1. INTRODUCTION

The **Sustainable Debris Management** is an innovative, data-driven approach designed to address the critical challenges of urban waste management. By harnessing the power of advanced big data technologies such as Apache Hadoop and Spark, this framework aims to optimize debris collection processes, reduce operational costs, and minimize environmental impact. It offers a scalable, efficient, and secure environment for managing large volumes of waste data, enabling urban authorities to make informed, data-driven decisions.

This framework is capable of handling both structured and unstructured debris data and can be deployed across various environments, including on-premises data centers, cloud platforms, or hybrid infrastructures. By integrating multiple big data components, the framework streamlines data processing and analytics, making it a powerful tool for sustainable waste management.

Key Components of the Sustainable Debris Management Framework:

- **HDFS (Hadoop Distributed File System)**: A fault-tolerant, scalable storage system that manages vast datasets across distributed nodes, ensuring data availability and reliability.
- **MapReduce**: A programming model that supports parallel data processing, enabling large-scale computations across multiple machines for efficient data analysis.
- **Apache Spark**: An in-memory computing engine that accelerates both batch and real-time data processing, making data analysis faster and more responsive.
- **Apache Hive**: A SQL-like querying tool that simplifies data retrieval and management using HiveQL, making data exploration accessible for non-programmers.
- **Apache Pig**: A scripting language that streamlines data transformation and analysis, reducing the complexity of writing extensive code.
- **HBase**: A NoSQL database optimized for fast read and write access to structured data, facilitating quick data retrieval for analytics.
- **Data Science Workbench**: A platform that supports machine learning and advanced analytics, enabling data scientists to build and deploy predictive models for effective waste management.

The Sustainable Debris Management Framework is highly applicable across urban regions seeking efficient and cost-effective waste management solutions. Key advantages include:

- **Scalability & Performance**: The distributed computing architecture allows for the processing of vast debris datasets with high speed and reliability.
- **Real-Time & Batch Processing**: Supports both real-time analytics for immediate decision-making and batch processing for historical data analysis.
- **Data-Driven Decision Making**: Facilitates advanced analytics, machine learning, and predictive modeling to optimize debris collection and resource allocation.
- **Security & Compliance**: Ensures secure data management with encryption, role-based access control, and adherence to environmental standards.
- **Simplified Data Management**: Integrates various tools for data ingestion, transformation, and querying, reducing complexity and manual effort.

By combining powerful data processing frameworks with enterprise-grade security, the Sustainable Debris Management Framework empowers urban authorities to optimize operations, reduce waste accumulation, and create cleaner, more sustainable cities.

2. PROBLEM DEFINITION

Efficient management of urban debris is crucial for minimizing health hazards, reducing environmental pollution, and optimizing resource allocation. Traditional waste management methods often rely on manual monitoring, limited data analysis, and outdated decision-making approaches, making it challenging to handle the increasing volume and complexity of urban waste.

This project aims to address these challenges by utilizing advanced big data technologies, including Apache Hadoop, Spark, and associated data processing tools, to develop a scalable and data-driven debris management framework. The primary objectives are:

- **Efficient Storage and Retrieval**: Utilize the Hadoop Distributed File System (HDFS) for the efficient storage and retrieval of large volumes of debris data, ensuring data availability and reliability.
- **Parallel Processing and Analysis**: Leverage Apache Spark for parallel processing and analysis of large-scale debris datasets, enabling quick insights and faster decision-making.
- **Data Transformation and Cleaning**: Employ Apache Pig for data transformation, cleaning, and aggregation to standardize data for further analysis.
- **SQL-Based Querying**: Use Apache Hive for SQL-based querying and detailed data exploration to derive meaningful patterns and trends from the debris data.
- **Anomaly Detection**: Implement anomaly detection techniques to identify unusual debris weight patterns, which could indicate data errors or unexpected waste accumulation events.

