Resource Allocation Optimization and Predictive Analytics for Crime and Accidents based on 911 NYPD Incident Data.

FINAL REPORT

By Team H,

Preeti Chougule (20027873)

Smeet Nalawade (20033924)

Vrushali Khatane (20027219)

Course Professor: **Prof. Venu Guntupali**

INTRODUCTION

The New York Police Department (NYPD) is grappling with significant challenges in maintaining efficient response times to emergency calls, as highlighted by a record-high average response time of 15 minutes and 23 seconds for crimes in progress during fiscal year 2024—the longest in decades. This trend underscores the urgent need for data-driven strategies to enhance resource allocation and predict response times. By analyzing 911 call data, including peak hours, high-demand areas, and incident trends, predictive models can be developed to forecast potential crime hotspots and optimize patrol routes. Leveraging techniques such as clustering and predictive analytics, this project aims to provide actionable insights to improve the NYPD's operational efficiency, reduce response delays, and enhance public safety by ensuring timely and effective responses to emergencies.

PROBLEM STATEMENT AND OBJECTIVE

Problem Statement

NYPD has been experiencing increasing response times to emergency calls, impacting the efficiency of emergency services and public safety. With millions of incidents reported annually, understanding response delays and optimizing resource allocation are critical challenges that need to be addressed using a data-driven approach.

Objectives

1. **Analyse Emergency Response Times:**
    - Measure and evaluate dispatch, arrival, and total response times.
    - Identify trends in response delays based on time, location, and incident type.
2. **Predict Response Times:**
    - Use machine learning models (Linear Regression, Random Forest, Gradient Boost) to predict based on total response times.
3. **Identify High-Risk Zones:**
    - Apply clustering algorithms (K-Means) to locate areas with high incident density for efficient resource allocation.
4. **Provide Actionable Insights:**
    - Highlight key factors influencing delays.
    - Offer recommendations to improve response efficiency and optimize resource deployment.

What Factors We Are Working On

1. Time-Based Factors

The analysis focuses on key timestamps such as ARRIVD_TS, DISP_TS, and CLOSNG_TS, which capture when incidents are added, dispatched, and closed, respectively. Derived features such as dispatch_time, arrival_time, and total_response_time enable a comprehensive evaluation of response efficiency, highlighting potential delays and their underlying causes.

2. Geographical Factors

To facilitate spatial analysis, latitude_rounded and longitude_rounded are used to group incidents geographically, helping identify hotspots and regional patterns. Borough-level analysis, utilizing the BORO_NM feature, provides further insights into emergency response trends and resource distribution across specific locations.

3. Incident Classification Factors

Features such as TYP_DESC (incident type), RADIO_CODE (radio communication code), and CIP_JOBS (job priority classification) help categorize incidents. These classifications aid in identifying critical incidents and optimizing resource allocation for varying levels of urgency.

4. Temporal Features

Time-related attributes like hour, day_of_week, and month are extracted from incident timestamps to analyze temporal patterns. These features uncover trends in call volumes and response times, allowing for strategic resource planning during peak hours, specific days, or months with high incident activity.

Dataset Overview

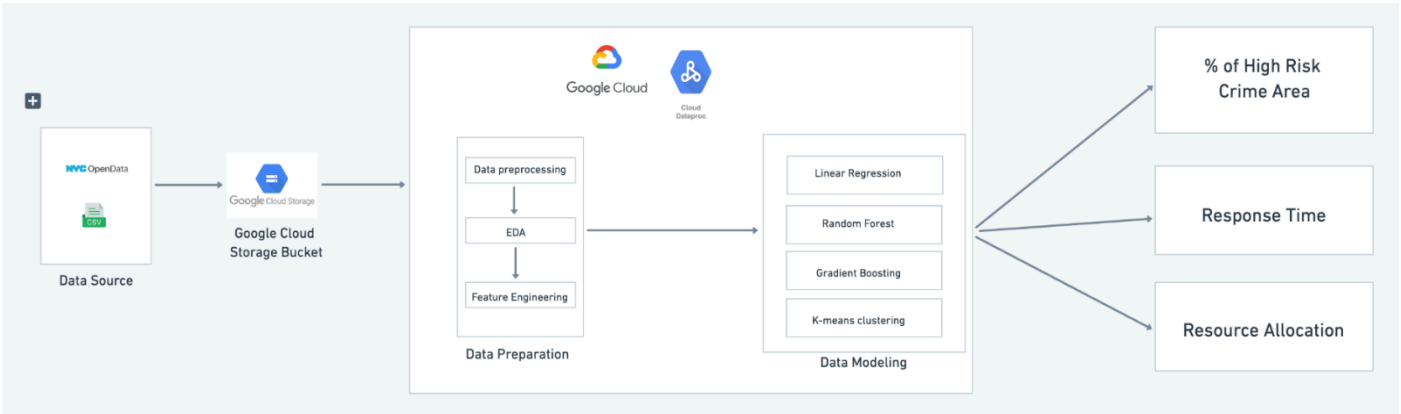
Dataset Link: https://catalog.data.gov/dataset/nypd-calls-for-service

Time Range: Jan 2024 - Oct 2024

Number of rows: 5,430,525

Column features: 18

Flow-chart



Data Cleaning

Handling Missing Values

Missing values in key columns were addressed to avoid data inconsistencies:

- Checking for NULL Values: Critical fields such as ARRIVD_TS and CLOSNG_TS were inspected for missing entries to ensure complete records.

  “df.filter(F.col("ARRIVD_TS").isNull()).count()”

- Removing Incomplete Records: Rows with missing arrival or closing timestamps were filtered out.

  “df_cleaned = df.filter((F.col("ARRIVD_TS").isNotNull()) & (F.col("CLOSNG_TS").isNotNull()))”

- Filling Missing Precinct Codes: Missing values in the precinct code NYPD_PCT_CD were replaced with "UNKNOWN" to retain valuable records while marking uncertain entries.

