips rows with missing or invalid values in the Community Area or Location columns. For each valid record, it emits the Community Area (or another location identifier) as the key and the value 1 as output.
3. Now, the **Reducer** starts.
4. The Reducer groups data by Community Area and sums up all the 1s received from the Mapper to calculate the total count of crimes in each area.
5. The Reducer then sorts the areas by crime count in descending order and outputs the top 10 high-crime areas.
6. If the location data is invalid or missing, the output will state that the operation cannot be completed.

C. Third Map-Reduce: Temporal Crime Trends

1. The **Mapper** starts first.
2. The Mapper reads the dataset line by line and skips rows with missing or invalid date and time fields. For each valid record, it extracts the required time unit (e.g., year, month, or hour) and emits it as the key, with the value 1 as output.
3. Now, the **Reducer** starts.
4. The Reducer groups data by the time unit and sums up all the 1s received from the Mapper to calculate the total number of crimes for each unit of time.
5. The Reducer then organizes the data in chronological order and outputs the crime frequency for each time unit.
6. If the time data is invalid or missing, the output will state that the operation cannot be performed.

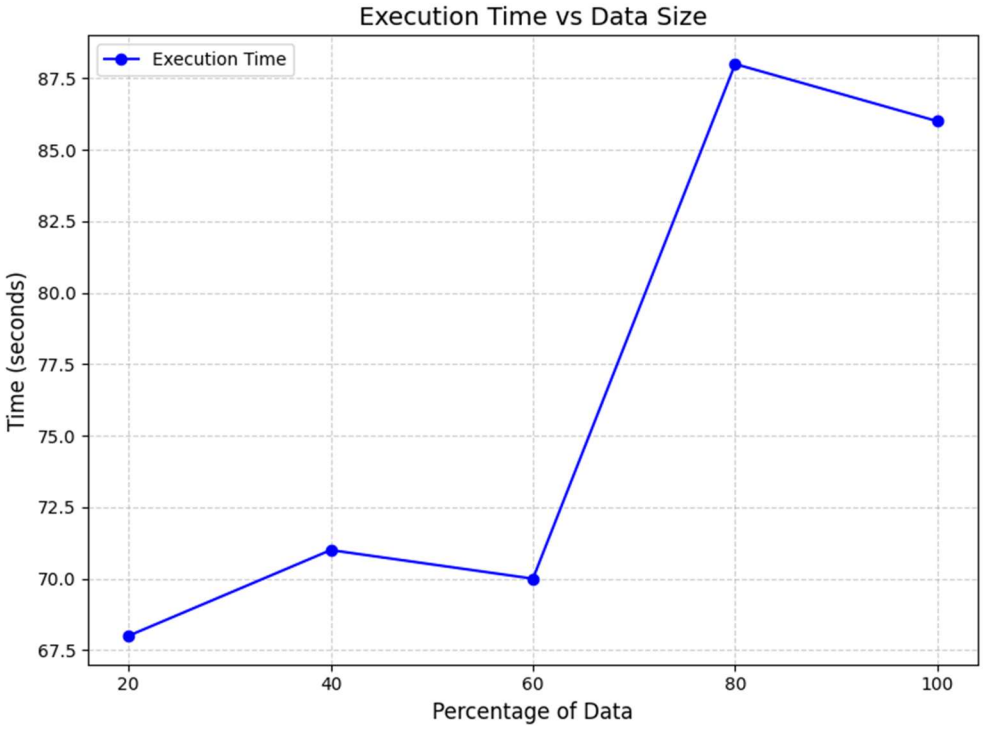
PERFORMANCE MEASUREMENT PLOTS

1. Processing Time vs Number of VMs



The graph illustrates the relationship between the number of nodes and execution time for a computational workload. As the number of nodes increases, the execution time decreases significantly, demonstrating improved parallel processing efficiency. For instance, with 2 nodes, the execution time is 506 seconds, which reduces to 86 seconds with 10 nodes. This trend highlights the scalability of the distributed system, where additional nodes effectively share the workload, leading to faster job completion. However, the rate of improvement diminishes at higher node counts, indicating possible overhead or diminishing returns as the system scales.

2. Processing Time vs Data Percentage (10 VMs)



This graph illustrates the relationship between the percentage of data processed and the execution time. As the data size increases, the execution time shows a gradual rise to 60% of the data, after which there is a sharp increase in execution time. For example, processing 20% of the data takes 68 seconds, while processing 80% takes 88 seconds. This trend highlights the non-linear relationship between data size and execution time, potentially due to system resource limitations or processing inefficiencies at higher data volumes. The graph underscores the importance of efficient data handling as dataset sizes grow.

.