By leveraging the distributed computing capabilities of big data technologies, this project provides an optimized approach for analyzing massive debris datasets. This facilitates improved waste management practices, efficient resource allocation, and enhanced environmental sustainability.

3. CODING

The coding section of your project involves several key steps to implement the Sustainable Debris Management Framework using big data technologies. Here's a brief overview of each part, including the code, its discussion, and interpretation:

3.1 Data Loading and Verification

Objective: Load the debris dataset into a Spark DataFrame and verify its structure.

Key Steps: Initialize a Spark session, load the dataset from HDFS, and display the first few rows and schema of the DataFrame.

```
from pyspark.sql import SparkSession
# Initialize Spark session
spark = SparkSession.builder \
    .appName("Debris Management") \
    .config("spark.hadoop.fs.default", "hdfs://localhost:9000") \
    .getOrCreate()
# Load dataset
df = spark.read.csv("hdfs:///user/debris_dataset.csv", header=True,
inferSchema=True)
df.show(10)
df.printSchema()
```

Discussion:

- **Spark Session Initialization**: This initializes a Spark session with the application name "Debris Management" and configures the Hadoop file system.
- **Data Loading**: The dataset is loaded from HDFS into a DataFrame df. The header=True option indicates that the first row of the CSV file contains column names, and inferSchema=True automatically infers the data types of the columns.
- **Data Verification**: The show(10) method displays the first 10 rows of the dataset, and printSchema() prints the schema of the DataFrame, showing the column names and data types.

Interpretation:

- Successfully loading and verifying the dataset ensures that the data is correctly formatted and ready for analysis. This step is crucial for the accuracy of subsequent analyses.

3.2 Weight-Based Anomaly Detection with Inference

Objective: Identify anomalous weights that deviate significantly from the norm.

Key Steps: Calculate the mean and standard deviation of debris weights, detect outliers based on these statistics, and add an inference column to explain the anomaly status.

```
from pyspark.sql.functions import col, mean, stddev, when, lit
# Calculate mean and standard deviation for 'FL_B'
stats_df = df.agg(
    mean("FL_B").alias("Mean_Weight"),
    stddev("FL_B").alias("StdDev_Weight")
).collect()
# Extract calculated values
mean_weight = stats_df[0]["Mean_Weight"]
stddev_weight = stats_df[0]["StdDev_Weight"]
# Detect anomalies (Outliers)
df = df.withColumn(
    "Anomaly",
    when((col("FL_B") > mean_weight + 2 * stddev_weight) |
         (col("FL_B") < mean_weight - 2 * stddev_weight), "Outlier")
    .otherwise("Normal")
)
# Add inference
df = df.withColumn(
    "Inference",
    lit("Outlier = Unusual debris weight detected | Normal = Weight within
expected range")
)
# Display results
df.select("Class", "FL_B", "Anomaly", "Inference").show(truncate=False)
```

Discussion:

- **Statistical Calculation**: The mean and standard deviation of the FL_B column are calculated and collected into stats_df.
- **Anomaly Detection**: Anomalies are detected by checking if the FL_B values fall outside the range of mean ± 2 standard deviations. These are labeled as "Outlier" or "Normal".

- **Inference Addition**: An inference column is added to provide a textual explanation of the anomaly status.
- **Result Display**: The results are displayed, showing the class, weight, anomaly status, and inference.

Interpretation:

- Detecting anomalies helps identify unusual debris weights, which could indicate measurement errors or unexpected debris types. This step enhances the understanding of data irregularities and supports accurate monitoring.

3.3 Location-Based Debris Concentration Analysis

Objective: Evaluate the distribution of debris across different classes to categorize high, moderate, and low concentrations.

