Analyzing and Visualizing Chicago Crime Data Using Big Data Tools

---

Group Information

- **Group Number: 1**
- **Group Members:**
  - Omar Ackef - 202101592
  - Abdelrahman Ammar - 202102787

Project Overview

This project focuses on analyzing a sample of crime data from Chicago to uncover patterns, trends, and insights using Big Data tools. It integrates geospatial and temporal data analysis techniques to identify crime distributions by location and type, offering a scalable approach to processing and visualizing large datasets.

Tools and Technologies

The project leverages the following tools and technologies:

- **Apache Spark:** A distributed processing framework used for data ingestion, transformation, and querying.
- **Apache Sedona:** A geospatial processing library integrated with Spark for handling spatial operations, such as spatial joins and geometry manipulations.
- **GeoPandas:** A Python library for geospatial data visualization, used for generating choropleth maps.
- **Matplotlib:** A Python library for creating bar charts and other visualizations.
- **QGIS:** A desktop GIS tool for advanced spatial data visualization.

Goals and Objectives

1. **Data Cleaning and Enrichment:**
   - Process raw crime data from Chicago.
   - Add geospatial attributes (e.g., ZIP codes) for each crime record.
2. **Spatial Analysis:**
   - Aggregate crime data by location.
   - Visualize crime distribution across Chicago's ZIP codes using a choropleth map.
3. **Temporal Analysis:**
   - Analyze crime trends over a specified time range.
   - Generate bar charts to illustrate the number of crimes by type.
4. **Scalability:**
   - Demonstrate the use of Big Data tools for handling large datasets.
   - Optimize data processing and visualization for performance.

Dataset Description

The project utilizes a crime dataset provided by the Chicago Police Department. The dataset includes the following key attributes:

- **Case Number:** Unique identifier for each crime.
- **Primary Type:** The type/category of the crime (e.g., theft, assault).
- **Date:** The date and time the crime occurred.
- **Location Description:** The type of location where the crime took place.
- **X, Y Coordinates:** The geographic location of the crime in a coordinate system.
- **Community Area:** The community area where the crime occurred.
- **FBI Code:** The classification code assigned by the FBI.

Methodology Overview

The methodology for this project is divided into three major tasks:

1. **Task 1: Data Cleaning and Enrichment**
    - Loading the dataset, cleaning it, and enriching it with geospatial attributes using spatial joins.
    - Saving the processed data in Parquet format for further analysis.
2. **Task 2: Spatial Analysis**
    - Aggregating crime data by ZIP code using Spark SQL.
    - Exporting the results as a GeoJSON file and visualizing crime distributions through a choropleth map.
3. **Task 3: Temporal Analysis**
    - Filtering crimes based on a user-specified date range.
    - Analyzing crime trends by type and generating visualizations using bar charts.

Task 1 - Data Preprocessing

Task Objective

The goal of Task 1 was to process raw crime data, clean and enrich it by adding geospatial attributes, and save the resulting dataset in a scalable format for further analysis.

Methodology

Step 1: Load Raw Data

- The crime data was loaded from a **CSV file** into a Spark DataFrame.
- This data contained key attributes such as Case Number, Primary Type, Date, Location Description, and coordinates (X, Y).

Step 2: Clean the Data

- Renamed columns containing spaces to improve compatibility with Big Data tools and SQL queries. For example:
    - "Case Number" → CaseNumber
    - "Primary Type" → PrimaryType
    - "Location Description" → LocationDescription

Step 3: Load Geospatial Data

- A **shapefile** containing ZIP code geometries for Chicago was loaded using **Apache Sedona**.
- The shapefile was registered as a Spark DataFrame for spatial operations.

Step 4: Perform Spatial Join

- A spatial join was performed using Sedona's ST_Contains function to associate each crime record with the ZIP code it occurred in.
    - **Operation**: ST_Contains(ZIPCode.geometry, ST_Point(X, Y))
    - This enriched each crime record with a ZIPCode attribute.

