

## **Group 5: Deadly & Damaging Storm Events**



**American  
Red Cross**

Ankur Bhattacharjee, Bimala Joshi, Cezane Karki, Isabella Pham, Kiran S. Kuchekar, Shivangi

Vipulbhai Borad

University of North Texas

ADTA 5240: Harvesting, Storing and Retrieving Data

Dr. Schenita Floyd

April 29, 2025

## Contents

<b>Executive Summary .....</b>	<b>2</b>
<b>Introduction.....</b>	<b>3</b>
<b>Work Process Overview.....</b>	<b>3</b>
Generation.....	3
Collection.....	3
Processing .....	4
<i>OpenRefine.....</i>	4
Storage .....	6
<i>GCP Project.....</i>	6
<i>GCP Storage Bucket .....</i>	7
Management.....	7
Analysis.....	8
<i>GCP Dataproc .....</i>	8
<i>BigQuery: Streaming Data .....</i>	9
<i>BigQuery: Static Data.....</i>	11
<i>Apache Hive: Streaming Data .....</i>	15
<i>Apache Spark: Streaming Data .....</i>	20
<i>Apache Hive: Static Data.....</i>	23
<i>Apache Spark: Static Data.....</i>	27
Hive & Spark Comparisons .....	30
<i>Static Dataset.....</i>	30
<i>Streaming Dataset.....</i>	30
<i>Comparison Conclusion.....</i>	31
Machine Learning (BONUS).....	32
<i>Damage Forecast For the Year 2025 .....</i>	32
Visualizations (BONUS) .....	33
<i>Static Dataset.....</i>	33
<i>Streaming Dataset.....</i>	34
Interpretation (BONUS).....	35
<b>Conclusion .....</b>	<b>36</b>
Discussion.....	36
Recommendations.....	37
1. <i>Forecasting Storm Effects Using Predictive Modeling .....</i>	37
2. <i>Geospatial Risk Mapping for Resource Deployment.....</i>	37
3. <i>Real-Time, Automated Alert Systems.....</i>	38
4. <i>Optimization of Public Messaging.....</i>	38
<b>Supporting Documentation.....</b>	<b>39</b>
Data Sheet.....	39
Meeting Notes.....	39
<i>Meeting 1 .....</i>	39
<i>Meeting 2 .....</i>	41
<i>Meeting 3 .....</i>	42
<i>Meeting 4 .....</i>	43
<i>Meeting 5 .....</i>	45
Access & Permission Policy .....	46
Jira Dashboard (BONUS) .....	46
<b>References.....</b>	<b>47</b>

## Executive Summary

As a team of data engineers and data analysts hired by the American Red Cross, a nonprofit organization that helps the ones in need and focus to prepare well against the disaster, we work to improve the preparedness for the disaster created by storms. Our project aims to collect, process, analyze and interpret the required data for the further preparedness of this disaster.

For doing the analysis, we first collected two sets of data: static and streaming data. The static data was collected from National Oceanic and Atmospheric Administration's (NOAA) from 1950 till 2024. The streaming data was obtained from National Weather Service (NWS) API and was collected on 20<sup>th</sup> April, 2025 at various times. The data was stored on Google Cloud Platform (GCP) through buckets and Dataproc. The data was also cleaned using OpenRefine and was thus prepared for analysis.

The analysis was done through various tools like BigQuery, Apache Hive and Apache Spark. Spark was found to have faster processing time compared to Hive. The critical trends of storm, severity, impact on geographic locations and seasonality were affected by severe weather. It was also found through streaming data that there were inconsistencies in urgency and emergency alerts. Our analysis found out that a more predictive model, targeted resource management, and effective alert systems are required for effective management of storm disaster response and relief.

We recommend processing the mechanism of the predictive models, geographic risk mapping for storm and alert system management for the better preparedness of such disasters. This will help in quick mobilization of resources and effective implementation of the process. We recommend that data scientists work to make a predictive model for forecasting storm events, risk mapping, and better communication and alerts to the public.

## Introduction

The American Red Cross is a network of volunteers and employees who are committed to providing help to the ones who need it. While providing help to those in need, they also prepare the community for impending disasters (American Red Cross, 2025). Many states regularly suffer from storm events such as tornadoes, hurricanes, and winter storms. Through the use of static and streaming (live) data, our team of data engineers were hired to construct a data pipeline and work alongside data scientists and analysts to provide actionable insight into how the Red Cross can better prepare for such incidents.

By framing our work process through the various stages of the data life cycle, we were able to work more efficiently and find workflow solutions that can be implemented later down the line. We believe that better preparedness and management of post-disaster scenarios could be achieved through this research. To that end, we are trying to help the American Red Cross with this project.

## Work Process Overview

### **Generation**

The data was generated from the storm incidents that occurred in different areas and at different times. Two data sets were extracted: one static and one streaming. The static dataset contains the data on storms from 1950 until 2024. The streaming dataset was collected on 20th of April, 2025 at different times, and the latest data was used in our analysis. Both sets of generated data were collected and stored in a GCP bucket for further analysis.

### **Collection**

Streaming data was sourced from the [National Weather Service \(NWS\) API](#). It was transferred to Google Cloud Platform (GCP) using python and transferred to a GCP bucket at different times.

Static data was obtained from the National Oceanic and Atmospheric Administration's (NOAA) [Storm Events Database](#). The database contains storm event data from different years, and mentions the type of the event like thunderstorm wind, Tornado, winter storm, Hurricane or Typhoon. The beginning and the end time of each storm type is collected for the analysis. Similarly, it also contains the data of the beginning and ending of the storm.

Weather report streaming data contains the timely, up-to-date information, and includes observations such as the onset and end time of each recorded storm, as well as the severity of each storm. Response and urgency data were also collected as part of this dataset.

## Processing

The data was processed (i.e. wrangled and cleaned) using OpenRefine. Null and unnecessary columns were removed using this application. Following the maxim “garbage in, garbage out”, we removed columns that were deemed unnecessary or contained too many null values. Further techniques such as text transformation and splitting individual cells which contained multiple values into distinct columns were also employed. After cleaning, each dataset was stored for future analysis.

## OpenRefine

The screenshot shows the OpenRefine interface with a dataset titled "Group\_5\_Static\_Data". The sidebar on the left lists 13 numbered steps for data cleaning:

- Value(replace(, null, 1000000, value toNumber()))
- Remove 2 rows
- Create new column Event\_filtered based on column EVENT\_TYPE by filling 90,663 rows with gref("Hurricane (Typhoon)", "Hurricane", "Tornado", "Thunderstorm Wind", "Winter Storm") contains(value), "Keep", "Remove")
- Remove column Event\_filtered
- Create new column Event\_filtered based on column EVENT\_TYPE by filling 2,935,109 rows with gref("Hurricane (Typhoon)", "Hurricane", "Tornado", "Thunderstorm Wind", "Winter Storm") return "Keep" if value in storm\_types else "Remove"
- Remove 1904935 rows

The main table view displays 1,030,174 rows with columns including EVENT\_ID, STATE, YEAR, MONTH\_NAME, EVENT\_TYPE, CZ\_TYPE, CZ\_FIPS, CZ\_NAME, WFO, BEGIN\_DATE\_TIME, and CZ\_ID. The table is paginated at the top, showing page 1 of 1000.

Figure 1.1. Static dataset loaded into OpenRefine. Screenshot taken by Ankur Bhattacharjee.

**OpenRefine streaming\_data\_storm Permalink**

Facet / Filter Undo / Redo 35 / 35 Extract... Apply...

1,875 rows Show as: rows records Show: 5 10 25 50 100 500 1000 rows < first < previous 1 - 10 next > last »

Extensions | Wikibase

All	areaDesc	sent	effective	onset	expires	ends	status	messageType	category	\$
1.	Montgomery	2025-04-20T23:12:20+00:00	2025-04-20T23:12:20+00:00		2025-04-20T23:22:20+00:00		Test	Alert	Met	Unknown
2.	Pike, IL	2025-04-20T18:11:00-05:00	2025-04-20T18:11:00-05:00	2025-04-20T18:11:00-05:00	2025-04-20T19:00:00-05:00	2025-04-20T19:00:00-05:00	Actual	Alert	Met	Extreme
3.	Pike, MO	2025-04-20T18:11:00-05:00	2025-04-20T18:11:00-05:00	2025-04-20T18:11:00-05:00	2025-04-20T19:00:00-05:00	2025-04-20T19:00:00-05:00	Actual	Alert	Met	Extreme
4.	Ralls, MO	2025-04-20T18:11:00-05:00	2025-04-20T18:11:00-05:00	2025-04-20T18:11:00-05:00	2025-04-20T19:00:00-05:00	2025-04-20T19:00:00-05:00	Actual	Alert	Met	Extreme
35.	Fill down 0 cells in column areaDesc									

Figure 1.2. Streaming dataset loaded into OpenRefine. Screenshot taken by Ankur Bhattacharjee.

We used OpenRefine to clean and structure the raw data, organizing the static and streaming storm raw datasets into two separated projects. We applied various operations to refine the datasets, removing blank and erroneous values. While working with data, the golden rule remains: "Garbage in, garbage out".

The common OpenRefine operations used in our project are as follows:

- i. Removing null value column
- ii. Text Transformation
- iii. Split multi-valued cell in column
- iv. Fill down column
- v. Remove null value instances
- vi. Create new column based on desired event\_type

## Storage

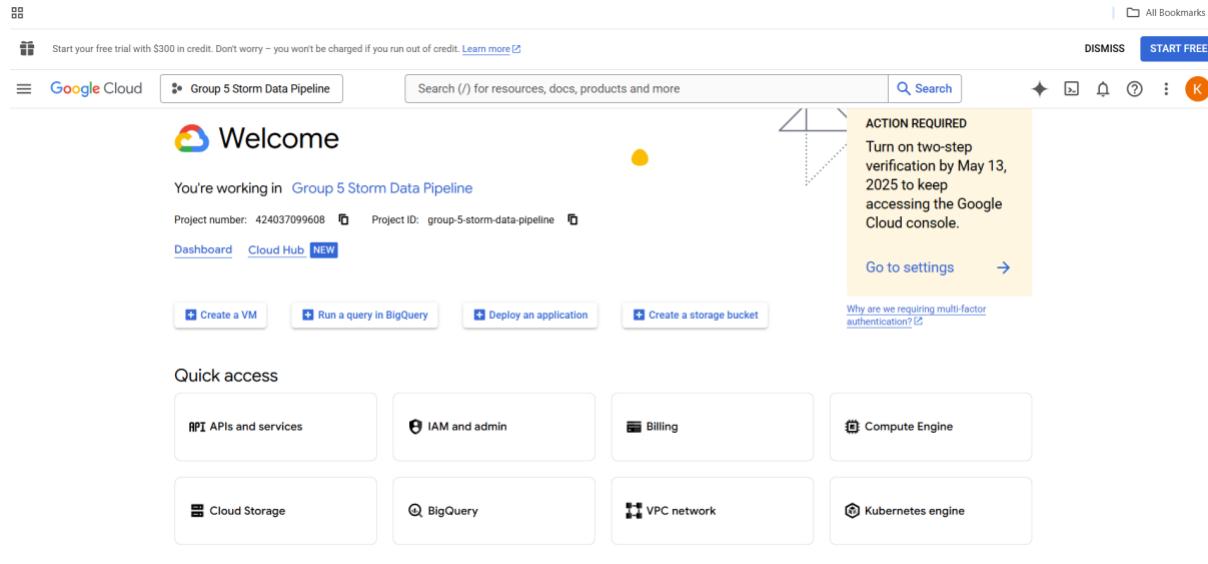


Figure 2. Screenshot taken by Kiran S. Kuchekar.