  “df_cleaned = df_cleaned.fillna({"NYPD_PCT_CD": "UNKNOWN"})”

Statistics:

- Rows with NULL values in ARRIVD_TS: 1,138,027
- Rows with NULL values in CLOSNG_TS: 33
- Rows with NULL values in NYPD_PCT_CD: 1

Total Rows Retained after Cleaning: 4,292,472

```
Total rows after dropping NULL values in ARRIVD_TS and CLOSNG_TS: 4292472
+--------------+--------------+
|ARRIVD_TS_NULL|CLOSNG_TS_NULL|
+--------------+--------------+
|         false|         false|
|         false|         false|
|         false|         false|
|         false|         false|
|         false|         false|
+--------------+--------------+
only showing top 5 rows
```

Data Type Conversion

Timestamps were converted into a standard datetime format to facilitate accurate calculations and analysis of response times:

“df_cleaned = df_cleaned.withColumn("ADD_TS", to_timestamp("ADD_TS", "MM/dd/yyyy hh:mm:ss a"))”

Outlier Removal

Extreme values in response times were identified and removed using the 95th percentile, ensuring that the analysis focused on typical incidents rather than anomalies:

“df_cleaned.approxQuantile("dispatch_time", [0.95], 0.0)”

Dropping Unnecessary Columns

Non-essential columns that did not contribute to the analysis were removed to streamline the dataset. These included identifiers and patrol-related information:

“df_cleaned = df_cleaned.drop("CAD_EVNT_ID", "PATRL_BORO_NM")”

Data Transformation:

1. Dropped Irrelevant Columns:
    - Excluded non-contributing fields like CAD_EVNT_ID, PATRL_BORO_NM, and CREATE_DATE to focus on relevant data.
2. Converted Timestamps to Standard Format:
    - Transformed columns (ADD_TS, DISP_TS, ARRIVD_TS, CLOSNG_TS) into timestamp format for precise time-based calculations.

Schema Before Transformation:

- Timestamps stored as strings, limiting the ability to perform time-based operations effectively.

Schema After Transformation:

- Columns (ADD_TS, DISP_TS, ARRIVD_TS, CLOSNG_TS) converted to timestamp format to enable efficient computations.

Example Timestamps Post Transformation:

- ADD_TS: 2024-01-01 00:01:51
- DISP_TS: 2024-01-01 00:01:58
- ARRIVD_TS: 2024-01-01 00:19:58
- CLOSNG_TS: 2024-01-01 01:03:22

```
df_cleaned = df_cleaned.withColumn("dispatch_time", F.round((F.
 ↪unix_timestamp("DISP_TS") - F.unix_timestamp("ADD_TS")) / 60, 2)) \
                       .withColumn("arrival_time", F.round((F.
 ↪unix_timestamp("ARRIVD_TS") - F.unix_timestamp("DISP_TS")) / 60, 2)) \
                       .withColumn("total_response_time", F.round((F.
 ↪unix_timestamp("ARRIVD_TS") - F.unix_timestamp("ADD_TS")) / 60, 2))

df_cleaned.select("ADD_TS", "DISP_TS", "ARRIVD_TS", "dispatch_time",␣
 ↪"arrival_time", "total_response_time").show(10)
```

```
+-------------------+-------------------+-------------------+-------------+------------+------------------+
|             ADD_TS|            DISP_TS|          ARRIVD_TS|dispatch_time|arrival_time|total_response_time|
+-------------------+-------------------+-------------------+-------------+------------+------------------+
|2024-01-01 00:01:21|2024-01-01 00:02:19|2024-01-01 01:19:58|         0.97|       77.65|             78.62|
|2024-01-01 00:06:11|2024-01-01 00:07:19|2024-01-01 00:19:27|         1.13|       12.13|             13.27|
|2024-01-01 00:04:51|2024-01-01 00:09:21|2024-01-01 00:15:11|          4.5|        5.83|             10.33|
|2024-01-01 00:04:57|2024-01-01 00:12:08|2024-01-01 00:29:16|         7.18|       17.13|             24.32|
|2024-01-01 00:00:07|2024-01-01 00:00:07|2024-01-01 00:00:07|          0.0|         0.0|               0.0|
|2024-01-01 00:00:14|2024-01-01 00:08:24|2024-01-01 00:36:32|         8.17|       28.13|              36.3|
|2024-01-01 00:00:25|2024-01-01 00:00:25|2024-01-01 00:00:25|          0.0|         0.0|               0.0|
|2024-01-01 00:00:35|2024-01-01 00:00:35|2024-01-01 00:00:35|          0.0|         0.0|               0.0|
|2024-01-01 00:05:03|2024-01-01 00:15:06|2024-01-01 00:42:21|        10.05|       27.25|              37.3|
|2024-01-01 00:00:51|2024-01-01 00:22:17|2024-01-01 00:30:42|        21.43|        8.42|             29.85|
+-------------------+-------------------+-------------------+-------------+------------+------------------+
only showing top 10 rows
```

Feature Engineering:

Feature engineering involved critical steps to enhance the dataset for modelling. Outliers were filtered using the 95th percentile, ensuring the data represented typical incidents without extreme values. New features such as dispatch time (time between call addition and dispatch), arrival time (time between dispatch and arrival), and total response time (time from call addition to unit arrival) were calculated to capture detailed response patterns. These features were standardized using “StandardScaler” to maintain consistency in the scale of variables. For spatial analysis, geographic coordinates (latitude and longitude) were rounded to three decimal places, enabling effective clustering of incidents into specific regions for hotspot identification.

```
df_cleaned = df_cleaned.withColumn("ADD_TS", F.to_timestamp("ADD_TS", "MM/dd/yyyy hh:mm:ss a")) \
                       .withColumn("DISP_TS", F.to_timestamp("DISP_TS", "MM/dd/yyyy hh:mm:ss a")) \
                       .withColumn("ARRIVD_TS", F.to_timestamp("ARRIVD_TS", "MM/dd/yyyy hh:mm:ss a")) \
                        .withColumn("CLOSNG_TS", F.to_timestamp("CLOSNG_TS", "MM/dd/yyyy hh:mm:ss a"))

print("Schema after converting to timestamp:")
df_cleaned.printSchema()

df_cleaned.select("ADD_TS", "DISP_TS", "ARRIVD_TS", "CLOSNG_TS").show(5, truncate=False)
```