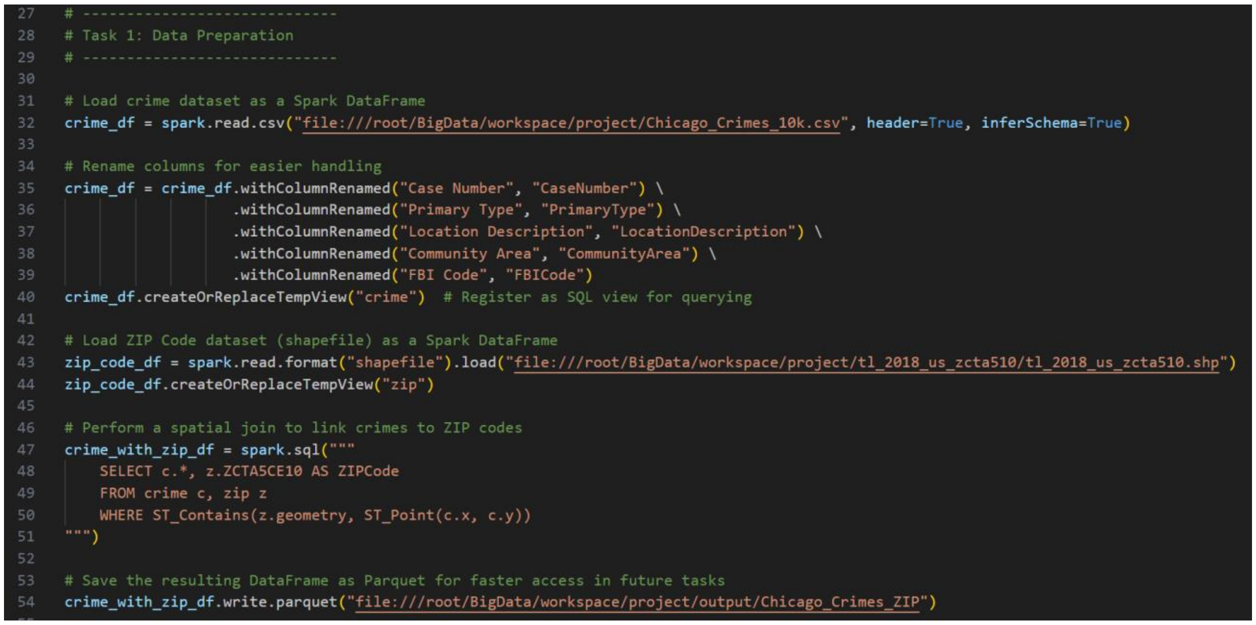
Step 5: Save the Processed Data

- The enriched dataset, including crime data and ZIP codes, was saved in **Parquet format**.
- **Reason for Parquet**:
    - Optimized for query performance.
    - Efficient storage for large datasets.

Tools and Libraries Used

- **Apache Spark**: For data processing and SQL-based queries.
- **Apache Sedona**: For performing spatial joins and handling geospatial data.
- **Parquet Format**: For efficient storage and scalability.

Code Snippet


```
# ----------------------------
# Task 1: Data Preparation
# ----------------------------

# Load crime dataset as a Spark DataFrame
crime_df = spark.read.csv("file:///root/BigData/workspace/project/Chicago_Crimes_10k.csv", header=True, inferSchema=True)

# Rename columns for easier handling
crime_df = crime_df.withColumnRenamed("Case Number", "CaseNumber") \
                   .withColumnRenamed("Primary Type", "PrimaryType") \
                   .withColumnRenamed("Location Description", "LocationDescription") \
                   .withColumnRenamed("Community Area", "CommunityArea") \
                   .withColumnRenamed("FBI Code", "FBICode")
crime_df.createOrReplaceTempView("crime")  # Register as SQL view for querying

# Load ZIP Code dataset (shapefile) as a Spark DataFrame
zip_code_df = spark.read.format("shapefile").load("file:///root/BigData/workspace/project/tl_2018_us_zcta510/tl_2018_us_zcta510.shp")
zip_code_df.createOrReplaceTempView("zip")

# Perform a spatial join to link crimes to ZIP codes
crime_with_zip_df = spark.sql("""
    SELECT c.*, z.ZCTA5CE10 AS ZIPCode
    FROM crime c, zip z
    WHERE ST_Contains(z.geometry, ST_Point(c.x, c.y))
""")

# Save the resulting DataFrame as Parquet for faster access in future tasks
crime_with_zip_df.write.parquet("file:///root/BigData/workspace/project/output/Chicago_Crimes_ZIP")
```

6. Outcome

- The processed data included the following attributes:
    - Original crime attributes: CaseNumber, PrimaryType, Date, LocationDescription, etc.
    - Geospatial attribute: ZIPCode
- The dataset was saved in **Parquet format** for subsequent tasks.

7. Why Parquet Format is Helpful for This Project

Parquet is a great fit for this project because it stores data efficiently and speeds up queries. Its column-based format compresses data better than traditional formats like CSV, saving storage space. When running queries, Parquet only reads the necessary columns, making it faster for big datasets like Chicago crime data. It works seamlessly with tools like Apache Spark, which is perfect for our analysis. Overall, Parquet helps us handle large data easily and keeps the process efficient.

Dataset	.csv size	.parquet size
1k	200 KB	80 KB
10k	1998 KB	640 KB
100k	19985 KB	661 KB

Task 2 - Spatial Analysis

Task Objective

The goal of Task 2 was to analyze the spatial distribution of crimes across Chicago by aggregating crime counts by ZIP code and visualizing the results using a choropleth map. The primary task was to create this map using **QGIS**, and we added an **automated choropleth map** using **GeoPandas and Matplotlib** as an enhancement.