## GCP Project

GCP was used to configure and process resources for the "Group 5 Storm Data Pipeline" in this project. The GCP console allowed us to manage cloud services like Compute Engine, BigQuery, and Apache sub-modules for a wide variety of data-related tasks. Our purpose for using GCP was to oversee a cloud setup in which we could execute queries in BigQuery, deploy apps, create buckets for storage, and control VMs to process and deal with storm-related information effectively.

The flexibility and scalability of GCP made it ideal for this project, as it involved a significant amount of big data. With services such as IAM for controlling access, we were able to maintain good performance as well as data security. Using this platform, we could streamline our workflow, automate deployments, and enable collaborative, cloud-native data analysis.

## GCP Storage Bucket

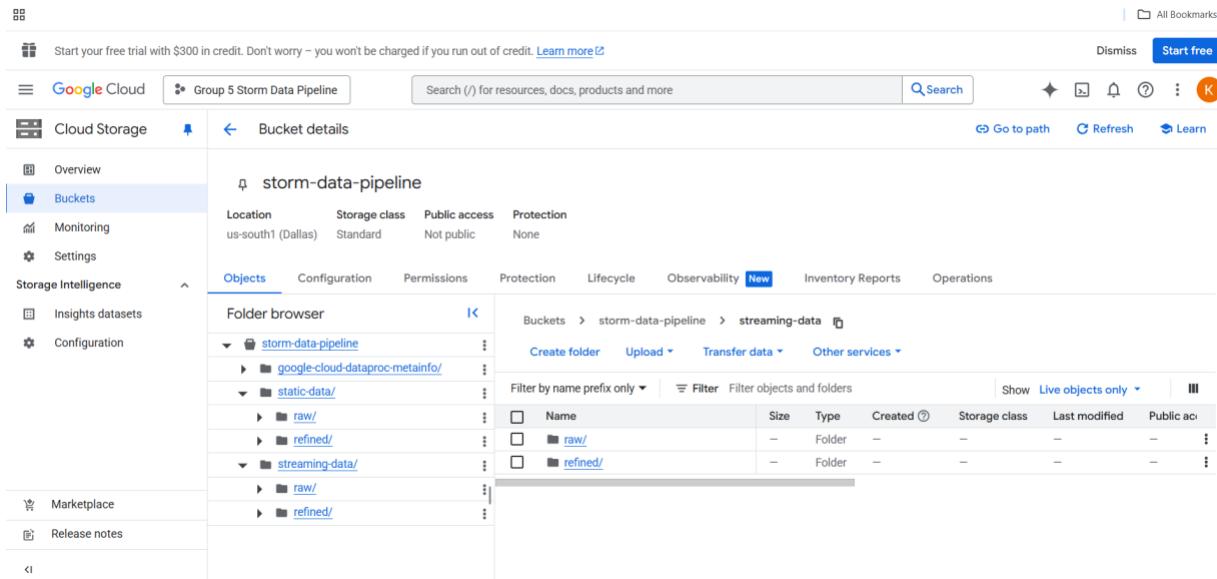


Figure 3. Screenshot taken by Kiran S. Kuchekar.

We used Google Cloud Storage for the project data pipeline. The service made it easier for us to manage and store extensive datasets in a well-structured manner in the cloud. All cleaned and processed data were uploaded to a bucket named "storm-data-pipeline". Streaming and static data were uploaded to separate folders, with subdirectories within each folder dedicated to the raw, original data and refined data, respectively. This enhanced our workflow by facilitating easy separation between processed and unprocessed data. We used Google Cloud Storage due to its scalability, seamless integration with other GCP services, and robust support for collaborative data engineering workflows.

## Management

As previously stated data was stored in a GCP bucket with the proper roles assigned to each member through IAM. Specific IAM access was provided to each member of the team for further analysis work using specialized tools - such as Apache Hive and Spark - for enhanced security. In short, the data and its analysis progress was shared with everyone through GCP.

## Analysis

### *GCP Dataproc*

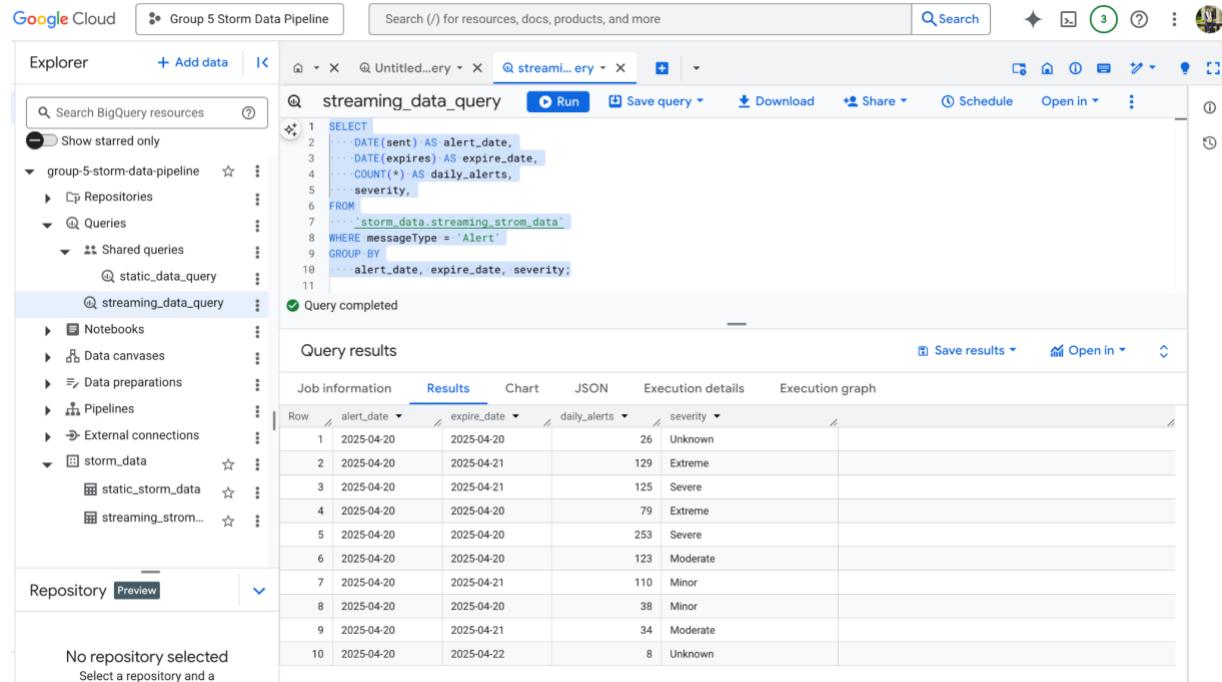
The screenshot shows the Google Cloud Compute Engine interface for managing VM instances. The left sidebar is titled 'Compute Engine' and includes sections for Overview, Virtual machines, and VM instances (which is currently selected). The main content area is titled 'VM instances' and shows a table of three instances:

Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
OK	dataproc-cluster-group-5-m	us-central1-a			10.128.0.2 (nic0)		SSH
OK	dataproc-cluster-group-5-w-0	us-central1-a			10.128.0.4 (nic0)		SSH
OK	dataproc-cluster-group-5-w-1	us-central1-a			10.128.0.3 (nic0)		SSH

Figure 4. GCP Dataproc. Screenshot taken by Bimala Joshi.

We used GCP Dataproc, which simplified the job of running workloads of big data. To process storm data, Dataproc was used to create and manage the cluster, which consisted of master and worker virtual machine (VM) instances. These VMs will be used in the future to run data processing jobs through applications such as BigQuery, Apache Spark and HiveQL. The use of Compute Engine and Dataproc together will help with parallel processing large quantities of data, providing the scalability, and other cloud-based resources. This is fast and cost-effective, as it can be started and shut down quickly to reduce costs.

## BigQuery: Streaming Data



The screenshot shows the Google Cloud BigQuery interface. The left sidebar lists projects and datasets, with 'group-5-storm-data-pipeline' selected. In the main area, a query titled 'streaming\_data\_query' is displayed:

```

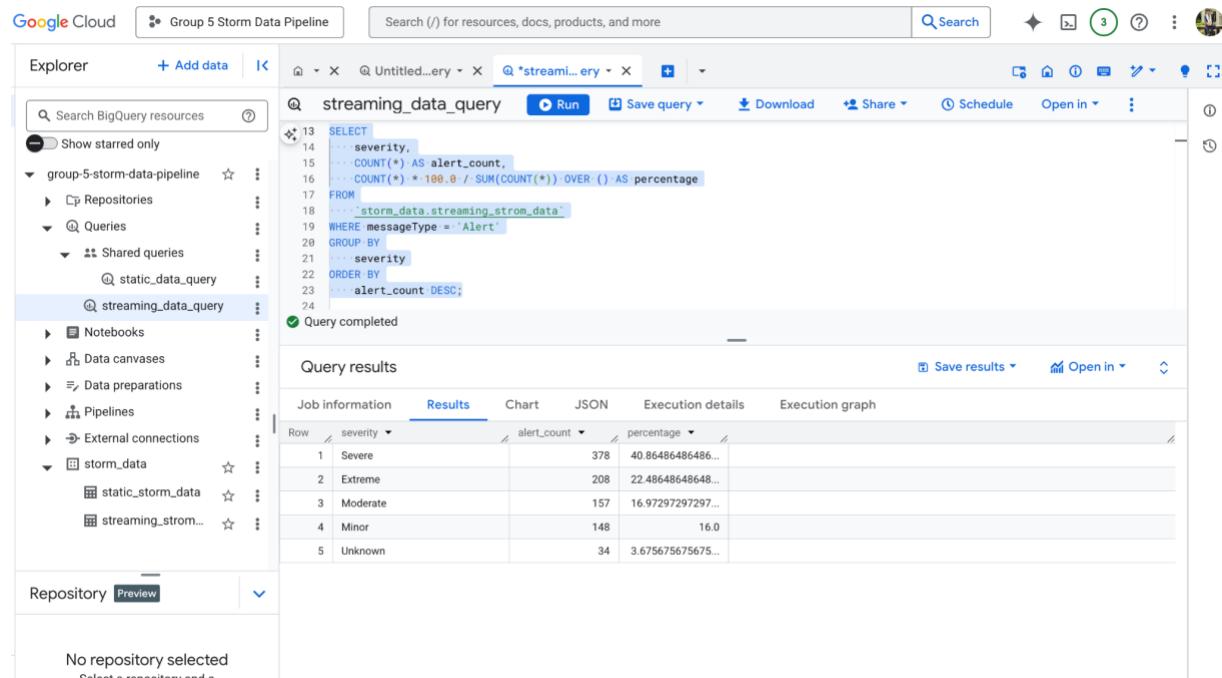
1 SELECT
2   DATE(sent) AS alert_date,
3   DATE(expires) AS expire_date,
4   COUNT(*) AS daily_alerts,
5   severity,
6   FROM
7   `storm_data.streaming_strom_data`
8   WHERE messageType = 'Alert'
9   GROUP BY
10  alert_date, expire_date, severity;
11