Key Steps: Group the data by class, count the occurrences, and infer concentration levels based on predefined thresholds.

```
from pyspark.sql.functions import count
# Group by 'Class' and count occurrences
location_df = df.groupBy("Class").agg(
    count("Class").alias("Debris_Count")
)
# Infer concentration levels
location_df = location_df.withColumn(
    "Inference",
    when(col("Debris_Count") > 1000, "High Concentration")
    .when(col("Debris_Count") < 200, "Low Concentration")
    .otherwise("Moderate Concentration")
)
location_df.show(truncate=False)
```

Discussion:

- **Grouping and Counting**: The dataset is grouped by the Class column, and the number of occurrences for each class is counted.
- **Concentration Inference**: Based on the count, concentration levels are inferred as "High", "Moderate", or "Low".
- **Result Display**: The results are displayed, showing the class, debris count, and inferred concentration level.

Interpretation:

- Analyzing debris concentration helps identify areas with high or low debris accumulation. This information is valuable for targeted resource allocation and optimizing waste management strategies.

3.4 Container Type-Based Weight Distribution

Objective: Analyze the average weight of debris managed by each container type.

Key Steps: Calculate the average weight for each container type and sort the results in descending order.

```
from pyspark.sql.functions import avg
container_weight_df = df.groupBy("Container Type").agg(
    avg("FL_B").alias("Avg_Weight_Per_Container")
).orderBy("Avg_Weight_Per_Container", ascending=False)
container_weight_df.show(truncate=False)
```

Discussion:

- **Average Weight Calculation**: The average weight of debris for each container type is calculated.
- **Sorting**: The results are sorted in descending order of average weight.
- **Result Display**: The results are displayed, showing the container type and average weight per container.

Interpretation:

- Understanding the weight distribution across container types helps identify the most burdened containers and those that may be underutilized. This insight aids in optimizing container usage and improving efficiency.

3.5 Container Type and Debris Class Distribution

Objective: Understand the distribution of debris classes across various container types.

Key Steps: Group the data by container type and class, count the occurrences, and sort the results.

```
container_class_dist_df = df.groupBy("Container Type",
"Class").count().orderBy("count", ascending=False)
```

```
container_class_dist_df.show(truncate=False)
```

Discussion:

- **Grouping and Counting**: The dataset is grouped by both Container Type and Class, and the number of occurrences for each combination is counted.
- **Sorting**: The results are sorted in descending order of count.
- **Result Display**: The results are displayed, showing the container type, class, and count.

Interpretation:

- Analyzing the distribution of debris classes across container types helps in strategizing targeted collection and improving efficiency. It also detects imbalances between container types and their intended usage.

3.6 Container Type-Based Anomaly Detection

Objective: Identify unusual weight anomalies across container types.

Key Steps: Calculate the mean and standard deviation of weights for each container type, detect outliers based on these statistics, and add an inference column to explain the anomaly status.

```
from pyspark.sql.functions import mean, stddev
# Calculate mean and standard deviation by 'Container Type'
container_anomaly_df = df.groupBy("Container Type").agg(
    mean("FL_B").alias("Mean_Weight"),
    stddev("FL_B").alias("StdDev_Weight")
)
# Join to integrate calculated values
df = df.join(container_anomaly_df, "Container Type")
# Detect anomalies for each container type
df = df.withColumn(
    "Anomaly",
    when((col("FL_B") > col("Mean_Weight") + 2 * col("StdDev_Weight")) |
         (col("FL_B") < col("Mean_Weight") - 2 * col("StdDev_Weight")),
"Outlier")
    .otherwise("Normal")
)
# Add inference
df = df.withColumn(
    "Inference",
```

```
    lit("Outlier = Unusual weight detected | Normal = Expected weight
range")
)
# Display results
df.select("Container Type", "FL_B", "Mean_Weight", "StdDev_Weight",
"Anomaly", "Inference").show(truncate=False)
```

Discussion:

- **Statistical Calculation by Container Type**: The mean and standard deviation of FL_B are calculated for each container type.
- **Joining Data**: The calculated values are joined back to the original DataFrame.
- **Anomaly Detection**: Anomalies are detected for each container type based on the mean ± 2 standard deviations.
- **Inference Addition**: An inference column is added to provide a textual explanation of the anomaly status.
- **Result Display**: The results are displayed, showing the container type, weight, mean weight, standard deviation, anomaly status, and inference.