Methodology

Step 1: Load Enriched Data

- The enriched dataset saved in **Parquet format** during Task 1 was loaded into Spark for spatial analysis.

Step 2: Aggregate Crime Data by ZIP Code

- Crime records were grouped by the ZIPCode attribute, and the total number of crimes in each ZIP code was calculated using Spark SQL.
- The result included two columns:
    - ZIPCode: The ZIP code of the area.
    - CrimeCount: The total number of crimes in that ZIP code.

Step 3: Join with Geospatial Data

- The aggregated crime data was joined with the ZIP code geometries from the shapefile using the ZIPCode attribute.
- This step added the geometric boundaries of each ZIP code to the aggregated crime data, enabling spatial visualization.

Step 4: Save Data in GeoJSON Format

- The resulting dataset, containing CrimeCount and ZIP code geometries, was saved as a **GeoJSON file**. This format is compatible with tools like QGIS for advanced spatial visualization.

Step 5: Use QGIS to Generate the Choropleth Map

- The GeoJSON file was imported into **QGIS** for an additional choropleth map visualization:
    1. The GeoJSON file was loaded as a vector layer in QGIS.

2. A **graduated symbology** was applied to the CrimeCount column:

3. Labels, legends, and map elements were added for clarity.

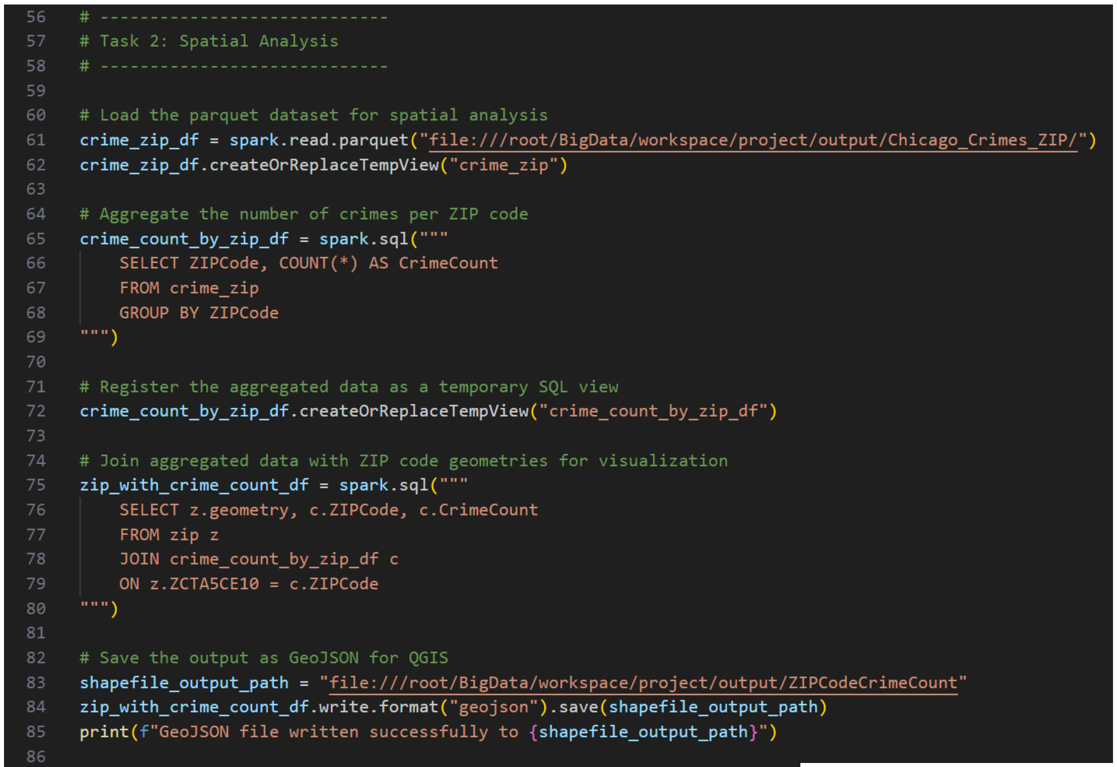
Step 6: Automate Choropleth Map Creation (Optional)

- In addition to the QGIS map, we automated the creation of a choropleth map using **GeoPandas** and **Matplotlib**:
- This approach enabled programmatic generation of the map for quick visualization and validation.