```

The results table shows the following data:

Row	alert_date	expire_date	daily_alerts	severity
1	2025-04-20	2025-04-20	26	Unknown
2	2025-04-20	2025-04-21	129	Extreme
3	2025-04-20	2025-04-21	125	Severe
4	2025-04-20	2025-04-20	79	Extreme
5	2025-04-20	2025-04-20	253	Severe
6	2025-04-20	2025-04-20	123	Moderate
7	2025-04-20	2025-04-21	110	Minor
8	2025-04-20	2025-04-20	38	Minor
9	2025-04-20	2025-04-21	34	Moderate
10	2025-04-20	2025-04-22	8	Unknown

Figure 5.1. Big Query to get the number of daily alerts and severity between the alert and expiry date. Screenshot taken by Cezane Karki.



The screenshot shows the Google Cloud BigQuery interface. The left sidebar lists projects and datasets, with 'group-5-storm-data-pipeline' selected. In the main area, a query titled 'streaming\_data\_query' is displayed:

```

13 SELECT
14   severity,
15   COUNT(*) AS alert_count,
16   COUNT(*) * 100.0 / SUM(COUNT(*)) OVER () AS percentage
17   FROM
18   `storm_data.streaming_strom_data`
19   WHERE messageType = 'Alert'
20   GROUP BY
21   severity
22   ORDER BY
23   alert_count DESC;
24

```

The results table shows the following data:

Row	severity	alert_count	percentage
1	Severe	378	40.8648648648...
2	Extreme	208	22.48648648648...
3	Moderate	157	16.97297297297...
4	Minor	148	16.0
5	Unknown	34	3.675675675675...

Figure 5.2. Big Query to get the severity level and its alert count along with the percentage of each severity level out of the total number of alerts. Screenshot taken by Cezane Karki.

The screenshot shows the Google Cloud BigQuery interface. The top navigation bar includes 'Google Cloud' and 'Group 5 Storm Data Pipeline'. The main area displays a query titled 'streaming\_data\_query'.

```

78 SELECT word,
79      COUNT(*) AS word_count
80   FROM `storm_data.streaming_strom_data`
81  CROSS JOIN
82    UNNEST(SPLIT(headline, ' ')) AS word
83 WHERE messageType = 'Alert'
84 AND word NOT IN ('the', 'and', 'is', 'of', 'a', 'an', 'in', 'to', 'for', 'on', 'by', 'at', 'it', 'this', 'that', 'these', 'those', 'are', 'was', 'were', 'be', 'been', 'being',
85      'have', 'has', 'had', 'do', 'does', 'did', 'can', 'will', 'would', 'should', 'could', 'from', 'with', 'its', 'as')
86 GROUP BY
87      word
88 ORDER BY
89      word_count DESC
90 LIMIT 20;
91
92

```

The results table shows the top 20 words from the headlines:

word	word_count
April	1678
20	1467
COT	1356
Issued	892
NWS	892
until	786
Warning	489
Thunderstorm	288
Severe	288
MO	236
Tornado	214
Rock	210
Little	210
AR	210
weather	114

Figure 5.3. Big Query to get the word counts used in the headlines of the data. Screenshot taken by Cezane Karki.

The screenshot shows the Google Cloud BigQuery interface. The top navigation bar includes 'Google Cloud' and 'Group 5 Storm Data Pipeline'. The main area displays a query titled 'streaming\_data\_query'.

```

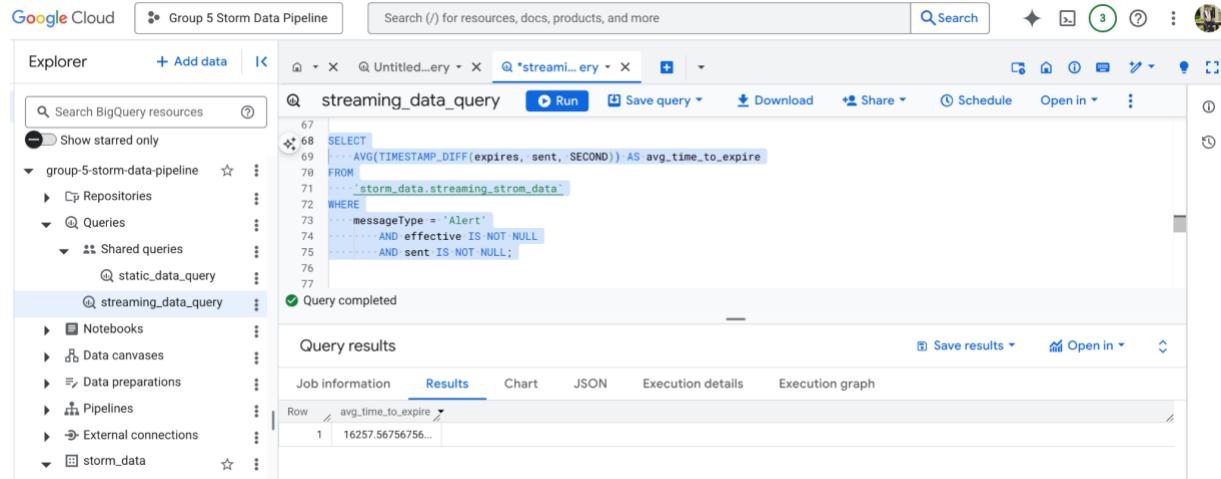
94 SELECT alert_date,
95      severity,
96      urgency,
97      response,
98      COUNT(*) AS daily_alert_count
99
100 FROM `storm_data.streaming_strom_data`
101 WHERE
102      messageType = 'Alert'
103 GROUP BY
104      alert_date,
105      severity,
106      urgency,
107      response,
108      ORDER BY
109      alert_date,
110      severity,
111      urgency,
112      response;
113

```

The results table shows the daily count of alerts categorized by date, severity, urgency, and response:

alert_date	severity	urgency	response	daily_alert_count
2025-04-20	Extreme	Future	Monitor	126
2025-04-20	Extreme	Immediate	Shelter	82
2025-04-20	Minor	Expected	Avoid	52
2025-04-20	Minor	Expected	Monitor	7
2025-04-20	Minor	Expected	Prepare	59
2025-04-20	Minor	Past	AllClear	30
2025-04-20	Moderate	Expected	Avoid	27
2025-04-20	Moderate	Expected	Execute	130
2025-04-20	Severe	Expected	Avoid	15
2025-04-20	Severe	Expected	Prepare	52
2025-04-20	Severe	Future	Prepare	15
2025-04-20	Severe	Immediate	Avoid	17

Figure 5.4. Big Query to get the number of storm alerts for each date, broken down by severity, urgency, and response. Screenshot taken by Cezane Karki.



The screenshot shows the Google Cloud BigQuery interface. The left sidebar is titled 'Explorer' and lists several datasets and queries under 'group-5-storm-data-pipeline'. The main area shows a query named 'streaming\_data\_query' with the following SQL code:

```

67
68 SELECT
69   AVG(TIMESTAMP_DIFF(expires, sent, SECOND)) AS avg_time_to_expire
70 FROM
71   `group-5-storm-data-pipeline.storm_data.streaming_strom_data`
72 WHERE
73   messageType = 'Alert'
74   AND effective IS NOT NULL
75   AND sent IS NOT NULL;
76
77

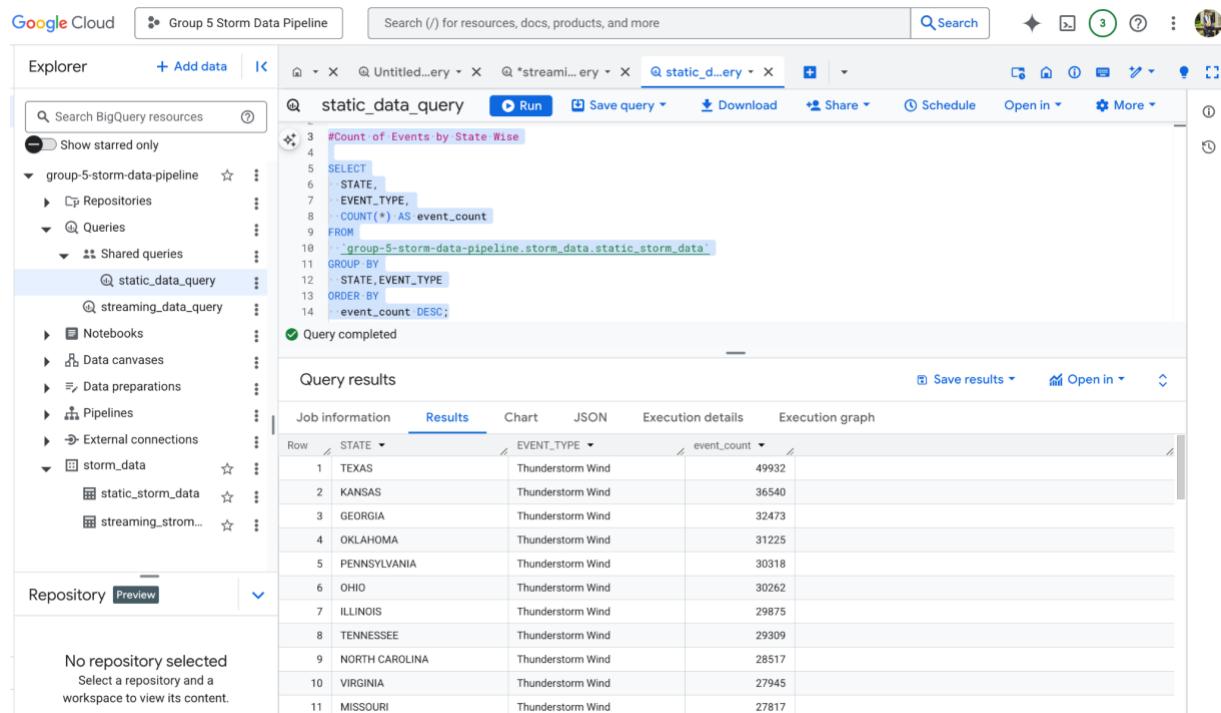
```

The status bar indicates 'Query completed'. Below the code, the 'Results' tab is selected, showing a single row of results:

Row	avg_time_to_expire
1	16257.56756756...