PERFORMANCE OPTIMIZATION

To optimize the performance of the Hadoop MapReduce workflow, several adjustments were made to both system configurations and job execution parameters:

1. **Resource Allocation:** The YARN and MapReduce memory parameters were fine-tuned. Specifically, the mapreduce.map.memory.mb and mapreduce.reduce.memory.mb values were adjusted to prevent resource underutilization or excessive task terminations. Similarly, yarn.scheduler.minimum-allocation-mb and yarn.scheduler.maximum-allocation-mb were configured to align with the total available memory of the EC2 instances.

2. **Data Partitioning:** The input data was carefully partitioned using custom partitioners to ensure an even distribution of data across mappers and reducers. This prevented some tasks from becoming bottlenecks due to uneven load distribution.
3. **Task Parallelism:** The number of mappers and reducers was optimized based on the size of the dataset and cluster capacity. By configuring mapreduce.job.reduces and ensuring sufficient mapper splits, the system achieved better parallelism and reduced execution time.
4. **Intermediate Data Management:** Parameters such as mapreduce.task.io.sort.mb and mapreduce.reduce.shuffle.parallelcopies were optimized to improve the efficiency of intermediate data transfer and sorting processes, which reduced disk I/O overheads.
5. **Network and Disk Usage:** To minimize network overhead, data locality was prioritized by enabling HDFS rack-awareness. Additionally, temporary files were stored on SSDs to speed up disk I/O during shuffle and sort operations.
6. **Monitoring and Logs:** Continuous monitoring using the Hadoop Resource Manager and analyzing execution logs helped identify bottlenecks like memory starvation and node failures. Iterative adjustments based on these insights led to improved resource utilization and system throughput.

Through these optimizations, significant improvements in processing time and system efficiency were achieved, enabling the cluster to handle larger data sizes effectively with reduced runtime.

ERROR HANDLING AND TROUBLESHOOTING

During the workflow execution, several challenges were encountered, particularly while running the MapReduce jobs. One notable issue was misconfiguration of YARN and MapReduce parameters, which resulted in errors such as *"Java Heap Space Error"* and *"Container killed by YARN for exceeding memory limits."* These errors stemmed from incorrect memory allocations in the mapreduce.map.memory.mb and mapreduce.reduce.memory.mb settings, leading to tasks being terminated prematurely.

To resolve these issues, I revisited the YARN configuration and adjusted parameters such as yarn.nodemanager.resource.memory-mb, yarn.scheduler.maximum-allocation-mb, and mapreduce.task.io.sort.mb to ensure better alignment with the available system resources. Additionally, by analyzing the Hadoop logs (yarn logs and task-specific logs), I identified and fixed input data partitioning issues that were causing load imbalances. These adjustments stabilized the MapReduce workflow and improved the efficiency of task execution.

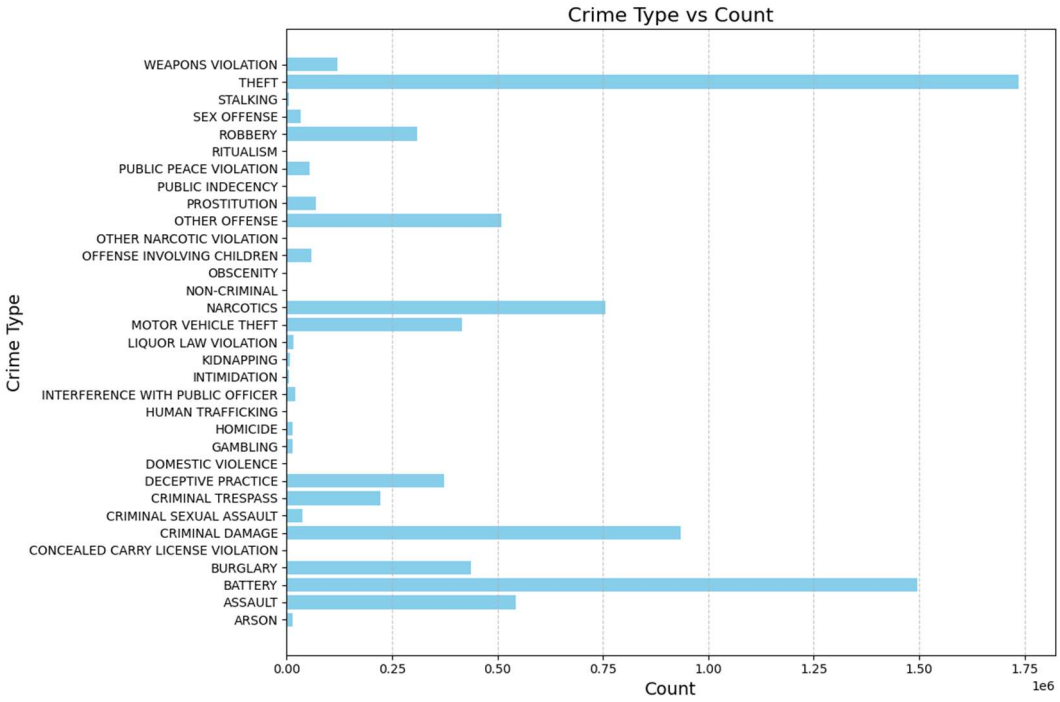
RESULTS AND VISUALIZATION

1. **Crime Type Analysis (Job 1)**:

- The analysis of the crime type reveals the most frequent crime types:

- **Theft** is the most reported crime with **1,736,261 cases**.
- **Battery** follows with **1,495,010 cases**.
- Other high-frequency crime types include **Criminal Damage** (933,672 cases), **Narcotics** (756,709 cases), and **Assault** (544,229 cases).

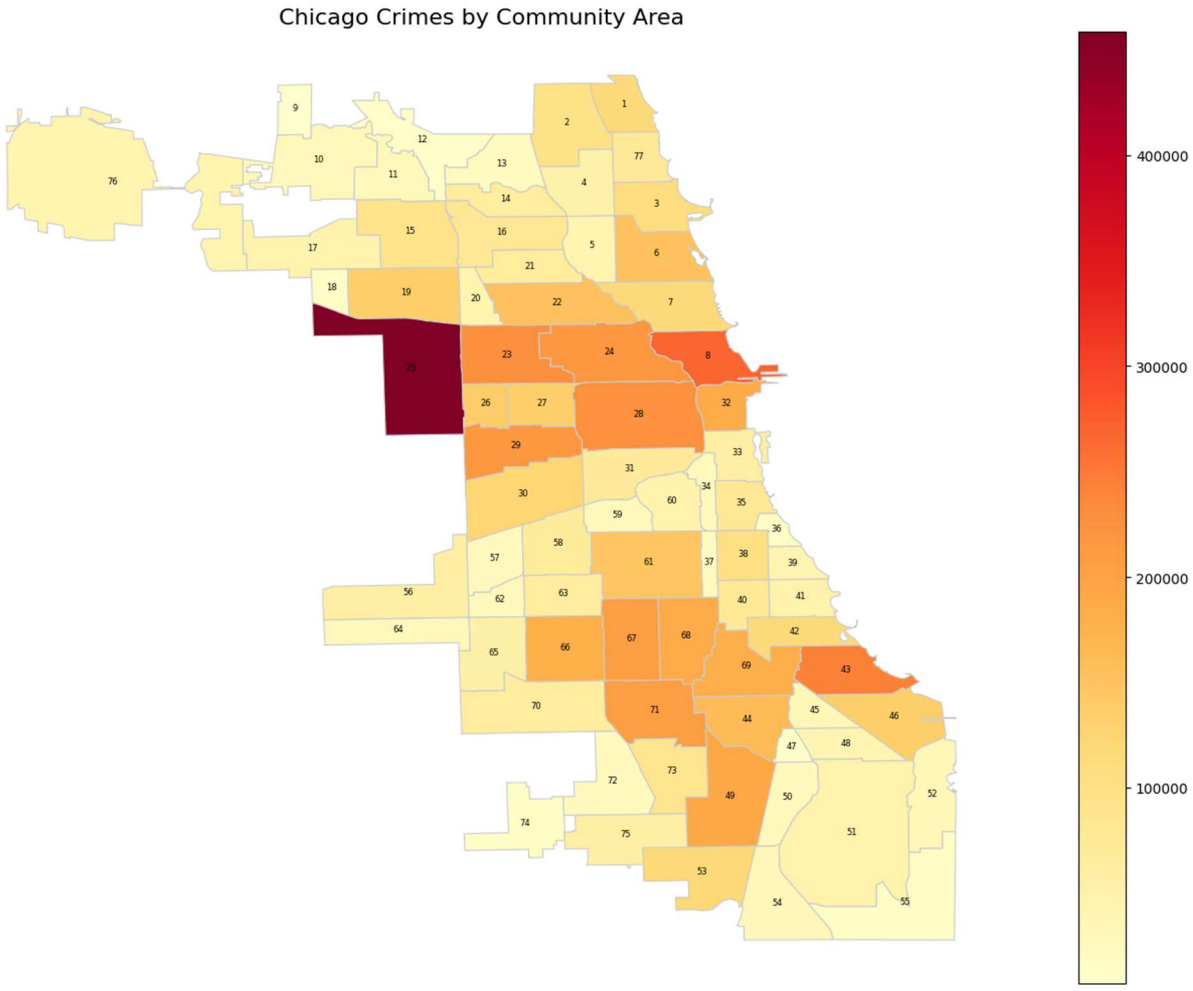
- Less frequent crimes include **Human Trafficking** (106 cases), **Obscenity** (897 cases), and **Domestic Violence** (1 reported case).