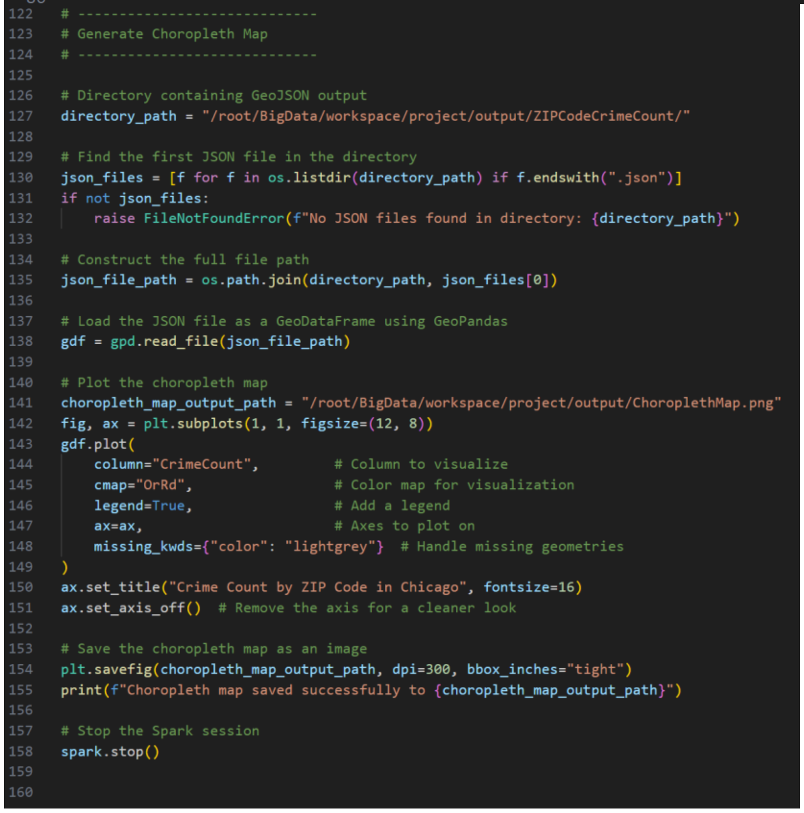
Tools and Libraries Used

- **Apache Spark**: For data aggregation and SQL-based analysis.
- **Apache Sedona**: For handling geospatial data and performing spatial joins.
- **GeoPandas**: For geospatial visualization using Python.
- **QGIS**: For creating the primary choropleth map.
- **GeoPandas/MatPlotLib**: For automating choropleth map creation as an additional feature.

Code Snippet



```
# ----------------------------
# Task 2: Spatial Analysis
# ----------------------------

# Load the parquet dataset for spatial analysis
crime_zip_df = spark.read.parquet("file:///root/BigData/workspace/project/output/Chicago_Crimes_ZIP/")
crime_zip_df.createOrReplaceTempView("crime_zip")

# Aggregate the number of crimes per ZIP code
crime_count_by_zip_df = spark.sql("""
    SELECT ZIPCode, COUNT(*) AS CrimeCount
    FROM crime_zip
    GROUP BY ZIPCode
""")

# Register the aggregated data as a temporary SQL view
crime_count_by_zip_df.createOrReplaceTempView("crime_count_by_zip_df")

# Join aggregated data with ZIP code geometries for visualization
zip_with_crime_count_df = spark.sql("""
    SELECT z.geometry, c.ZIPCode, c.CrimeCount
    FROM zip z
    JOIN crime_count_by_zip_df c
    ON z.ZCTA5CE10 = c.ZIPCode
""")

# Save the output as GeoJSON for QGIS
shapefile_output_path = "file:///root/BigData/workspace/project/output/ZIPCodeCrimeCount"
zip_with_crime_count_df.write.format("geojson").save(shapefile_output_path)
print(f"GeoJSON file written successfully to {shapefile_output_path}")