Figure 5.5. Big Query to get the average time for an alert to expire. Screenshot taken by Cezane Karki.

## BigQuery: Static Data



The screenshot shows the Google Cloud BigQuery interface. The left sidebar is titled 'Explorer' and lists several datasets and queries under 'group-5-storm-data-pipeline'. The main area shows a query named 'static\_data\_query' with the following SQL code:

```

3 #Count of Events by State Wise
4
5 SELECT
6   STATE,
7   EVENT_TYPE,
8   COUNT(*) AS event_count
9 FROM
10  `group-5-storm-data-pipeline.storm_data.static_storm_data`
11 GROUP BY
12  STATE, EVENT_TYPE
13 ORDER BY
14  event_count DESC;

```

The status bar indicates 'Query completed'. Below the code, the 'Results' tab is selected, showing a table of event counts by state and type:

STATE	EVENT_TYPE	event_count
TEXAS	Thunderstorm Wind	49932
KANSAS	Thunderstorm Wind	36540
GEORGIA	Thunderstorm Wind	32473
OKLAHOMA	Thunderstorm Wind	31225
PENNSYLVANIA	Thunderstorm Wind	30318
OHIO	Thunderstorm Wind	30262
ILLINOIS	Thunderstorm Wind	29875
TENNESSEE	Thunderstorm Wind	29309
NORTH CAROLINA	Thunderstorm Wind	28517
VIRGINIA	Thunderstorm Wind	27945
MISSOURI	Thunderstorm Wind	27817

Figure 6.1. Big Query to get the number of storm events for each state and event type, ordered by the event count in descending order. Screenshot taken by Cezane Karki.

The screenshot shows the Google Cloud BigQuery interface. The top navigation bar includes 'Google Cloud' and a search bar. The main area has tabs for 'Explorer' and '+ Add data'. A sidebar on the left lists project resources under 'group-5-storm-data-pipeline': 'Repositories', 'Queries' (selected), 'Shared queries' (including 'static\_data\_query' which is also selected), 'Notebooks', 'Data canvases', 'Data preparations', 'Pipelines', 'External connections', and 'storm\_data' (containing 'static\_storm\_data' and 'streaming\_storm\_data'). The central pane displays a query editor with the following SQL code:

```
17
18 #total_property_damage_by_event_type
19 SELECT
20   EVENT_TYPE,
21   SUM(CAST(DAMAGE_PROPERTY AS INT64)) AS total_property_damage
22 FROM
23   `group-5-storm-data-pipeline.storm_data.static_storm_data`
24 GROUP BY
25   EVENT_TYPE
26 ORDER BY
27   total_property_damage DESC;
28
29
```

A message 'Query completed' is shown below the code. The 'Results' tab is selected in the 'Query results' section, displaying the following data:

Row	EVENT_TYPE	total_property_da...
1	Hurricane (Typhoon)	172255768618
2	Tornado	102357868717
3	Hurricane	69351109000
4	Thunderstorm Wind	23589627962
5	Winter Storm	5445706510

Figure 6.2. Big Query to get the total damaged property grouped by the event type. Screenshot taken by Cezane Karki.

The screenshot shows the Google Cloud BigQuery interface. The left sidebar lists projects and datasets, including 'group-5-storm-data-pipeline' and its sub-tables 'static\_storm\_data' and 'streaming\_storm\_data'. The main area displays a query titled 'static\_data\_query' which retrieves total injuries by state from the 'static\_storm\_data' table. The results table shows the top 11 states with their respective total injuries.

```
#total_injuries (direct + indirect) by state
SELECT
  STATE,
  SUM(CAST(INJURIES_DIRECT AS INT64) + CAST(INJURIES_INDIRECT AS INT64)) AS total_injuries
FROM
  `group-5-storm-data-pipeline.storm_data.static_storm_data`
GROUP BY
  STATE
ORDER BY
  total_injuries DESC;
```

STATE	total_injuries
TEXAS	17089
ALABAMA	12556
MISSISSIPPI	8898
OKLAHOMA	8221
MISSOURI	7772
ARKANSAS	7405
TENNESSEE	7286
ILLINOIS	6374
GEORGIA	6341
FLORIDA	6305
OHIO	5686

Figure 6.3. Big Query to get the total number of injuries in each state. Screenshot taken by Cezane Karki.

The screenshot shows the Google Cloud BigQuery interface. The left sidebar lists repositories and queries, with 'static\_data\_query' selected. The main area displays a SQL query and its results.

```

40
41 # top 10 states with highest death tolls
42 SELECT
43   STATE,
44   SUM(CAST(DEATHS_DIRECT AS INT64) + CAST(DEATHS INDIRECT AS INT64)) AS total_deaths
45 FROM
46   `group-5-storm-data-pipeline.storm_data.static_storm_data`
47 GROUP BY
48   STATE
49 ORDER BY
50   total_deaths DESC
51 LIMIT 10;
52

```

**Query completed**

STATE	total_deaths
LOUISIANA	1154
ALABAMA	1132
TEXAS	915
MISSISSIPPI	862
MISSOURI	744
TENNESSEE	636
ARKANSAS	580
FLORIDA	564
OKLAHOMA	551
INDIANA	415

Figure 6.4. Big Query to get the total number of deaths in each state. Screenshot taken by Cezane Karki.

The screenshot shows the Google Cloud BigQuery interface. The left sidebar lists repositories and queries, with 'static\_data\_query' selected. The main area displays a SQL query and its results.

```

54 # top 100 event frequency by year and month along with casualties and damages
55 SELECT
56   YEAR,
57   MONTH_NAME,
58   COUNT(*) AS event_count,
59   SUM(CAST(INJURIES_DIRECT AS INT64) + CAST(INJURIES INDIRECT AS INT64)) AS total_injuries,
60   SUM(CAST(DEATHS_DIRECT AS INT64) + CAST(DEATHS INDIRECT AS INT64)) AS total_deaths,
61   SUM(CAST(DAMAGE_PROPERTY AS INT64)) AS total_property_damage,
62   SUM(CAST(DAMAGE_CROPS AS INT64)) AS total_crops_damage
63 FROM
64   `group-5-storm-data-pipeline.storm_data.static_storm_data`
65 GROUP BY
66   YEAR, MONTH_NAME
67 ORDER BY
68   event_count DESC
69 LIMIT 100;

```

**Query completed**

YEAR	MONTH_NAME	event_count	total_injuries	total_deaths	total_property_da...	total_crops.damage
2008	Jun	11769	377	56	2099460900	25697000
2012	July	10604	226	46	118776500	1985000
1998	June	10500	1000	18	757918920	79682600
2011	June	10390	586	28	644536300	87364000
2010	June	9950	506	40	790154100	3003000
2011	April	9752	7524	788	12171912698	39096000
2009	Jun	8994	140	18	171323800	5454000
2008	July	8265	132	12	688441900	15450000
2003	Julv	8164	150	22	501927000	98854600

Figure 6.5. Big Query to get the total number of storm events, injuries, deaths, property damage, and crop damage for each year and month. Screenshot taken by Cezane Karki.

The screenshot shows the Google Cloud BigQuery interface. The top navigation bar includes 'Google Cloud' and 'Group 5 Storm Data Pipeline'. The main area has tabs for 'Explorer', '+ Add data', and 'Search (/) for resources, docs, products, and more'. A search bar at the top right contains 'Search' and a magnifying glass icon.

The 'Queries' tab is selected, showing a list of queries. One query, 'static\_data\_query', is highlighted and expanded. The code for this query is as follows:

```

72 WITH target_years AS (
73   SELECT
74     EXTRACT(YEAR FROM CURRENT_DATE()) - 1 AS last_year,
75     EXTRACT(YEAR FROM CURRENT_DATE()) - 10 AS ten_years_ago
76 ),
77   filtered_data AS (
78   SELECT
79     YEAR,
80     EVENT_TYPE,
81     STATE,
82     CAST(DEATHS_DIRECT AS INT64) + CAST(DEATHS_INDIRECT AS INT64) AS total_deaths,
83     CAST(INJURIES_DIRECT AS INT64) + CAST(INJURIES_INDIRECT AS INT64) AS total_injuries,
84     CAST(DAMAGE_PROPERTY AS INT64) AS property_damage,
85     CAST(DAMAGE_CROP AS INT64) AS crop_damage
86   FROM
87     `group-5-storm-data-pipeline.storm_data.static_store_data`
88   WHERE
89     YEAR IN (
90       SELECT last_year FROM target_years
91     UNION ALL
92       SELECT ten_years_ago FROM target_years
93     )
94   )
95   SELECT
96     YEAR,
97     COUNT(*) AS event_count,
98     SUM(total_deaths) AS total_deaths,
99     SUM(total_injuries) AS total_injuries,
100    SUM(property_damage) AS total_property_damage,
101    SUM(crop_damage) AS total_crop_damage
102  FROM
103    filtered_data
104  GROUP BY
105    YEAR
106  ORDER BY
107    YEAR;
108  ORDER BY
109    YEAR;
110

```

The 'Preview' tab is selected, showing the results of the query. The results table has columns: Row, YEAR, event\_count, total\_deaths, total\_injuries, total\_property\_damage, and total\_crop\_damage. The data shows two rows: one for 2015 and one for 2024.