2. Crime Area Analysis (Job 2):

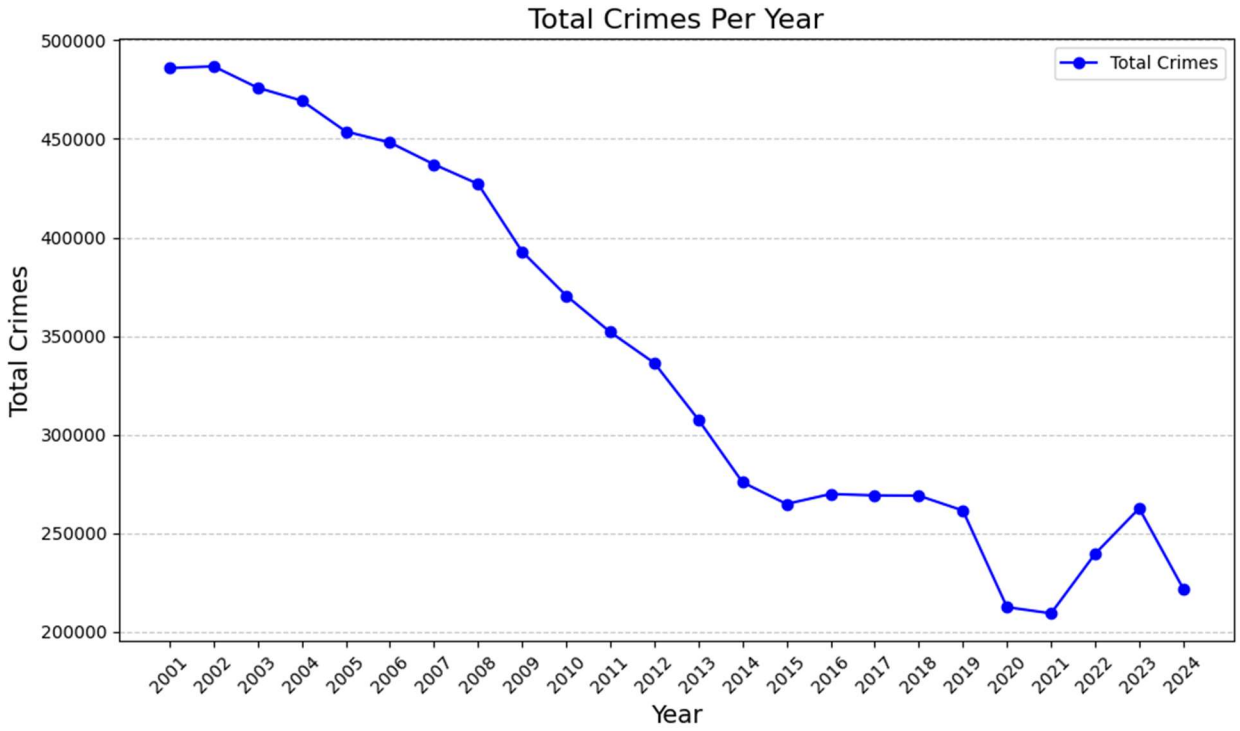
- The analysis of the crime type reveals the most frequent crime types:
- **Community Area 25** recorded the highest number of crimes, with **458,385 incidents**, making it the most crime-prone area in the dataset.
- **Community Area 9** reported the lowest number of crimes, with only **7,404 incidents**, indicating significantly lower crime activity in this region.
- The **average number of crimes per community area** is approximately **107,808**.
- **Median Crimes:** The middle value is **54,340** (Community Area 65), indicating that half the community areas experience less than this number of crimes, while the other half experience more.
- Areas with significantly high crime rates include:
  - **Area 23:** 227,888 crimes.
  - **Area 24:** 215,738 crimes.
  - **Area 28:** 226,065 crimes.
- Areas with relatively few crimes include:
  - **Area 47:** 10,879 crimes.

- **Area 12:** 13,972 crimes.
- **Area 74:** 15,686 crimes.



3. **Temporal Trend Analysis (Job 3)**:

- The data outlines monthly trends from **2001 to 2023**.
- There is a noticeable decline in the overall values from earlier years (2001–2010) to later years (2020–2023).
- Peak months typically occur in mid-year, such as **July and August**, while a significant drop is observed during **December**.
- Some notable values:
- **July 2001** shows one of the highest counts: **44,704**.
- By contrast, **December 2023** records a sharp decline at **21,263**.


Total Crimes Per Year
Total Crimes
500000
450000
400000
350000
300000
250000
200000
Total Crimes
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
Year