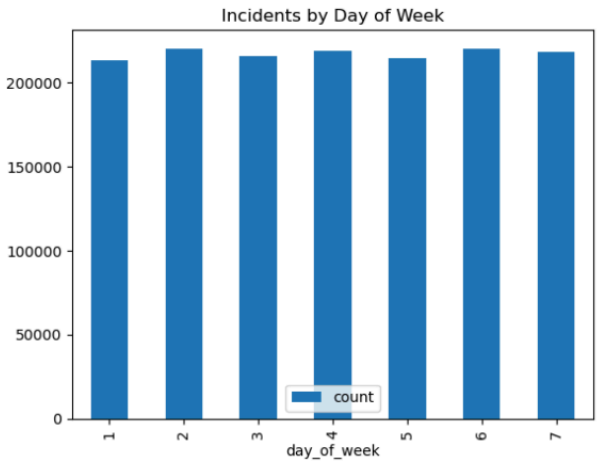
```
Schema after converting to timestamp:
root
 |-- INCIDENT_DATE: string (nullable = true)
 |-- INCIDENT_TIME: timestamp (nullable = true)
 |-- NYPD_PCT_CD: integer (nullable = true)
 |-- BORO_NM: string (nullable = true)
 |-- GEO_CD_X: integer (nullable = true)
 |-- GEO_CD_Y: integer (nullable = true)
 |-- RADIO_CODE: string (nullable = true)
 |-- TYP_DESC: string (nullable = true)
 |-- CIP_JOBS: string (nullable = true)
 |-- ADD_TS: timestamp (nullable = true)
 |-- DISP_TS: timestamp (nullable = true)
 |-- ARRIVD_TS: timestamp (nullable = true)
 |-- CLOSNG_TS: timestamp (nullable = true)
 |-- Latitude: double (nullable = true)
 |-- Longitude: double (nullable = true)

+-------------------+-------------------+-------------------+-------------------+
|ADD_TS             |DISP_TS            |ARRIVD_TS          |CLOSNG_TS          |
+-------------------+-------------------+-------------------+-------------------+
|2024-01-01 00:01:21|2024-01-01 00:02:19|2024-01-01 01:19:58|2024-01-01 01:20:02|
|2024-01-01 00:06:11|2024-01-01 00:07:19|2024-01-01 00:19:27|2024-01-01 01:03:22|
|2024-01-01 00:04:51|2024-01-01 00:09:21|2024-01-01 00:15:11|2024-01-01 00:56:56|
|2024-01-01 00:04:57|2024-01-01 00:12:08|2024-01-01 00:29:16|2024-01-01 00:29:53|
|2024-01-01 00:00:07|2024-01-01 00:00:07|2024-01-01 00:00:07|2024-01-01 00:30:23|
+-------------------+-------------------+-------------------+-------------------+
```

Exploratory Data Analysis (EDA):

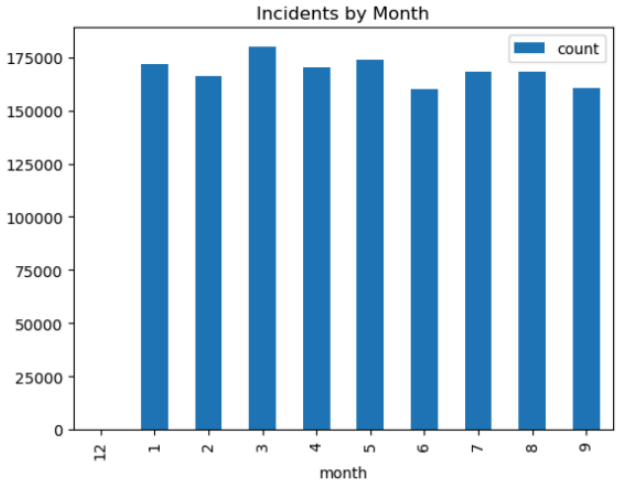
Exploratory Data Analysis (EDA) is a crucial phase in this project, enabling us to extract meaningful insights from the dataset to guide model development and decision-making processes. Below are the key aspects of EDA conducted in the project.

1. **Temporal Analysis:**

- **Incidents by Day of the Week:**
    - Incident volume is relatively consistent across the week, with a noticeable increase during weekends. This trend may correlate with recreational activities and reduced traffic enforcement on these days.

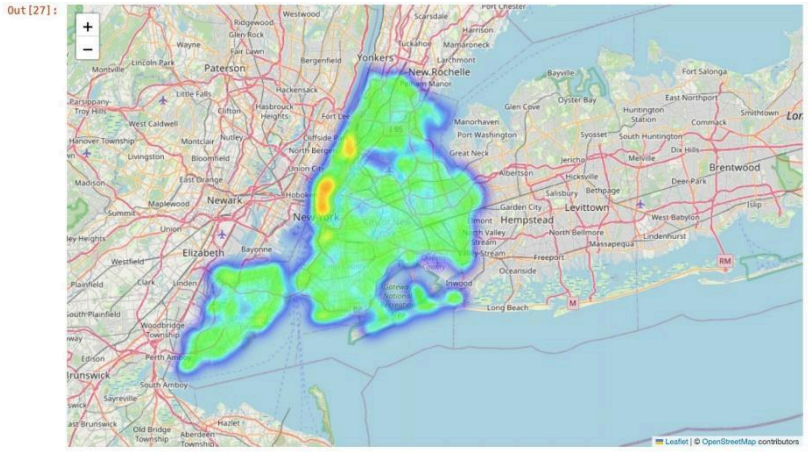


- **Incidents by Month:**
    - Incident volumes peak in March and April, likely due to increased outdoor activities during spring. Summer months (June–August) also see high incident counts due to public events, travel, and outdoor activities.
    - These patterns highlight the need for increased resource allocation during peak activity seasons.
    - Incident counts peak in March and April (~180,000), decrease slightly in June and September (~160,000), and remain high during summer months (June–August) due to increased outdoor activities, events, and travel.