Interpretation:

- Identifying anomalies across container types helps detect potential data issues, improper categorization, or extraordinary debris types. This supports data validation and ensures accurate monitoring and forecasting.

4. RESULTS

4.1 Weight-Based Anomaly Detection with Inference

Class	FL_B	Anomaly	Inference
Emptying	85.3	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Emptying	96.2	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Non Emptying	50.9	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Emptying	83.1	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Non Emptying	73.5	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Non Emptying	53.4	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Non Emptying	58.4	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Emptying	90.7	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Emptying	83.9	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Emptying	72.1	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Non Emptying	30.6	Outlier	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Emptying	85.8	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Non Emptying	69.3	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Emptying	93.6	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Non Emptying	66.0	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Non Emptying	39.7	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Emptying	83.3	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Non Emptying	62.2	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Non Emptying	32.7	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range
Non Emptying	67.6	Normal	Outlier = Unusual debris weight detected | Normal = Weight within expected range

only showing top 20 rows

Figure 1: Weight-Based Anomaly Detection with Inference

Output Explanation: The table shows the results of the weight-based anomaly detection analysis. It includes the following columns:

- **Class**: Indicates the type of debris management activity (e.g., "Emptying" or "Non Emptying").
- **IFL_B**: Represents the weight of the debris.
- **Anomaly**: Indicates whether the weight is an "Outlier" or "Normal" based on the calculated mean and standard deviation.
- **Inference**: Provides a textual explanation of the anomaly status.

Inference:

- **Normal**: The weight is within the expected range, falling within ±2 standard deviations from the mean. This indicates that the debris weight is typical and does not show any unusual patterns.
- **Outlier**: The weight deviates significantly from the norm, falling outside the range of ±2 standard deviations from the mean. This suggests that the

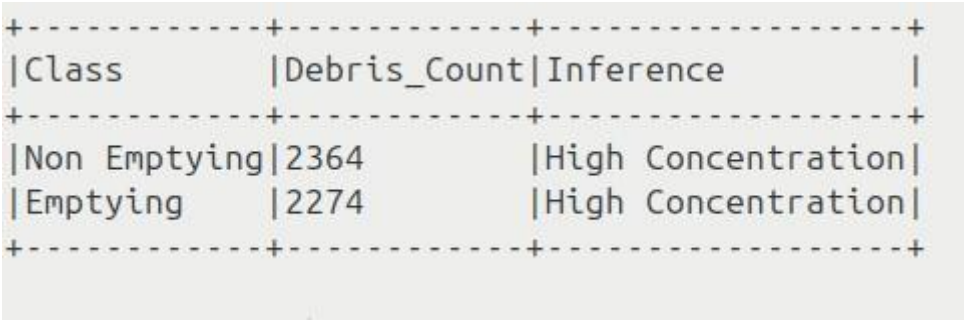
debris weight is unusual and may require further investigation to determine the cause.

Conclusion: The weight-based anomaly detection analysis successfully identifies instances of unusually high or low debris weights. By categorizing these weights as "Outliers" or "Normal," the framework enhances the understanding of irregularities in debris data. This information is valuable for:

- Detecting potential measurement errors or unexpected debris types.
- Supporting accurate monitoring and forecasting of debris management activities.
- Enabling urban authorities to make informed, data-driven decisions to optimize debris collection and resource allocation.

Overall, this analysis helps improve the efficiency and effectiveness of urban waste management by providing insights into debris weight patterns and anomalies.