```

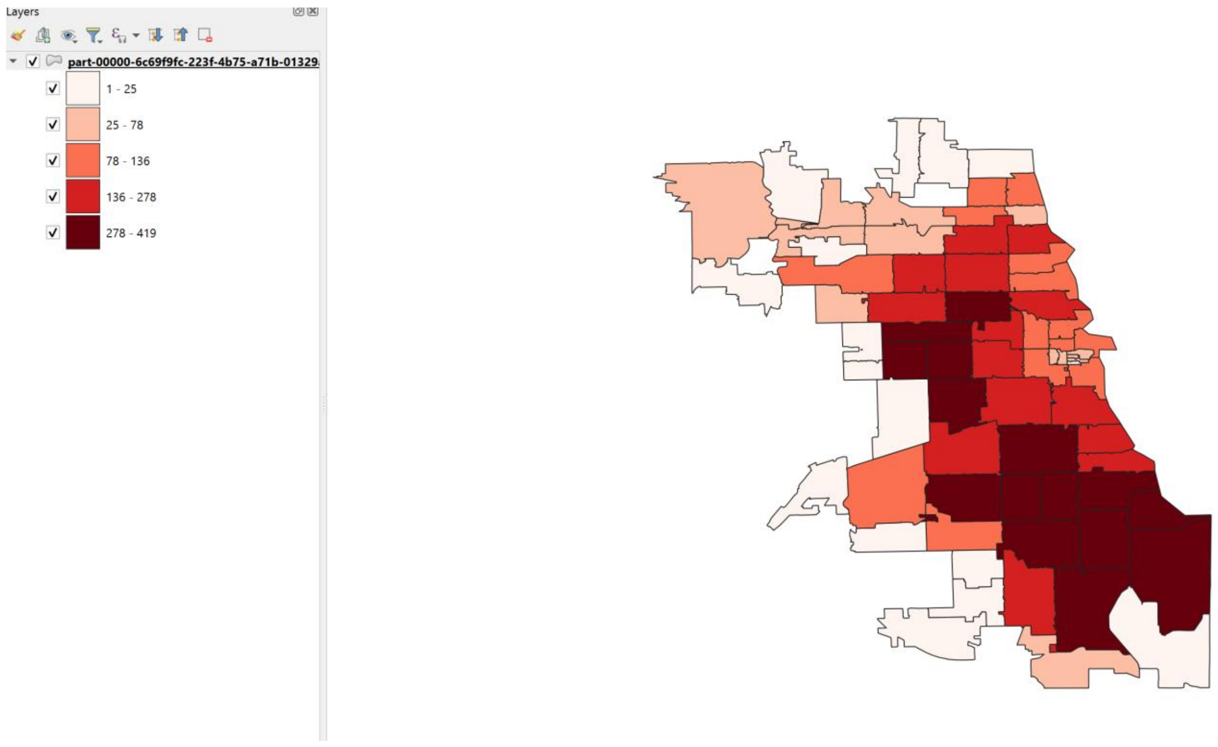


```
# ----------------------------
# Generate Choropleth Map
# ----------------------------

# Directory containing GeoJSON output
directory_path = "/root/BigData/workspace/project/output/ZIPCodeCrimeCount/"

# Find the first JSON file in the directory
json_files = [f for f in os.listdir(directory_path) if f.endswith(".json")]
if not json_files:
    raise FileNotFoundError(f"No JSON files found in directory: {directory_path}")

# Construct the full file path
json_file_path = os.path.join(directory_path, json_files[0])

# Load the JSON file as a GeoDataFrame using GeoPandas
gdf = gpd.read_file(json_file_path)

# Plot the choropleth map
choropleth_map_output_path = "/root/BigData/workspace/project/output/ChoroplethMap.png"
fig, ax = plt.subplots(1, 1, figsize=(12, 8))
gdf.plot(
    column="CrimeCount",          # Column to visualize
    cmap="OrRd",                  # Color map for visualization
    legend=True,                  # Add a legend
    ax=ax,                        # Axes to plot on
    missing_kwds={"color": "lightgrey"}  # Handle missing geometries
)
ax.set_title("Crime Count by ZIP Code in Chicago", fontsize=16)
ax.set_axis_off()  # Remove the axis for a cleaner look

# Save the choropleth map as an image
plt.savefig(choropleth_map_output_path, dpi=300, bbox_inches="tight")
print(f"Choropleth map saved successfully to {choropleth_map_output_path}")

# Stop the Spark session
spark.stop()