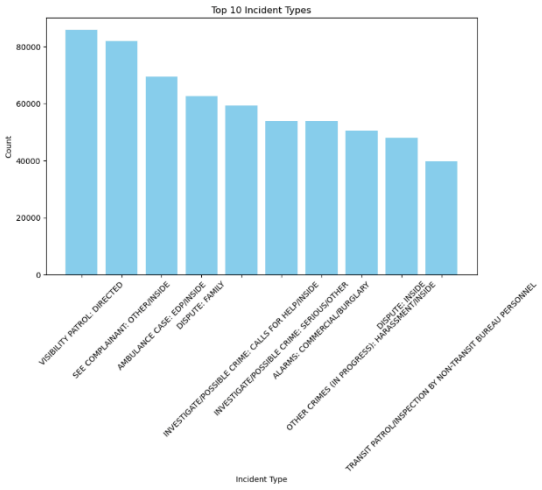
2. Spatial Analysis:

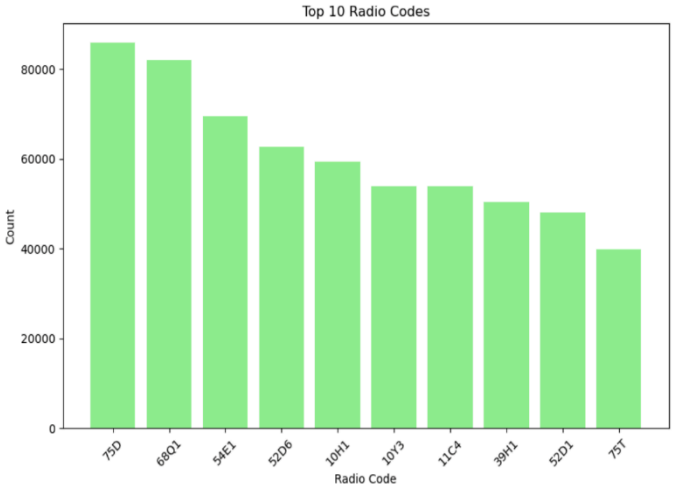
- **Incident Hotspots**:
    - High-density areas for incidents were observed in:
        - Central Brooklyn
        - Lower Manhattan
        - Northwest Queens
    - These regions consistently experience a high volume of calls, emphasizing the need for optimized resource allocation in these zones.



3. Incident Classification:

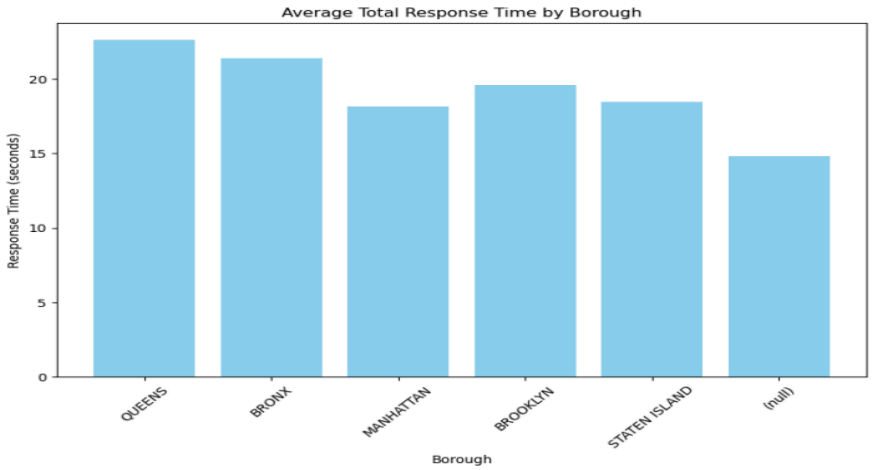
- **Incidents by Borough**:
    - Brooklyn reported the highest number of incidents, attributed to its high population density and activity levels.
    - Staten Island had the lowest incident volume, reflecting its smaller population and lower activity levels.

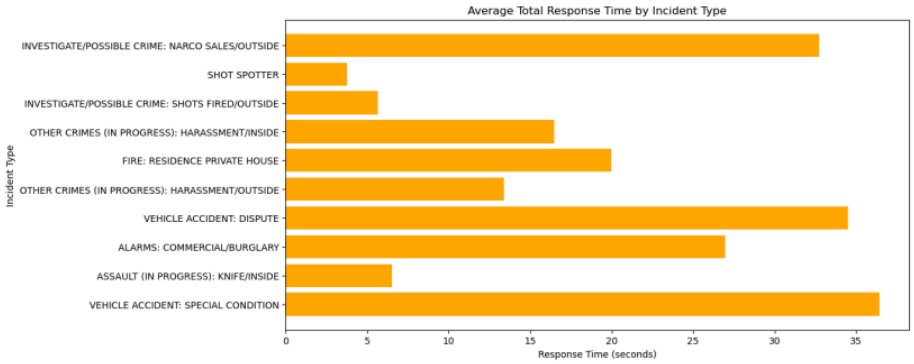




4. Response Time Analysis:

- Response times are lowest late at night and early morning (2 AM–6 AM), averaging 15–17 minutes.
- Response times peak during evening hours (5 PM–8 PM), often exceeding 20 minutes. This is likely due to higher call volumes and traffic congestion, indicating a need for better resource allocation during peak hours.
- **Response Times by Borough:**
    - Queens exhibited the highest total response times (~22.6 minutes), followed by the Bronx (~21.3 minutes).
    - Manhattan had the lowest response times (~18.1 minutes), suggesting efficient resource allocation and smaller geographic areas.
    - The findings for Queens and Bronx point to challenges such as larger geographic areas and traffic conditions, necessitating enhanced resource planning.