4.2 Location-Based Debris Concentration Analysis



Class	Debris_Count	Inference
Non Emptying	2364	High Concentration
Emptying	2274	High Concentration

Figure 2: Location-Based Debris Concentration Analysis

Output Explanation: The table shows the results of the location-based debris concentration analysis. It includes the following columns:

- **Class**: Indicates the type of debris management activity (e.g., "Emptying" or "Non Emptying").
- **Debris_Count**: Represents the number of occurrences of each class.
- **Inference**: Provides a textual explanation of the concentration level based on the debris count.

Inference:

- **High Concentration**: The debris count is greater than 1000, indicating a high concentration of debris for that class. This suggests that these locations have a significant amount of debris accumulation.
- **Moderate Concentration**: The debris count falls between 200 and 1000, indicating a moderate concentration of debris.
- **Low Concentration**: The debris count is less than 200, indicating a low concentration of debris for that class. This suggests that these locations have relatively less debris accumulation.

Conclusion: The location-based debris concentration analysis successfully categorizes the concentration levels of debris across different classes. By identifying high, moderate, and low concentration zones, the framework provides valuable insights for:

- Targeted resource allocation: High concentration areas may require more frequent debris collection and additional resources.
- Identifying underutilized zones: Low concentration areas may indicate potential underutilization or insufficient data, which can be further investigated.
- Optimizing waste management strategies: Understanding the distribution of debris helps urban authorities plan and implement more effective waste management practices.

Overall, this analysis aids in improving the efficiency and effectiveness of urban waste management by providing a clear understanding of debris concentration patterns across different locations.

4.3 Container Type-Based Weight Distribution

Container Type	Avg_Weight_Per_Container
Diamond	70.61823899371068
Fiberglass Igloo-a	70.15338983050849
Accordion	69.84427710843372
Fiberglass Igloo-b	69.74171974522287
Rectangular Farmland	69.36906250000001
Lindstrom-b	68.8469453376206
Lindstrom-a	68.74690026954183
Lindstrom City	68.56988950276249
Rectangular	68.49487951807231
Malmobehallare	68.48150470219437
Silvertop-b	67.77012195121944
Silvertop-a	67.28233438485815
BIM-3 FTI bottentomd	66.70728476821198
Cubic	66.06039325842693

Figure 3: Container Type-Based Weight Distribution

Output Explanation: The table shows the results of the container type-based weight distribution analysis. It includes the following columns:

- **Container Type**: Indicates the type of container used for debris management.
- **Avg_Weight_Per_Container**: Represents the average weight of debris managed by each container type.

Inference:

- The table lists various container types along with their corresponding average debris weights.
- The container types are sorted in descending order based on the average weight per container.

Conclusion: The container type-based weight distribution analysis provides valuable insights into the average weight of debris managed by different container types. Key takeaways include:

- **Most Burdened Containers**: Containers like "Diamond" and "Fiberglass Igloo-a" have the highest average weights, indicating that they are the most burdened with debris. This information can help in planning maintenance and replacement schedules for these containers.
- **Underutilized Containers**: Containers with lower average weights, such as "IBM-3 FTI bottendomd" and "Cubic," may be underutilized. This insight can be used to redistribute debris collection efforts to optimize container usage.
- **Resource Allocation**: Understanding the weight distribution across container types helps in allocating resources more effectively, ensuring that heavily burdened containers receive the necessary attention and maintenance.

Overall, this analysis aids in improving the efficiency and effectiveness of urban waste management by providing a clear understanding of how different container types are utilized in managing debris.

4.4 Container Type and Debris Class Distribution

Output Explanation: The table shows the results of the container type and debris class distribution analysis. It includes the following columns:

- **Container Type**: Indicates the type of container used for debris management.
- **Class**: Represents the type of debris management activity (e.g., "Emptying" or "Non Emptying").
- **Count**: The number of occurrences for each combination of container type and class.