```

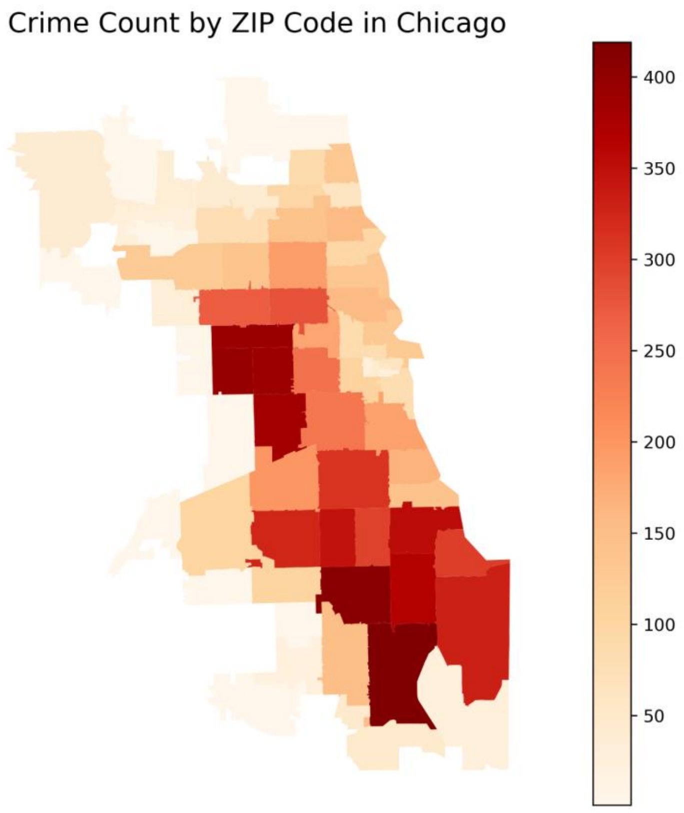
Outcome

1. Aggregated crime data:
   - Grouped by ZIP code, with total crime counts calculated for each area.
2. Spatial dataset saved as:
   - A **GeoJSON file** compatible with QGIS and other geospatial tools.
3. Visualization:
   - A **choropleth map** showing crime intensity by ZIP code.

QGIS Choropleth Map:



GeoPandas Choropleth Map:



Task 3 - Temporal Analysis

Task Objective

The goal of Task 3 was to analyze crime trends over a specific date range, filter crimes by type, and generate a bar chart to visualize the number of crimes for each type within the selected range.

Methodology

Step 1: Load Enriched Data

- The enriched dataset saved in **Parquet format** during Task 1 was loaded into Spark.

Step 2: Accept User Input

- The start and end dates for the analysis were provided as command-line arguments in the format MM/DD/YYYY.

Step 3: Filter Data by Date Range

- Crimes were filtered using the Date column, ensuring that only crimes within the specified range were included in the analysis.
- **SQL Query**:
    - Used Spark SQL's to_date function to parse and filter the dates.

Step 4: Aggregate Crimes by Type

- The filtered data was grouped by the PrimaryType column to count the number of crimes for each type.
- **Output Columns**:
    - PrimaryType: The type/category of crime (e.g., theft, assault).
    - CrimeCount: The total number of crimes of that type within the specified date range.

Step 5: Save Results

- The aggregated data was saved as a **CSV file** for storage and sharing.