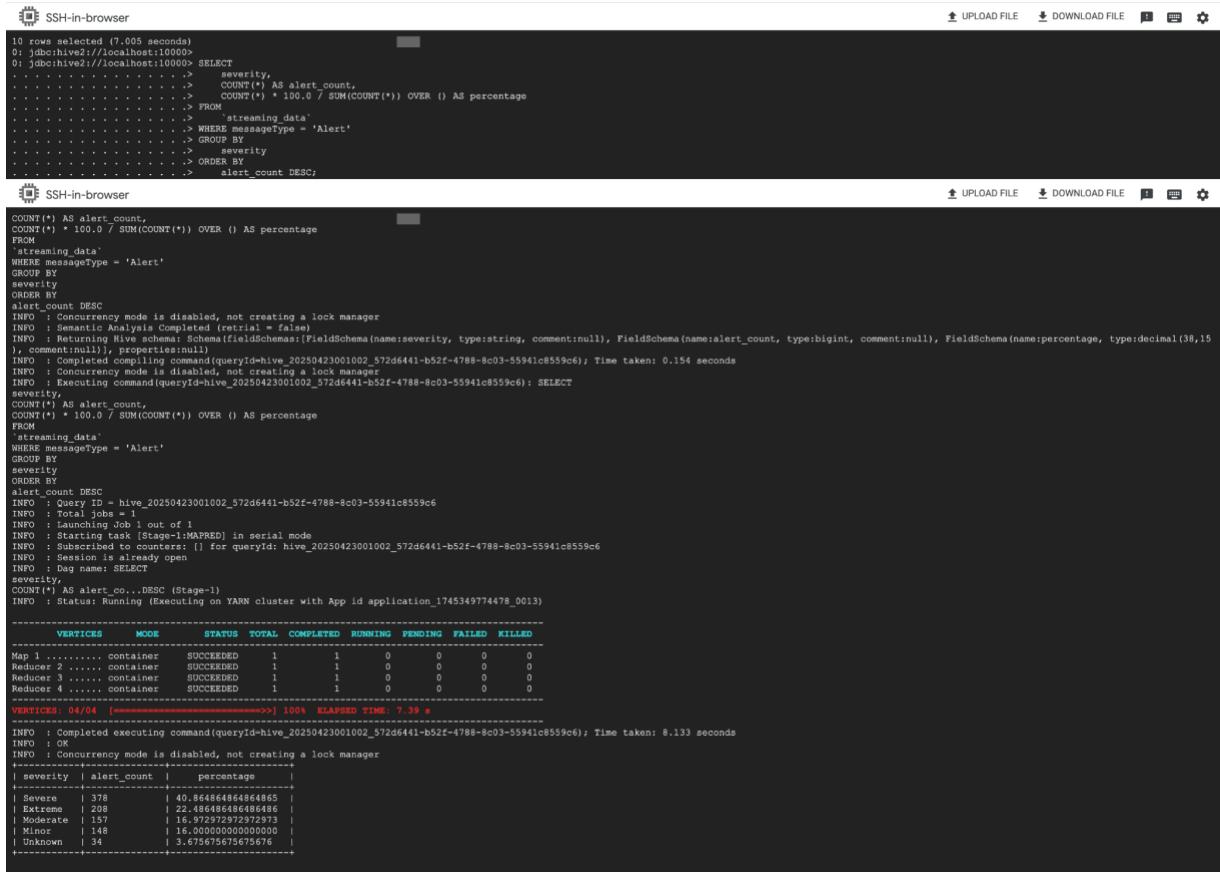
Row	YEAR	event_count	total_deaths	total_injuries	total_property_damage	total_crop_damage
1	2015	19332	104	1238	639578768	24295600
2	2024	24428	147	924	7451962769	1558027410

*Figure 6.6. Big Query to get the storm data for the last year and prior 10 years for the calculation of total number of events, deaths, injuries, property damage, and crop damage for each year (2024 and 2015). Screenshot taken by Cezane Karki.*

We used Google BigQuery to deeply analyze the storm dataset. We queried the real storm alert data in real-time to retrieve trends in frequency, severity, urgency, and types of response. SQL queries were written to aggregate the alerts over date, flag them as severe and urgent, and determine the average time from when an alert was made until when it lapsed. We also extracted the most common kinds of weather occurrences and extracted the most common words from alert titles to find dominant themes in storm communication. The high-velocity and scalable nature of BigQuery enables us to efficiently query this vast dataset, which supports the capacity to derive sound conclusions regarding public weather warning behavior and communication trends in terms of warnings.

*Apache Hive: Streaming Data*

*Figure 7.1.* HiveQL query to identify the number of cyclone alerts received on a daily basis by severity and an indication of alert frequency with the help of impacted timestamp conversion which would help in aggregation over time. Screenshots taken by Shivangi Borad.



The screenshot shows two terminal windows side-by-side, both titled "SSH-in-browser".

**Top Terminal:**

```
10 rows selected (7.005 seconds)
0: jdbc:hive2://localhost:10000> SELECT
...     severity,
...     COUNT(*) AS alert_count,
...     COUNT(*) * 100.0 / SUM(COUNT(*)) OVER () AS percentage
...   FROM streaming_data
...  WHERE messageType = 'Alert'
... GROUP BY
...   severity
... ORDER BY
...   alert_count DESC
```

**Bottom Terminal:**

```
COUNT(*) AS alert_count,
COUNT(*) * 100.0 / SUM(COUNT(*)) OVER () AS percentage
FROM streaming_data
WHERE messageType = 'Alert'
GROUP BY
severity
ORDER BY
alert_count DESC
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Semantic Analysis Completed (retry = false)
INFO : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:severity, type:string, comment:null), FieldSchema(name:alert_count, type:bigint, comment:null), FieldSchema(name:percentage, type:decimal(38,15), comment:null)], properties:null)
INFO : Compiled command (queryId=hive_20250423001002_572d6441-b52f-4788-8c03-55941c8559e6); Time taken: 0.154 seconds
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Executing command(queryId=hive_20250423001002_572d6441-b52f-4788-8c03-55941c8559e6): SELECT
severity,
COUNT(*) AS alert_count,
COUNT(*) * 100.0 / SUM(COUNT(*)) OVER () AS percentage
FROM streaming_data
WHERE messageType = 'Alert'
GROUP BY
severity
ORDER BY
alert_count DESC
INFO : Query ID = hive_20250423001002_572d6441-b52f-4788-8c03-55941c8559e6
INFO : Number of Jobs = 1
INFO : Launching Job 1 out of 1
INFO : Starting task [Stage-1-MAPRED] in serial mode
INFO : Subscribed to counters: [] for queryId: hive_20250423001002_572d6441-b52f-4788-8c03-55941c8559e6
INFO : Session is already open
INFO : Log name: SELECT
severity,
COUNT(*) AS alert_count DESC (Stage-1)
INFO : Status: Running (Executing on YARN cluster with App id application_1745349774478_0013)

-----  

VERTICES MODE STATUS TOTAL_COMPLETED RUNNING PENDING FAILED KILLED  

Map 1 ..... container SUCCEEDED 1 1 0 0 0 0  

Reducer 2 ..... container SUCCEEDED 1 1 0 0 0 0  

Reducer 3 ..... container SUCCEEDED 1 1 0 0 0 0  

Reducer 4 ..... container SUCCEEDED 1 1 0 0 0 0  

-----  

VERTICES: 04/04 [mapred] 100% ELAPSED TIME: 7.39 s  

INFO : Completed executing command(queryId=hive_20250423001002_572d6441-b52f-4788-8c03-55941c8559e6); Time taken: 8.133 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock manager
-----  

| severity | alert_count | percentage |
-----  

| Severe | 176 | 40.864864864864865 |
| Extreme | 208 | 22.484848484864866 |
| Moderate | 157 | 16.972972972972973 |
| Minor | 148 | 16.00000000000000 |
| Unknown | 34 | 3.679675675675676 |
-----
```

Figure 7.2. HiveQL query for estimating the number and percentage of alerts at each severity scale, which includes grouping by severity for indicating the most often-observed alert types generated during a cyclone event. Screenshot taken by Shivangi Borad.

The image shows two terminal windows side-by-side, both titled "SSH-in-browser".

**Terminal 1 (Left):**

```

5 rows selected (8.32 seconds)
0: jdbc:hive2://localhost:10000> SELECT
...     word,
...     COUNT(*) AS word_count
...   FROM
...     streaming_data
...   CROSS JOIN
...     UNNEST(SPLIT(headline, ' ')) AS word
...   WHERE messageType = 'alert'
...   AND word NOT IN ('the', 'and', 'is', 'of', 'a', 'an', 'in', 'to', 'for', 'on', 'by', 'at', 'it', 'this', 'that', 'these', 'those', 'are', 'was', 'were', 'be', 'been', 'being', 'ha
ver', 'has', 'had', 'do', 'does', 'did', 'can', 'will', 'would', 'should', 'could', 'from', 'with', 'its', 'as')
...   GROUP BY
...     word
...   ORDER BY
...     word_count DESC
...   LIMIT 20;
Error: Error while compiling statement: FAILED: ParseException line 7:7 cannot recognize input near 'UNNEST' '(' 'SPLIT' in joinSourcePart (state=42000,code=40000)
0: jdbc:hive2://localhost:10000> SELECT
...     word,
...     COUNT(*) AS word_count
...   FROM
...     streaming_data
...   LATERAL VIEW explode(split(headline, ' ')) exploded_table AS word
...   WHERE
...     messageType = 'Alert'
...     AND lower(word) NOT IN ('the', 'and', 'is', 'of', 'a', 'an', 'in', 'to', 'for', 'on', 'by', 'at', 'it', 'this', 'that', 'these', 'those', 'are', 'was', 'were', 'be', 'been', 'being', 'ha
ve', 'has', 'had', 'do', 'does', 'did', 'can', 'will', 'would', 'should', 'could', 'from', 'with', 'its', 'as')
...   GROUP BY
...     word
...   ORDER BY
...     word_count DESC
...   LIMIT 20;