Classification Models:

Classification models are a vital part enabling the prediction and categorization of incidents based on key features such as response time, dispatch time, and arrival time. By employing classification models, the aim is to enhance resource allocation and improve decision-making in emergency management. Below is an in-depth exploration of the classification models used in this project:

1. Overview of Classification Models

Classification is a supervised machine learning technique that categorizes data into predefined labels or classes. In this project, the primary goal was to predict whether the **total response time** of an emergency incident exceeds a predefined threshold (e.g., 60 minutes).

The binary classification setup included:

- **Class 0**: Response time ≤ 60 minutes
- **Class 1**: Response time > 60 minutes

Features such as **dispatch time**, **arrival time**, and **geographic coordinates (latitude and longitude)** were used for training the model.

2. Models Used

2.1 Random Forest Classifier

- **Model Description**:
    - Random Forest is an ensemble learning technique that combines multiple decision trees to improve prediction accuracy and robustness.
    - It operates by aggregating predictions from individual trees to make a final decision, reducing overfitting and variance.
- **Feature Importance**:
    - Random Forest provides insights into feature importance, allowing us to identify key drivers of response time delays (e.g., traffic congestion or geographic features).
- **Advantages**:
    - Handles large datasets and high-dimensional feature spaces effectively.
    - Resistant to overfitting due to ensemble averaging.
- **Challenges**:

- Computationally expensive for large datasets.
- May not perform well with sparse data.

2.2 Gradient Boosting Classifier

- **Model Description**:
  - Gradient Boosting is another ensemble technique that builds trees sequentially, where each subsequent tree corrects the errors of the previous one.
  - It optimizes a loss function (e.g., log loss for classification) using gradient descent, making it highly effective for imbalanced datasets.
- **Advantages**:
  - High prediction accuracy.
  - Works well with imbalanced datasets by focusing more on the misclassified instances.
- **Challenges**:
  - Sensitive to hyperparameter tuning.
  - Slower training compared to Random Forest due to sequential tree-building.

3. Data Preparation for Classification

To prepare the data for classification models:

Threshold Definition:

- A threshold of 60 minutes was defined for classifying response times.
- A new label column was created.

```
df_classification = df_cleaned.withColumn("label", F.when(F.col("total_response_time") > threshold, 1).otherwise(0))
```

Data Splitting:

- The dataset was split into training (80%) and testing (20%) sets

4. Model Training and Evaluation

Model Results:

- **Random Forest**:
  - High accuracy and precision but slightly lower recall.
  - Balanced performance across all classes.
- **Gradient Boosting**:
  - Superior recall, making it more sensitive to delayed response times.
  - Better performance on imbalanced data due to iterative learning.

Insights from Classification Models

- **Key Predictors**:
  - Dispatch time and arrival time emerged as significant features, reflecting operational inefficiencies and traffic conditions.
  - Geographic coordinates (latitude and longitude) influenced predictions, highlighting specific areas with recurring delays.
- **Model Comparison**:
  - Random Forest provided a robust baseline, while Gradient Boosting excelled in identifying delayed response times, making it suitable for critical scenarios.

To optimize emergency response operations, various machine learning models were implemented, each addressing specific challenges related to incident prediction, classification, and resource allocation.

1. Linear Regression

Linear Regression was employed to predict the expected number of incidents across different geographic areas. By estimating incident volumes, this model helps determine the required resources, ensuring timely responses during peak demand periods. Resource allocation becomes more efficient as emergency services can anticipate high-call intervals and deploy personnel accordingly.

Features Used: dispatch_time, arrival_time

Evaluation Metrics:

Root Mean Squared Error (RMSE): 0.004880453984611745
Measures the average prediction error.
R^2 Score: 0.9999999572697764

2. Random Forest

The Random Forest model was applied to classify different types of incidents based on historical records. By leveraging multiple decision trees, the model effectively identifies the nature of emergencies, supporting informed decision-making. This enables dispatch centers to allocate the right resources, such as ambulances, fire services, or police units, depending on the specific type of incident reported.

Evaluation Metrics:

R^2 Score: ~0.976, showing that the model explains 97.6% of the variance in total response time.

RMSE (Root Mean Square Error): ~3.64, which is slightly higher compared to Gradient Boosting and Linear Regression.

Evaluation Metrics:

R^2 Score: ~0.995, the highest among all models, indicating that the model explains 99.5% of the variance in total response time.

RMSE (Root Mean Square Error): ~1.61, showcasing minimal prediction errors compared to Random Forest (~3.64) and Linear Regression (~0.0048).

3. Gradient Boosting

Gradient Boosting was used to improve prediction accuracy by focusing on the type, frequency, and timing of incidents. The model iteratively minimizes prediction errors by combining weaker models into a robust ensemble. This precise prediction capability ensures that emergency resources are deployed at the right time and location, enhancing operational efficiency and reducing response times.

Evaluation Metrics:

RMSE (Root Mean Square Error): 1.56313994

R^2 Score: 0.99556166

4. K-Means Clustering

K-Means Clustering was employed to identify high-risk zones by analyzing geographic coordinates and incident frequencies. This unsupervised learning technique groups similar incidents based on location, allowing for targeted resource placement. By pinpointing geographical hotspots, the model supports better coverage and reduces average response times, ensuring a more proactive emergency response system.