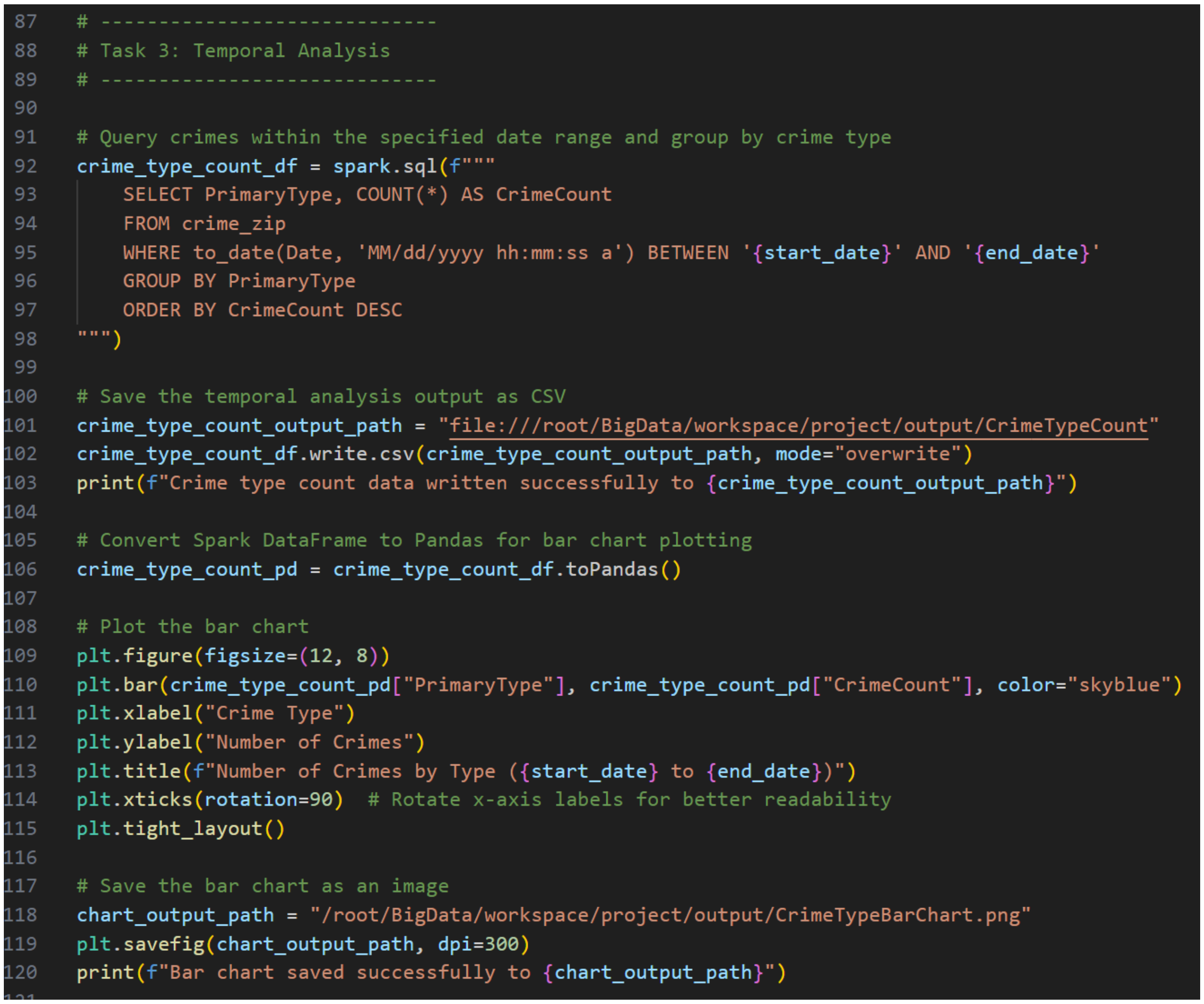
Step 6: Generate a Bar Chart

- The results were converted to a **Pandas DataFrame**, and a bar chart was generated using **Matplotlib**:
    - X-Axis: Crime types (PrimaryType).
    - Y-Axis: Number of crimes (CrimeCount).
    - Title: Included the date range to provide context.

Tools and Libraries Used

- **Apache Spark**: For filtering, grouping, and aggregating data.
- **Matplotlib**: For creating a bar chart to visualize the number of crimes by type.

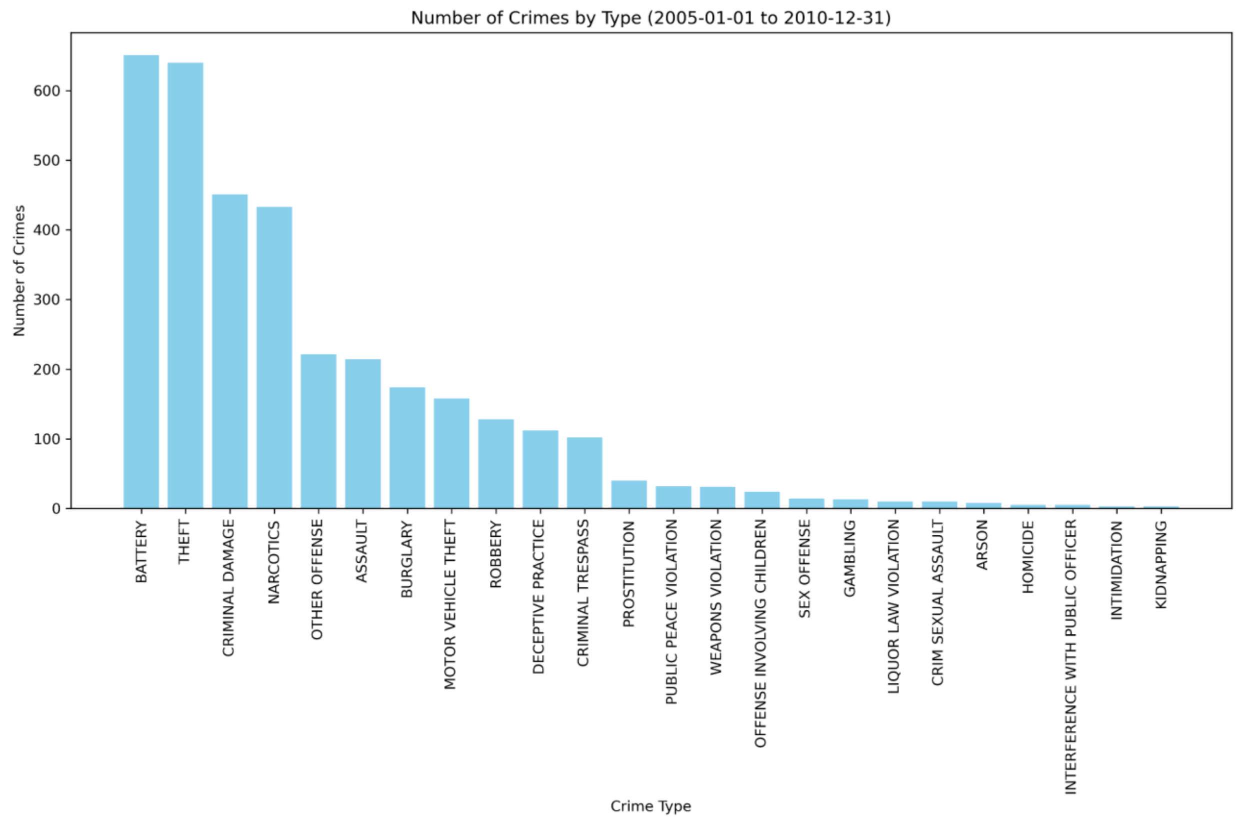
Code Snippet



```python
# ----------------------------
# Task 3: Temporal Analysis
# ----------------------------

# Query crimes within the specified date range and group by crime type
crime_type_count_df = spark.sql(f"""
    SELECT PrimaryType, COUNT(*) AS CrimeCount
    FROM crime_zip
    WHERE to_date(Date, 'MM/dd/yyyy hh:mm:ss a') BETWEEN '{start_date}' AND '{end_date}'
    GROUP BY PrimaryType
    ORDER BY CrimeCount DESC
""")

# Save the temporal analysis output as CSV
crime_type_count_output_path = "file:///root/BigData/workspace/project/output/CrimeTypeCount"
crime_type_count_df.write.csv(crime_type_count_output_path, mode="overwrite")
print(f"Crime type count data written successfully to {crime_type_count_output_path}")

# Convert Spark DataFrame to Pandas for bar chart plotting
crime_type_count_pd = crime_type_count_df.toPandas()

# Plot the bar chart
plt.figure(figsize=(12, 8))
plt.bar(crime_type_count_pd["PrimaryType"], crime_type_count_pd["CrimeCount"], color="skyblue")
plt.xlabel("Crime Type")
plt.ylabel("Number of Crimes")
plt.title(f"Number of Crimes by Type ({start_date} to {end_date})")
plt.xticks(rotation=90)  # Rotate x-axis labels for better readability
plt.tight_layout()

# Save the bar chart as an image
chart_output_path = "/root/BigData/workspace/project/output/CrimeTypeBarChart.png"
plt.savefig(chart_output_path, dpi=300)
print(f"Bar chart saved successfully to {chart_output_path}")
```