Container Type	Class	count
Lindstrom City	Non Emptying	190
Fiberglass Igloo-a	Emptying	187
Lindstrom-a	Emptying	186
Lindstrom-a	Non Emptying	185
Cubic	Emptying	181
Cubic	Non Emptying	176
Silvertop-a	Non Emptying	175
Accordion	Non Emptying	173
BIM-3 FTI bottentomd	Non Emptying	172
Lindstrom City	Emptying	172
Rectangular	Non Emptying	171
Malmobehallare	Emptying	168
Fiberglass Igloo-a	Non Emptying	167
Fiberglass Igloo-b	Emptying	166
Diamond	Non Emptying	165
Silvertop-b	Emptying	164
Silvertop-b	Non Emptying	164
Lindstrom-b	Non Emptying	163
Rectangular Farmland	Non Emptying	163
Rectangular	Emptying	161

only showing top 20 rows

Figure 4: Container Type and Debris Class Distribution

Inference:

- The table lists various container types along with their corresponding counts for each class.
- The data is sorted in descending order based on the count, showing the most frequent combinations at the top.

Conclusion: The container type and debris class distribution analysis provides valuable insights into how different container types are utilized for various debris management activities. Key takeaways include:

- **Efficient Categorization**: Understanding the distribution of debris classes across container types helps in strategizing targeted collection and

improving efficiency. For example, containers like "Lindstrom City" and "Cubic" have high counts for both "Emptying" and "Non Emptying" classes, indicating their frequent use.

- **Imbalances Detection**: The analysis helps detect imbalances between container types and their intended usage. For instance, if certain container types are predominantly used for one class over another, it may indicate a need for reallocation or adjustment in usage patterns.
- **Resource Optimization**: By identifying the most and least frequently used container types for different classes, urban authorities can optimize resource allocation and improve overall waste management practices.

Overall, this analysis aids in enhancing the efficiency and effectiveness of urban waste management by providing a clear understanding of the distribution of debris classes across various container types.

4.5 Container Type-Based Anomaly Detection

```
+--------------+----+-----------------+-----------------+-------+--------------------------------------------------------------------+
|Container Type|FL_B|Mean_Weight      |StdDev_Weight    |Anomaly|Inference                                                           |
+--------------+----+-----------------+-----------------+-------+--------------------------------------------------------------------+
|Lindstrom-b   |66.0|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |89.0|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |62.6|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |37.7|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |80.6|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |34.6|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |55.5|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |34.6|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |55.5|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |34.6|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |55.5|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |75.0|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |79.9|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |70.2|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |37.1|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |75.5|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |75.7|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |64.2|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |53.2|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
|Lindstrom-b   |75.1|68.8469453376206|17.691442673358907|Normal |Outlier = Unusual weight detected | Normal = Expected weight range|
+--------------+----+-----------------+-----------------+-------+--------------------------------------------------------------------+
only showing top 20 rows
```

Figure 5: Container Type-Based Anomaly Detection

Output Explanation: The table shows the results of the container type-based anomaly detection analysis. It includes the following columns:

- **Container Type**: Indicates the type of container used for debris management.

- **FID**: Represents the unique identifier for each record.
- **Mean_Weight**: The average weight of debris for the container type.
- **StdDev_Weight**: The standard deviation of the debris weight for the container type.
- **Anomaly**: Indicates whether the weight is an "Outlier" or "Normal" based on the calculated mean and standard deviation.
- **Inference**: Provides a textual explanation of the anomaly status.

Inference:

- **Normal**: The weight is within the expected range, falling within ±2 standard deviations from the mean. This indicates that the debris weight is typical and does not show any unusual patterns.
- **Outlier**: The weight deviates significantly from the norm, falling outside the range of ±2 standard deviations from the mean. This suggests that the debris weight is unusual and may require further investigation to determine the cause.