```

**Terminal 2 (Right):**

```

WHERE
messageType = 'Alert'
AND lower(word) NOT IN ('the', 'and', 'is', 'of', 'a', 'an', 'in', 'to', 'for', 'on', 'by', 'at', 'it', 'this', 'that', 'these', 'those', 'are', 'was', 'were', 'be', 'been', 'being', 'ha
ve', 'has', 'had', 'do', 'does', 'did', 'can', 'will', 'would', 'should', 'could', 'from', 'with', 'its', 'as')
GROUP BY
word
ORDER BY
word_count DESC
LIMIT 20
INFO : Query ID = hive_20250423001124_65b64940-78d6-4855-81f8-df23fb40d015
INFO : Total jobs = 1
INFO : Launching Job 1 out of 1
INFO : Planning [Stage-1-MAPRED] in serial mode
INFO : Subscribed to counters: [] for queryIdr hive_20250423001124_65b64940-78d6-4855-81f8-df23fb40d015
INFO : Session is already open
INFO : Dag name: SELECT
word
COUNT(*) AS word_count
FRO..20 (Stage-1)
INFO : Status: Running (Executing on YARN cluster with App id application_1745349774478_0013)

-----  

VERTICES    MODE    STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED  

Map 1 ..... container    SUCCEEDED    1    1    0    0    0    0  

Reducer 3 ..... container    SUCCEEDED    1    1    0    0    0    0  

Reducer 3 ..... container    SUCCEEDED    1    1    0    0    0    0  

-----  

VERTICES: 03/03 [=====>>>] 100% ELAPSED TIME: 7.25 s  

INFO : Completed executing command(queryId=hive_20250423001124_65b64940-78d6-4855-81f8-df23fb40d015), Time taken: 7.971 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock manager
-----  

| word | word_count |  

| April | 1678 |  

| 20 | 1467 |  

| CDT | 1356 |  

| WWD | 132 |  

| issued | 892 |  

| until | 786 |  

| Warning | 489 |  

| severe | 288 |  

| Thunderstorm | 288 |  

| NO | 236 |  

| Tornado | 214 |  

| Little | 210 |  

| AM | 20 |  

| Rock | 210 |  

| Watch | 175 |  

| 21 | 166 |  

| Advisory | 153 |  

| EDT | 153 |  

| 11:00PM | 115 |  

| Springfield | 109 |  

-----  

20 rows selected (8.094 seconds)
0: jdbc:hive2://localhost:10000>
```

Figure 7.3. HiveQL query extracting and counting the most repeated words in alert headlines filtering out stopwords for highlighting important terms in public understanding warnings. Screenshot taken by Shivangi Borad.

```

SSH-in-browser
0: jdbc:hive2://localhost:10000> SELECT
...>     alert_date,
...>     severity,
...>     urgency,
...>     response,
...>     COUNT(*) AS daily_alert_count
...>   FROM
...>     streaming_data
...>   WHERE
...>     messageType = 'Alert'
...> GROUP BY
...>     alert_date,
...>     severity,
...>     urgency,
...>     response
...> ORDER BY
...>     alert_date,
...>     severity,
...>     urgency,
...>     response;
Error: Error while compiling statement: FAILED: SemanticException [Error 10004]: Line 12:0 Invalid table alias or column reference 'alert_date': (possible column names are: areadesc, sent, effective, onset, expires, ends, status, messagetype, category, severity, certainty, urgency, event, sender, sendername, headline, description, instruction, response) (state=42000,code=10004)
0: jdbc:hive2://localhost:10000> SELECT
...>     DATE(CAST(TRANSLATE(SUBSTR(sent, 1, 19), '?', ' ') AS TIMESTAMP)) AS alert_date,
...>     severity,
...>     urgency,
...>     response,
...>     COUNT(*) AS daily_alert_count
...>   FROM
...>     streaming_data
...>   WHERE
...>     messageType = 'Alert'
...> GROUP BY
...>     DATE(CAST(TRANSLATE(SUBSTR(sent, 1, 19), '?', ' ') AS TIMESTAMP)),
...>     severity,
...>     urgency,
...>     response
...> ORDER BY
...>     alert_date,
...>     severity,
...>     urgency,
...>     response;
Count(*) AS daily_alert_count
FROM
streaming_data
WHERE
messageType = 'Alert'
GROUP BY
DATE(CAST(TRANSLATE(SUBSTR(sent, 1, 19), '?', ' ') AS TIMESTAMP)),
severity,
urgency,
response
ORDER BY
alert_date,
severity,
urgency,
response
INFO : Query ID = hive_20250423001228_a885eb65-b948-45b8-b294-218d3c8118fa
INFO : Total jobs = 1
INFO : Launching Job 1 out of 1
INFO : Planning (Stage=1-MAPRED) in serial mode
INFO : Subscribed to counters: [] for queryId: hive_20250423001228_a885eb65-b948-45b8-b294-218d3c8118fa
INFO : Session is already open
INFO : Dag name: SELECT
INFO : Date: 2025-04-20 10:45:00 UTC
INFO : Stage: 1
INFO : Status: Running (Executing on YARN cluster with App id application_1745349774478_0013)

----- VERTICES MODE STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED -----
Map 1 ..... container SUCCEEDED 1 1 0 0 0 0
Reducer 1 ..... container SUCCEEDED 1 1 0 0 0 0
Reducer 2 ..... container SUCCEEDED 1 1 0 0 0 0
Reducer 3 ..... container SUCCEEDED 1 1 0 0 0 0
----- VERTICES: 0/3 [=====] 100% ELAPSED TIME: 6.96 s
INFO : Completed executing command(queryId=hive_20250423001228_a885eb65-b948-45b8-b294-218d3c8118fa), Time taken: 7.688 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock manager
+-----+
| alert_date | severity | urgency | response | daily_alert_count |
+-----+
| 2025-04-20 | Extreme  | Future   | Monitor   | 126
| 2025-04-20 | Extreme  | Immediate| Shelter  | 82
| 2025-04-20 | Minor    | Expected  | Avoid    | 52
| 2025-04-20 | Minor    | Immediate| Monitor  | 1
| 2025-04-20 | Minor    | Expected  | Prepare  | 59
| 2025-04-20 | Minor    | Past     | AllClear | 30
| 2025-04-20 | Moderate | Expected  | Avoid    | 27
| 2025-04-20 | Moderate | Immediate| AllClear | 100
| 2025-04-20 | Severe   | Immediate| Avoid    | 15
| 2025-04-20 | Severe   | Expected  | Prepare  | 52
| 2025-04-20 | Severe   | Future   | Prepare  | 15
| 2025-04-20 | Severe   | Immediate| Avoid    | 17
| 2025-04-20 | Severe   | Immediate| Shelter  | 279
| 2025-04-20 | Unknown  | Unknown  | Monitor  | 8
| 2025-04-20 | Unknown  | None     | None    | 26
+-----+
15 rows selected (7.688 seconds)

```

Figure 7.4. HiveQL Query: Grouping of the data by date, severity, urgency, and response type of cyclone alerts for day-by-day study of how these characteristics change with the increase or decrease in danger levels. Screenshot taken by Shivangi Borad.

```

SSH-in-browser
15 rows selected (7.868 seconds)
0: jdbc:hive2://localhost:10000>
0: jdbc:hive2://localhost:10000> SELECT
...>     AVG(TIMEESTAMP_DIFF(expires, sent, SECOND)) AS avg_time_to_expire
...>   FROM
...>     streaming_data
...>   WHERE
...>     messageType = 'Alert'
...>     AND effective IS NOT NULL
...>     AND sent IS NOT NULL
Error: Error while compiling statement: FAILED: SemanticException [Error 10004]: Line 2:34 Invalid table alias or column reference 'SECOND': (possible column names are: areadesc, sent, effective, onset, expires, ends, status, messagetype, category, severity, certainty, urgency, event, sender, sendername, headline, description, instruction, response) (state=42000,code=10004)
0: jdbc:hive2://localhost:10000> SELECT
...>     AVG(unix_timestamp(expires) - unix_timestamp(sent)) AS avg_time_to_expire_seconds
...>   FROM
...>     streaming_data
...>   WHERE
...>     messageType = 'Alert'
...>     AND sent IS NOT NULL
...>     AND expires IS NOT NULL;

```

```

ends, status, message_type, category, severity, certainty, urgency, e_____, sender, sendername, headline, description, instruction, response) (state=42000,code=10004)
0: jdbc:hive2://localhost:10000> SELECT
...     . . . > FROM
...     . . . > streaming_data
...     . . . > WHERE
...     . . . >     message_type = 'Alert'
...     . . . >     AND sent IS NOT NULL
...     . . . >     AND expires IS NOT NULL;
INFO : Compiling command(queryId=hive_20250423001332_a10f7514-3d05-41d6-8235-174afed6dd87): SELECT
AVG(unix_timestamp(expires) - unix_timestamp(sent)) AS avg_time_to_expire_seconds
FROM
streaming_data
WHERE
message_type = 'Alert'
AND sent IS NOT NULL
AND expires IS NOT NULL
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Semantic Analysis Completed (retiring: false)
INFO : Returning Hive schema: FieldSchema(name:avg_time_to_expire_seconds, type:double, comment:null), properties:null
INFO : Compiler version: 1.1.0-SNAPSHOT(queryId:hive_20250423001332_a10f7514-3d05-41d6-8235-174afed6dd87); Time taken: 0.15 seconds
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Executing command(queryId=hive_20250423001332_a10f7514-3d05-41d6-8235-174afed6dd87): SELECT
AVG(unix_timestamp(expires) - unix_timestamp(sent)) AS avg_time_to_expire_seconds
FROM
streaming_data
WHERE
message_type = 'Alert'
AND sent IS NOT NULL
AND expires IS NOT NULL
INFO : Query ID = hive_20250423001332_a10f7514-3d05-41d6-8235-174afed6dd87
INFO : Total jobs = 1
INFO :Launching Job [Stage-1:MAPPED] in serial mode
INFO : Subscribed to counters: [] for queryId: hive_20250423001332_a10f7514-3d05-41d6-8235-174afed6dd87
INFO : Session is already open
INFO : Data name: SELECT
AVG(unix_timestamp(expires) - ...NULL (Stage-1)
INFO : Status: Running (Executing on YARN cluster with App id application_1745349774478_0013)

-----  

VERTICES   MODE    STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED  

Map 1 ..... container  SUCCEEDED   1      1      0      0      0      0  

Reducer 2 ..... container  SUCCEEDED   1      1      0      0      0      0  

-----  

VERIFIED: 07/02 [00:00:00.000] => 1001 ELAPSED TIME: 6.21 s  

INFO : Completed executing command(queryId=hive_20250423001332_a10f7514-3d05-41d6-8235-174afed6dd87); Time taken: 6.945 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock manager
+-----+
| avg_time_to_expire_seconds |
+-----+
| NULL |
+-----+
1 row selected (7.128 seconds)
0: jdbc:hive2://localhost:10000>
0: jdbc:hive2://localhost:10000> []

```

*Figure 7.5.* HiveQL query computes the average time elapsed between the sending of an alert and its expiration. This query aims to assess the entire timing and quality of the data but it has not returned any result. Screenshot taken by Shivangi Borad.

We used Apache Hive to run queries on cyclone alert information that is stored in our cluster. Using Hive, we were able to sort and examine the data with relatively simple SQL-like queries. Our queries included timestamp conversion, grouping, aggregations, and filtering, and enabled us to glean rich information on alert frequency, severity, urgency, and wording. This is helpful for us in transforming alert data into meaningful summaries for better situational awareness and disaster planning.