Outcome

1. **Filtered Data**:
   - Crimes filtered based on the user-provided date range.
2. **Aggregated Results**:
   - Crime counts grouped by type and saved as a **CSV file**.
3. **Visualization**:
   - A **bar chart** showing the distribution of crimes by type within the specified date range.



Conclusion and Future Work

Summary of Accomplishments

This project successfully analyzed and visualized a sample of Chicago crime data using **Big Data tools** for scalable and efficient data processing. The tasks were divided into three distinct parts:

1. **Task 1: Data Cleaning and Enrichment**:
   - Raw crime data was cleaned, enriched with geospatial attributes, and saved in Parquet format for optimized storage and querying.
2. **Task 2: Spatial Analysis**:
   - Crime data was aggregated by ZIP code, and a choropleth map was created in QGIS as per the project requirements.
   - An automated map generation process using GeoPandas and Matplotlib was added as an optional enhancement.
3. **Task 3: Temporal Analysis**:

    - Crime data was filtered by date range and grouped by type to identify trends.
    - A bar chart was generated to visualize the distribution of crimes by type over a specific period.

Key Learnings

- **Big Data Tools**:
    - Using **Apache Spark** and **Apache Sedona**, the project showcased the ability to handle large datasets with scalability and efficiency.
- **Geospatial Analysis**:
    - Implementing spatial joins and working with geospatial data formats like shapefiles and GeoJSON emphasized the importance of tools like Sedona and QGIS for spatial tasks.

Project Outputs

1. **Processed Data**:
    - Enriched dataset saved in **Parquet format** for scalable querying.
2. **Spatial Analysis**:
    - Aggregated crime data by ZIP code, saved in **GeoJSON format**.
    - Choropleth maps generated both manually (QGIS) and programmatically (GeoPandas).
3. **Temporal Analysis**:
    - Aggregated crime data by type for user-specified date ranges.
    - Results saved as a **CSV file**, with visualizations in the form of bar charts.

Future Work

1. **Real-Time Data Processing**:
    - Integrate streaming tools (e.g., **Apache Kafka**) to process real-time crime data feeds.
2. **Advanced Visualizations**:

    - Use interactive visualization tools like **Plotly** or **Tableau** for dynamic map and chart generation.
3. **Predictive Analytics**:
    - Incorporate machine learning models to predict crime hotspots and future trends.
4. **Larger Datasets**:
    - Scale the analysis to include the full Chicago crime dataset, testing the limits of the current architecture.

Repository Overview

- The GitHub repository contains:
    - **ChicagoCrimesAnalysis.py**: The main Python script implementing the entire workflow.
    - **Dataset Samples**: Sample datasets used for testing and analysis.
    - **Output Files**:
        - Processed datasets in **Parquet** and **GeoJSON** formats.
        - Visualizations (choropleth maps and bar charts) in **PNG** format.
    - **Documentation**: A detailed project report and a README file.
    - **Presentation**: A summary of the project methodology and results.