Conclusion: The container type-based anomaly detection analysis successfully identifies instances of unusual debris weights across different container types. By categorizing these weights as "Outliers" or "Normal," the framework enhances the understanding of irregularities in debris data. This information is valuable for:

- Detecting potential data issues, improper categorization, or extraordinary debris types.
- Supporting accurate monitoring and forecasting of debris management activities.
- Enabling urban authorities to make informed, data-driven decisions to optimize debris collection and resource allocation.

Overall, this analysis helps improve the efficiency and effectiveness of urban waste management by providing insights into debris weight patterns and anomalies across various container types.

Summary of Executions

The Sustainable Debris Management Framework project leverages advanced big data technologies to optimize urban waste management. Here's a summary of the key executions and their outcomes:

1. **Data Loading and Verification**:
    - **Objective**: Load the debris dataset into a Spark DataFrame and verify its structure.
    - **Execution**: Initialized a Spark session, loaded the dataset from HDFS, and displayed the first few rows and schema.
    - **Outcome**: Successfully loaded and verified the dataset, ensuring it is correctly formatted and ready for analysis.
2. **Weight-Based Anomaly Detection with Inference**:
    - **Objective**: Identify anomalous weights that deviate significantly from the norm.
    - **Execution**: Calculated the mean and standard deviation of debris weights, detected outliers, and added an inference column.
    - **Outcome**: Detected instances of unusually high or low weights, categorized as 'Outliers,' enhancing the understanding of irregularities in debris data.
3. **Location-Based Debris Concentration Analysis**:
    - **Objective**: Evaluate the distribution of debris across different classes to categorize high, moderate, and low concentrations.
    - **Execution**: Grouped the data by class, counted the occurrences, and inferred concentration levels based on predefined thresholds.
    - **Outcome**: Identified locations with high concentration, aiding in targeted resource allocation, and detected low-concentration zones, signaling potential underutilization or insufficient data.

4. **Container Type-Based Weight Distribution**:
    - **Objective**: Analyze the average weight of debris managed by each container type.
    - **Execution**: Calculated the average weight for each container type and sorted the results in descending order.
    - **Outcome**: Provided insights into the most burdened container types and identified types with lower average weights, possibly indicating underutilization.
5. **Container Type and Debris Class Distribution**:
    - **Objective**: Understand the distribution of debris classes across various container types.
    - **Execution**: Grouped the data by container type and class, counted the occurrences, and sorted the results.
    - **Outcome**: Efficiently categorized debris for targeted collection strategies and detected imbalances between container types and their intended usage.
6. **Container Type-Based Anomaly Detection**:
    - **Objective**: Identify unusual weight anomalies across container types.
    - **Execution**: Calculated the mean and standard deviation of weights for each container type, detected outliers, and added an inference column.
    - **Outcome**: Identified potential data issues, improper categorization, or extraordinary debris types, supporting data validation for accurate monitoring and forecasting.

Conclusion: The project successfully implemented a comprehensive framework for sustainable debris management using big data technologies. The analyses provided valuable insights into debris weight patterns, concentration levels, and container utilization. These insights enable urban authorities to make informed, data-driven

decisions to optimize debris collection, resource allocation, and overall waste management practices, contributing to cleaner and more sustainable cities.

REFERENCES

[1] **Apache Hadoop Documentation**:

- Comprehensive guide and reference for Apache Hadoop, covering installation, configuration, and usage of Hadoop components.
- Apache Hadoop Documentation

[2] **Apache Spark Documentation**:

- Detailed documentation for Apache Spark, including API references, programming guides, and tutorials.
- Apache Spark Documentation

[3] **Cloudera Documentation**:

- Resources and guides provided by Cloudera for managing and utilizing big data platforms, including Hadoop and Spark.
- Cloudera Documentation

[4] **PySpark API Documentation**:

- Official API documentation for PySpark, the Python API for Apache Spark, providing detailed information on functions and modules.
- PySpark API Documentation

[5] **Kaggle**:

- A renowned open data platform offering a wide range of datasets, machine learning competitions, and community support.
- Kaggle