# *Apache Spark: Streaming Data*

```
SSH-in-browser UPLOAD FILE DOWNLOAD FILE
Linux storm-databipeline-group5- m 5.10.0-0.deb10.16-cloud-amd64 #1 SMP Debian 5.10.127-2-bpo10+1 (2022-07-28) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

Last boot: Tue Mar 29 2025 from 35.235.245.130
ssh-add -L
storm-databipeline-group5:m ~ $ spark-sql
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
ivysettings.xml file not found in HIVE_HOME or HIVE_CONF_DIR, /etc/hive/conf.dist/ivysettings.xml will be used
23/04/23 00:46:17 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
23/04/23 00:46:17 INFO org.apache.spark.SparkEnv: Registering BlockManagerMasterHeartbeat
23/04/23 00:46:17 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
SparkSession available as spark.
spark> sql show tables
default streaming_data false
Time taken: 3.248 seconds, Fetched 1 row(s)
spark> sparkSQL SELECT
    >   DATE(CAST(TRANSLATE(SUBSTR(sent, 1, 19), 'T', ' ') AS TIMESTAMP)) AS alert_date,
    >   DATE(CAST(TRANSLATE(SUBSTR(expires, 1, 19), 'T', ' ') AS TIMESTAMP)) AS expire_date,
    >   COUNT(*) AS daily_alerts,
    >   severity
    > FROM
    >   streaming_data
    > WHERE
    >   messageType = 'Alert'
    > GROUP BY
    >   DATE(CAST(TRANSLATE(SUBSTR(sent, 1, 19), 'T', ' ') AS TIMESTAMP)),
    >   DATE(CAST(TRANSLATE(SUBSTR(expires, 1, 19), 'T', ' ') AS TIMESTAMP)),
    >   severity
25/04/23 00:48:12 WARN org.apache.hadoop.hive.session.SessionState: METASTORE_FILTER_HOOK will be ignored, since hive.security.authorization.manager is set to instance of HiveAuthorizerFactory.
2025-04-20 2025-04-20 314 Severe
2025-04-20 2025-04-20 64 Severe
2025-04-20 2025-04-21 10 Extreme
2025-04-20 2025-04-21 10 Moderate
2025-04-20 2025-04-22 8 Unknown
2025-04-20 2025-04-20 178 Extreme
2025-04-20 2025-04-20 56 Unknown
2025-04-20 2025-04-20 75 Minor
2025-04-20 2025-04-20 147 Moderate
2025-04-20 2025-04-21 73 Minor
Time taken: 8.952 seconds, Fetched 10 row(s)
spark> sql
```

*Figure 8.1.* The Spark SQL query is grouped by day and performs aggregations on cyclone alerts according to their level of severity so as to allow for distributed computation for fast analysis on the alert trends as they occur. Screenshot taken by Shivangi Borad.

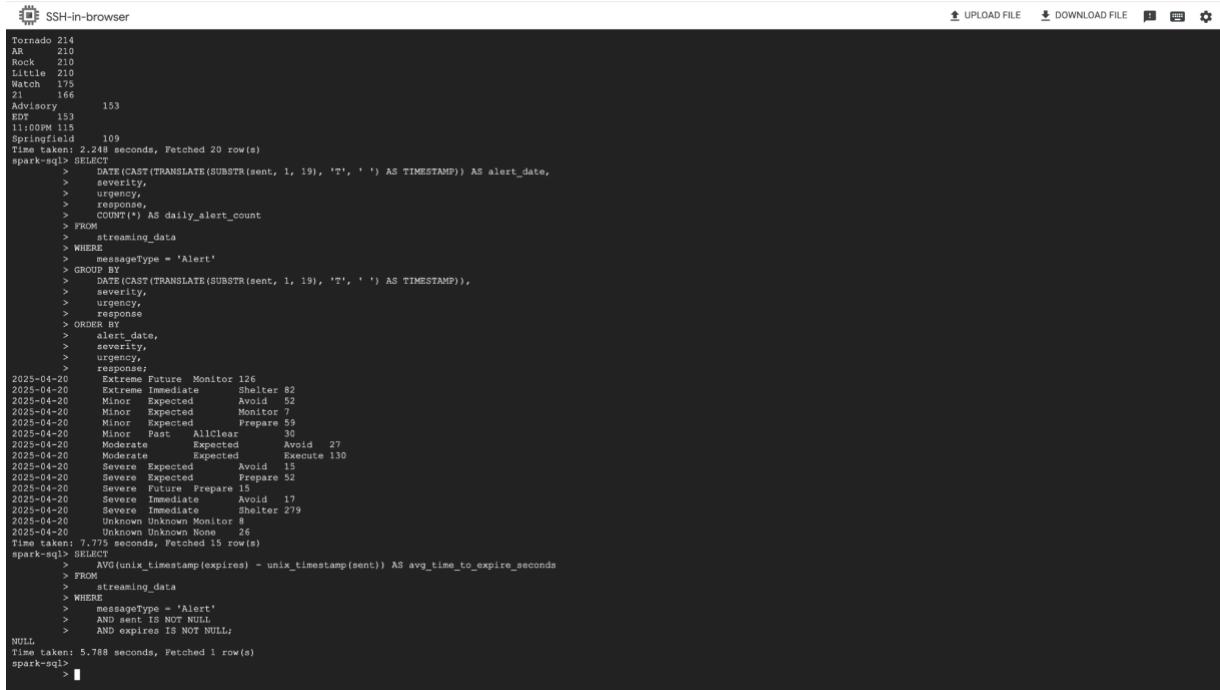
*Figure 8.2.* A Spark SQL query counts the alert notifications and calculates their percentages based on severity levels via parallel processing to help indicate the highly represented cyclone alert categories. Screenshot taken by Shivangi Borad.

```
SSH-in-browser UPLOAD FILE DOWNLOAD FILE
> severity
> ORDER BY
> alert count DESC;
25/04/23 00:50:05 WARN org.apache.spark.sql.execution.window.WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
25/04/23 00:50:05 WARN org.apache.hadoop.util.concurrent.ExecutorHelper: Thread [Thread@GetFileInfo #1,main] interrupted: java.lang.InterruptedException
at com.google.common.util.concurrent.AbstractFuture.get(AbstractFuture.java:510)
at com.google.common.util.concurrent.FluentFuture$TrustedFuture.get(FluentFuture.java:88)
at org.apache.hadoop.util.concurrent.HadoopThreadPoolExecutor$HadoopThreadExecutorHelper.java:89
at org.apache.hadoop.util.concurrent.HadoopThreadPoolExecutor.afterExecute(HadoopThreadPoolExecutor.java:90)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:157)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:760)
25/04/23 00:50:12 WARN org.apache.spark.sql.execution.window.WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
25/04/23 00:50:12 WARN org.apache.spark.sql.execution.window.WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
25/04/23 00:50:12 WARN org.apache.spark.sql.execution.window.WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
Severe 378 40.86486486486486
Extreme 208 23.486666666666666
Moderate 157 16.97397297297297
Minor 148 16.00000000000000
Unknown 34 3.67567567567567568
Time taken: 8.673 seconds, Fetched 5 row(s)
spark-sql> SELECT
    > word,
    > COUNT(*) AS word_count
    > FROM
    > streaming_data
    > LATERAL VIEW explode(split(headline, ' ')) exploded_table AS word
    > WHERE
    > message_type = 'Alert'
    > AND lower(word) NOT IN ('the', 'and', 'is', 'of', 'a', 'an', 'in', 'to', 'for', 'on', 'by', 'at', 'it', 'this', 'that', 'these', 'those', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has',
'had', 'do', 'does', 'did', 'can', 'will', 'would', 'should', 'could', 'from', 'with', 'its', 'as')
'April' 1678
20 1467
CDT 1356
issued 892
SWS 72
Hail 786
Warning 489
Severe 288
Thunderstorm 288
MJO 6
Tornado 214
AR 210
Rock 210
Little 210
March 157
21 166
Advisory 153
EDT 133
1100PM 135
Springfield 109
Time taken: 2.248 seconds, Fetched 20 row(s)
spark-sql>
```

*Figure 8.3.* The Spark SQL query transforms the alert headlines, removes stopwords, and finds the most common important words, aiding the fast and scalable extraction of keywords from stream data.  
Screenshot taken by Shivangi Borad.

```
SSH-in-browser
UPLOAD FILE DOWNLOAD FILE
April 1678
20 1467
CDX 1156
issued 892
NWS 892
until 786
Warning 3
Clouds 288
Thunderstorm 288
MO 236
Dernado 214
AR 210
Rock 210
Little 210
Watch 175
JL 146
Advisory 153
EDT 153
11:00PM 15
Springfield 109
Time taken: 2.248 seconds, Fetched 20 row(s)
spark-sql> SELECT
    > DATE(CAST(TRANSLATE(SUBSTR(sent, 1, 19), 'T', ' ') AS TIMESTAMP)) AS alert_date,
    > severity,
    > urgency,
    > response,
    > COUNT(*) AS daily_alert_count
    > FROM
    >     streaming_data
    > WHERE
    >     messageType = 'Alert'
    > GROUP BY
    >     DATE(CAST(TRANSLATE(SUBSTR(sent, 1, 19), 'T', ' ') AS TIMESTAMP)),
    >     severity,
    >     urgency,
    >     response
    > ORDER BY
    >     alert_date,
    >     severity,
    >     urgency,
    >     response
2025-04-20 Extreme Future Monitor 126
2025-04-20 Extreme Immediate Shelter 82
2025-04-20 Minor Expected Avoid 52
2025-04-20 Minor Expected Monitor 7
2025-04-20 Minor Expected Prepare 59
2025-04-20 Minor Past Alclear 30
2025-04-20 Moderate Expected Avoid 27
2025-04-20 Moderate Expected Execute 130
2025-04-20 Severe Expected Avoid 18
2025-04-20 Severe Expected Prepare 52
2025-04-20 Severe Future Prepare 15
2025-04-20 Severe Immediate Avoid 17
2025-04-20 Severe Immediate Shelter 279
2025-04-20 Unknown Unknown Monitor 8
2025-04-20 Unknown Unknown None 26
Time taken: 7.775 seconds, Fetched 15 row(s)
spark-sql> |
```