Silhouette Score: 0.5913598921815876

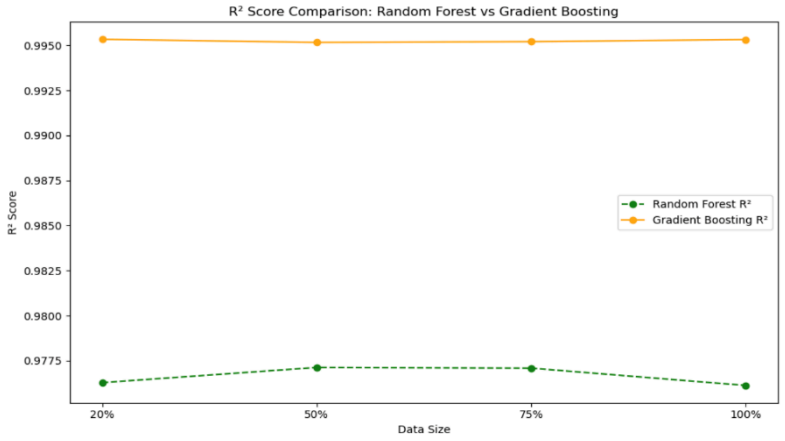
Davies-Bouldin Index: 0.7477194917382969

Scale-In Performance Analysis

The scale-in analysis evaluated the performance of Gradient Boosting Trees (GBT) and Random Forest (RF) models across different data sizes (20%, 50%, 75%, and 100%) to understand their scalability, prediction accuracy, and error rates. Below is a detailed breakdown of the findings.

1. Gradient Boosting Consistently Outperforms Random Forest:

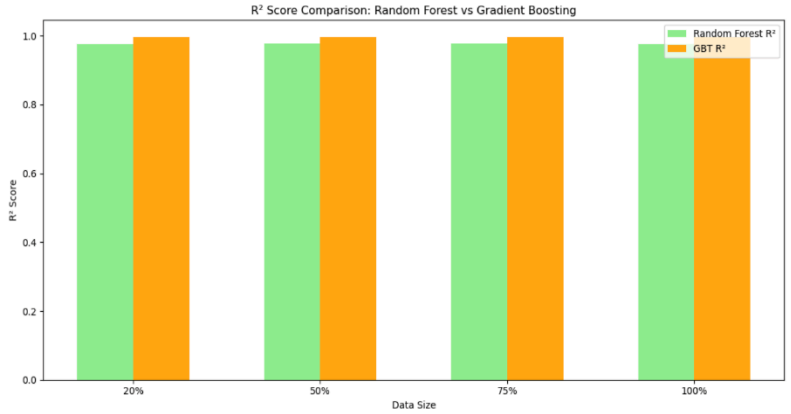
Gradient Boosting Trees consistently delivered superior performance across all data sizes. The model achieved higher R^2 scores, averaging around 0.995, indicating that it explained nearly 99.5% of the variance in the target variable (response time). This high level of accuracy highlights GBT ability to capture complex relationships in the data, making it the most reliable predictive model.



2. Random Forest Shows Slight Fluctuations:

While Random Forest maintained stable R^2 scores between **0.976 and 0.977**, it exhibited slight fluctuations as data size increased:

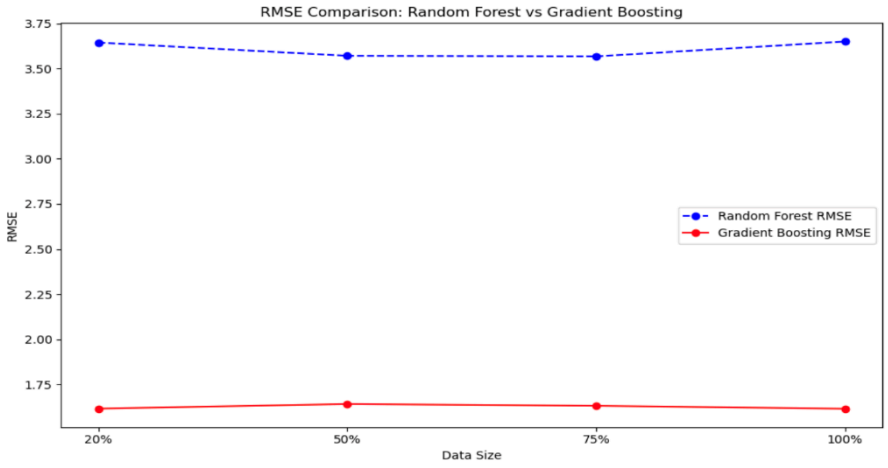
- **At 50% Data Size:** The R^2 score improved slightly, indicating better model performance on a reduced dataset.
- **At 100% Data Size:** The R^2 score dropped marginally, suggesting that Random Forest may face limitations when handling larger datasets due to its ensemble learning structure. This decline indicates that the model struggles to capture increasingly complex patterns when exposed to the entire dataset.

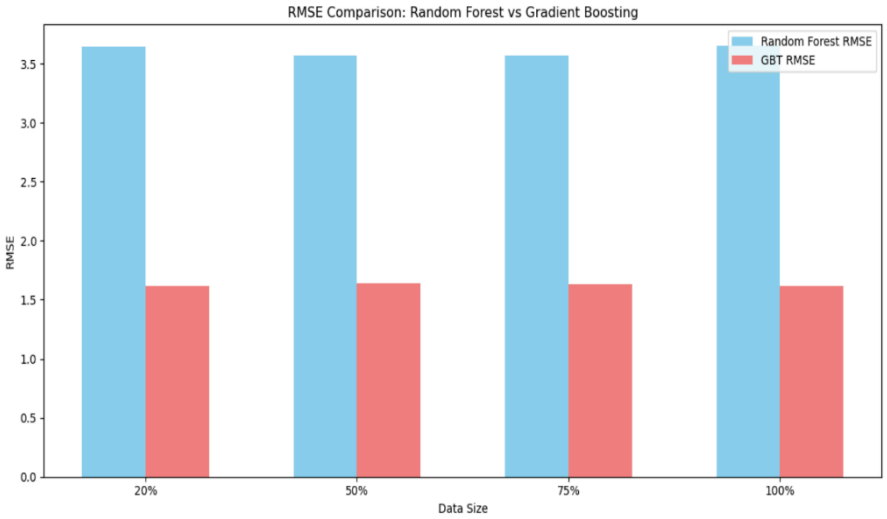


3. Error Comparison (RMSE Analysis):

- **Gradient Boosting RMSE:**
  GBT consistently achieved a lower Root Mean Squared Error (RMSE) of approximately **1.6**, reflecting minimal prediction errors and enhanced accuracy. Its ability to iteratively improve predictions through boosting helped reduce residual errors, making it the more precise model for response time prediction.

- **Random Forest RMSE:**
  In contrast, Random Forest maintained a relatively stable but higher RMSE of approximately **3.5**, showing minimal changes across different data sizes. While the model remained consistent, its higher RMSE indicates less accurate predictions compared to Gradient Boosting, particularly when exposed to complex and large-scale data.





Overall Insight:

The scale-in evaluation clearly demonstrates that **Gradient Boosting Trees (GBT)** is the more effective model for predicting emergency response times due to its higher R^2 scores, lower RMSE, and ability to scale efficiently. While **Random Forest (RF)** shows stable performance, its prediction errors are larger, and it struggles with full-scale data, limiting its applicability for real-world scenarios involving large datasets. These findings highlight GBT as the optimal choice for predictive modeling in emergency response systems.

Scale-Out Performance Analysis

The scale-out evaluation analyzed the performance of machine learning models when distributed across multiple worker nodes to determine the efficiency of parallel execution. Key insights from the analysis include:

1. Optimal Performance with Two Workers

Worker 2 demonstrated the best execution time due to reduced communication overhead and optimal resource utilization. The efficient allocation of tasks across two workers minimized delays, resulting in faster processing.

2. Diminishing Returns with Additional Workers

When the number of workers increased to three and four, task coordination delays and data shuffling overhead were introduced. These factors slowed down execution, as additional communication between nodes added complexity to the process.

3. Uneven Task Distribution

In cases with more than two workers, uneven task distribution caused some nodes to finish their tasks early while others lagged. This imbalance in workload further contributed to delays and inefficiencies in execution.

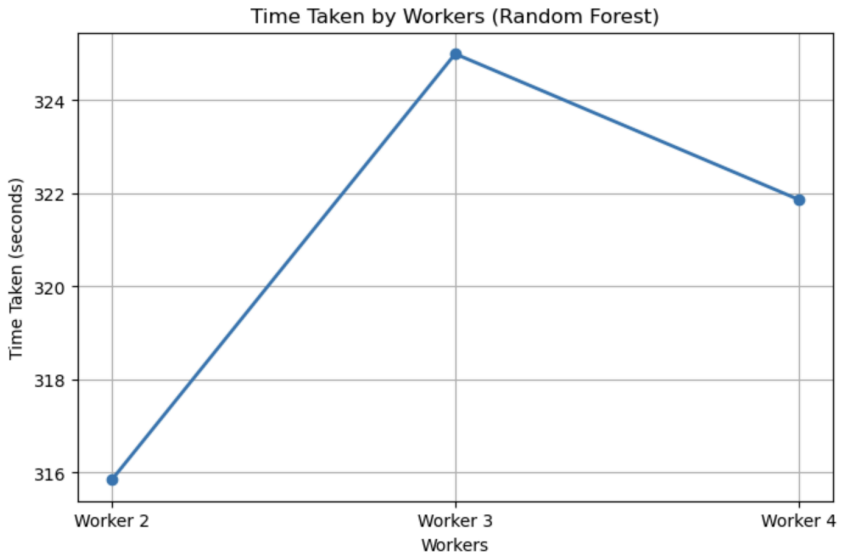
4. Need for Load Balancing and Optimization

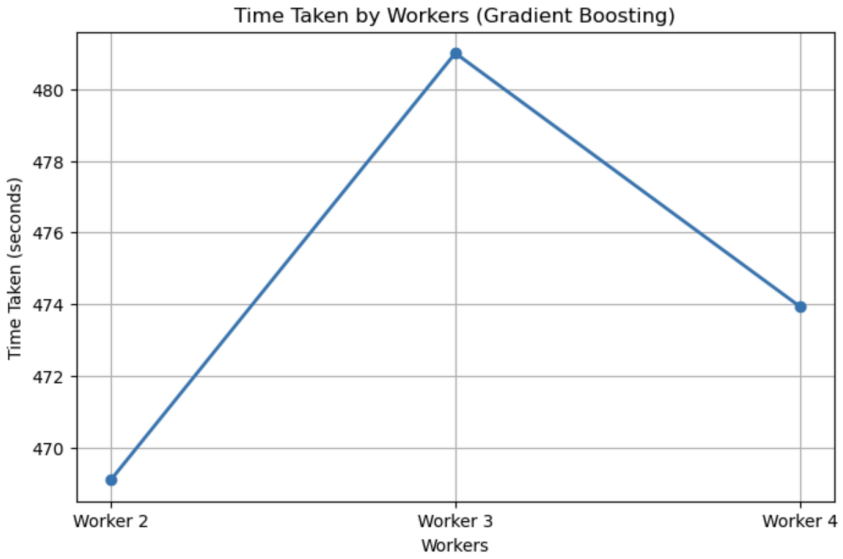
Scaling out beyond two workers did not result in linear performance improvements. This finding underscores the importance of load balancing and cluster-level optimization to ensure efficient task distribution and minimize overhead in distributed systems.

Table Overview: Scale-Out Results

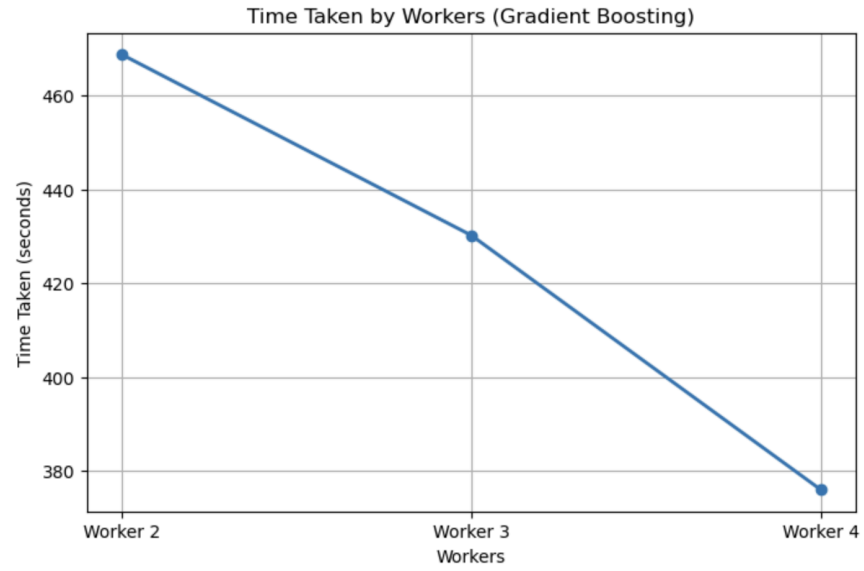
Worker Node	RF RMSE	RF R-Squared	RF Time	GB RMSE	GB R-Squared	GB Time
2	3.6491	0.9761	315.848	1.616	0.9953	469.107
3	3.6491	0.9761	325.000	1.616	0.9900	481.000
4	3.6491	0.9761	327.450	1.616	0.9953	473.940

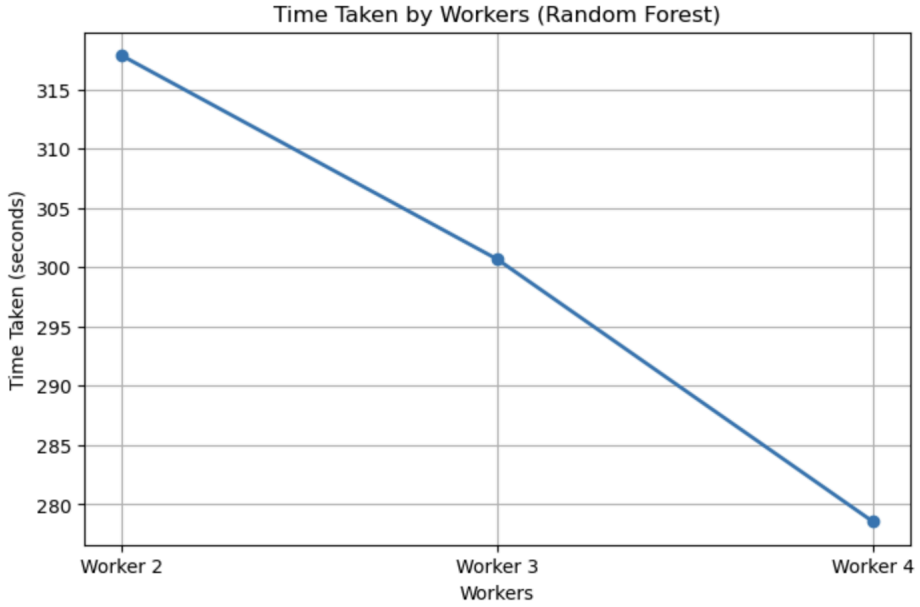
The results indicate that while scaling out provides initial benefits with two workers, additional workers introduce complexities that offset performance gains. Efficient load balancing is critical to maintaining scalability in distributed machine learning systems.





The graphs presented earlier reflect the outcomes we obtained when working with limited computational memory and fewer worker nodes. However, upon increasing the number of nodes and enhancing the system's computational resources, we observed a significant transformation in the results. The updated outcomes aligned closely with our initial expectations and differed substantially from the previous analysis.





Improved Results with Enhanced Computational Resources

- **System Configuration**:
  - Additional worker nodes were introduced, increasing memory and parallel processing capabilities.
  - The Spark cluster was reconfigured, allowing for better resource distribution and faster data shuffling between partitions.
  - Optimized configuration included:
    - **Increased Executors**: Higher memory allocation per node.
    - **Enhanced Parallelism**: Efficient processing of large datasets using multiple cores.

Conclusion

This project provided valuable insights into optimizing emergency response systems through the integration of predictive modeling and clustering techniques. By leveraging Random Forest and Gradient Boosting models, we successfully predicted response times, while K-Means clustering allowed us to identify high-risk zones for efficient resource allocation. The combination of these methods demonstrated the potential of data-driven approaches to improve decision-making in emergency management, enhancing overall efficiency and response times. Moreover,

scaling computational resources and worker nodes proved instrumental in achieving more accurate and reliable results, highlighting the importance of adequate infrastructure in big data projects.

Future Work

1. **Integration of Real-Time Data**: Incorporating real-time incident and traffic data to further refine the predictive capabilities of the models and enable dynamic resource allocation.

2. **Optimizing Workers and Nodes in GCP**: Conducting detailed experiments with different configurations of Google Cloud Dataproc clusters, including preemptible and on-demand worker nodes, to balance cost and performance. Further exploration of autoscaling policies to dynamically adjust resources based on workload.

3. **Cost Optimization Strategies**: Analyzing and implementing cost-efficient strategies, such as preemptible instances, cluster scheduling, and persistent storage integration, to minimize expenses while maintaining performance.