*Figure 8.4.* A Spark SQL query groups the alert data by several fields such as date, severity, urgency, and response type, employing Spark's distributed platform to detect the daily changes in alerting strategies. Screenshot taken by Shivangi Borad.



```

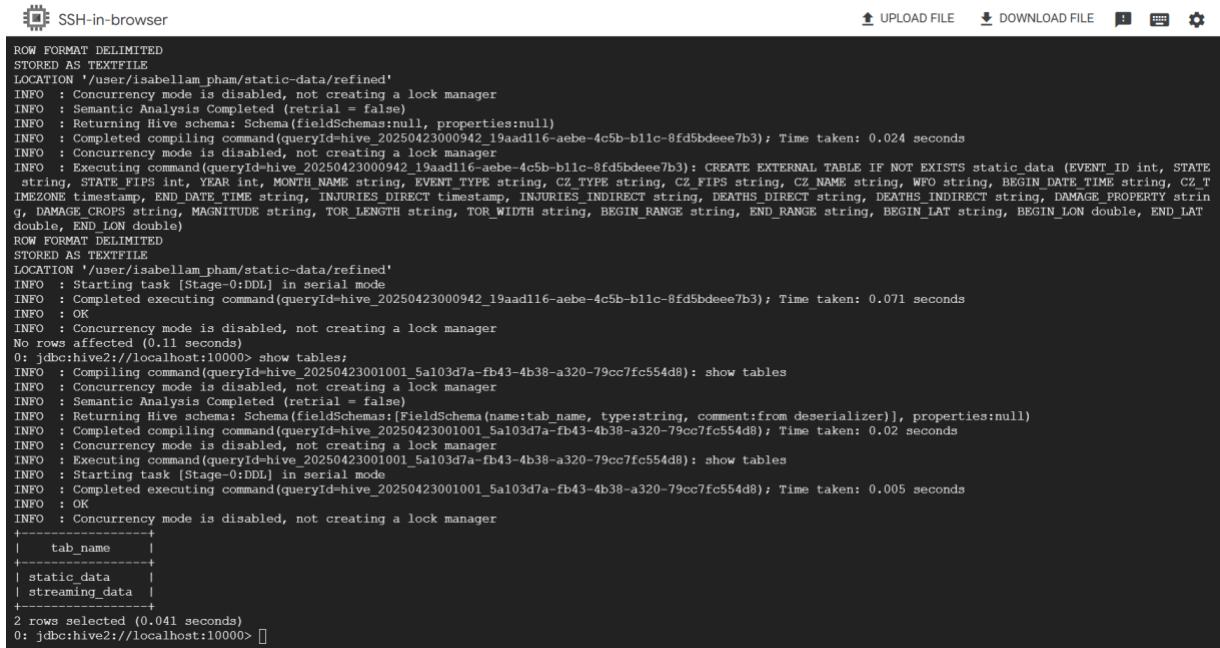
SSH-in-browser
UPLOAD FILE DOWNLOAD FILE
Tornado 214
AR 0
Rock 210
Little 210
Watch 175
ZJ 166
Advisory 153
EDT 153
11:00PM 115
Springfield 109
Time taken: 2.26 seconds, Fetched 20 rows(s)
spark> SELECT
    >     DATE(CAST(TRANSLATE(SUBSTR(sent, 1, 19), 'T', ' ') AS TIMESTAMP)) AS alert_date,
    >     severity,
    >     urgency,
    >     timestamp,
    >     COUNT(*) AS daily_alert_count
    > FROM
    >     streaming_data
    > WHERE
    >     messageType = 'Alert'
    > GROUP BY
    >     DATE(CAST(TRANSLATE(SUBSTR(sent, 1, 19), 'T', ' ') AS TIMESTAMP)),
    >     severity,
    >     urgency,
    >     response
    > ORDER BY
    >     alert_date,
    >     severity,
    >     urgency,
    >     response;
2025-04-20 Extreme Future Monitor 126
2025-04-20 Extreme Immediate Shelter 82
2025-04-20 Minor Expected Avoid 52
2025-04-20 Minor Expected Monitor 7
2025-04-20 Minor Expected Prepare 59
2025-04-20 Minor Fast AllClear 30
2025-04-20 Moderate Expected Avoid 27
2025-04-20 Moderate Expected Execute 130
2025-04-20 Severe Expected Avoid 15
2025-04-20 Severe Expected Prepare 52
2025-04-20 Severe Immediate Prepare 15
2025-04-20 Severe Immediate Avoid 17
2025-04-20 Severe Immediate Shelter 279
2025-04-20 Unknown Unknown Monitor 8
2025-04-20 Unknown Unknown None 26
Time taken: 7.78 seconds, Fetched 15 rows(s)
spark> SELECT
    >     AVG(unix_timestamp(expires) - unix_timestamp(sent)) AS avg_time_to_expire_seconds
    > FROM
    >     streaming_data
    > WHERE
    >     messageType = 'Alert'
    >     AND sent IS NOT NULL
    >     AND expires IS NOT NULL;
NULL
Time taken: 5.788 seconds, Fetched 1 row(s)
spark>

```

*Figure 8.5.* A Spark SQL query determines the average time lag between the alert being issued and its expiration with an effort to analyze the performance and timing of their warning system through the advantage of speed provided by Spark. Screenshot taken by Shivangi Borad.

We used the Apache Spark SQL to efficiently handle alert data and effectively process massive amounts of data. Using Spark SQL, we performed operations like timestamps and group aggregations on a large scale. This aspect improved our data processing, enabling us to compute repetitive word counts from alert information and determine message lengths. With Spark SQL, we had a seamless workflow despite high levels of data.

## Apache Hive: Static Data



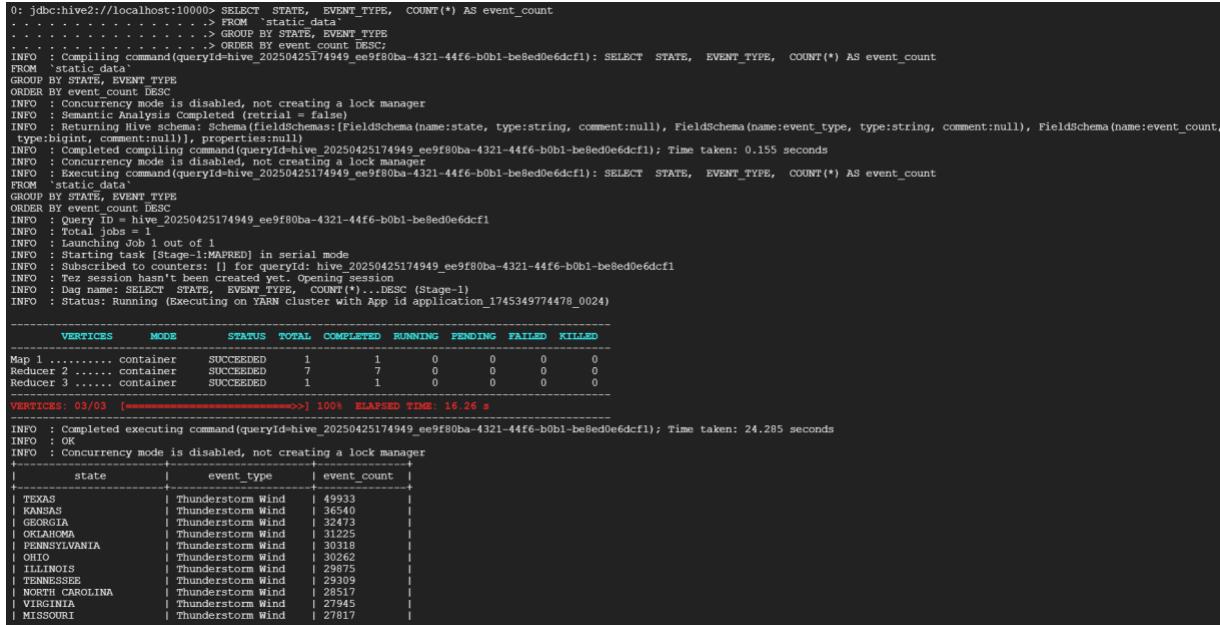
The screenshot shows the Apache Hive interface in a browser. At the top, there are buttons for 'UPLOAD FILE', 'DOWNLOAD FILE', and other settings. The main area displays the command-line interface for creating a table:

```

ROW FORMAT DELIMITED
STORED AS TEXTFILE
LOCATION '/user/isabellam_pham/static-data/refined'
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Semantic Analysis Completed (retryal = false)
INFO : Returning Hive schema: Schema(fieldSchemas:null, properties:null)
INFO : Completed compiling command(queryId:hive_20250423000942_19aad116-aebc-4c5b-b11c-8fd5bdeee7b3); Time taken: 0.024 seconds
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Executing command(queryId:hive_20250423000942_19aad116-aebc-4c5b-b11c-8fd5bdeee7b3): CREATE EXTERNAL TABLE IF NOT EXISTS static_data (EVENT_ID int, STATE string, STATE_FIPS int, YEAR int, MONTH_NAME string, EVENT_TYPE string, CZ_TYPE string, CZ_FIPS string, CZ_NAME string, WFO string, BEGIN_DATE_TIME string, CZ_TIMEZONE timestamp, END_DATE_TIME string, INJURIES_DIRECT string, INJURIES INDIRECT string, DEATHS_DIRECT string, DEATHS INDIRECT string, DAMAGE_PROPERTY string, DAMAGE_CROPS string, MAGNITUDE string, TOR_LENGTH string, TOR_WIDTH string, BEGIN_RANGE string, END_RANGE string, BEGIN_LAT string, BEGIN_LON double, END_LON double)
ROW FORMAT DELIMITED
STORED AS TEXTFILE
LOCATION '/user/isabellam_pham/static-data/refined'
INFO : Starting task [Stage-0:DDL] in serial mode
INFO : Completed executing command(queryId:hive_20250423000942_19aad116-aebc-4c5b-b11c-8fd5bdeee7b3); Time taken: 0.071 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock manager
No rows affected (0.11 seconds)
0: jdbc:hive2://localhost:10000> show tables;
INFO : Compiling command(queryId:hive_20250423001001_5a103d7a-fb43-4b38-a320-79cc7fc554d8): show tables
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Semantic Analysis Completed (retryal = false)
INFO : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:tab_name, type:string, comment:from deserializer)], properties:null)
INFO : Completed compiling command(queryId:hive_20250423001001_5a103d7a-fb43-4b38-a320-79cc7fc554d8); Time taken: 0.02 seconds
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Executing command(queryId:hive_20250423001001_5a103d7a-fb43-4b38-a320-79cc7fc554d8): show tables
INFO : Starting task [Stage-0:DDL] in serial mode
INFO : Completed executing command(queryId:hive_20250423001001_5a103d7a-fb43-4b38-a320-79cc7fc554d8); Time taken: 0.005 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock manager
+-----+
| tab_name |
+-----+
| static_data |
| streaming_data |
+-----+
2 rows selected (0.041 seconds)
0: jdbc:hive2://localhost:10000> []

```

Figure 9.1. Creation of an Apache Hive table for static data. Screenshot taken by Isabella Pham.



The screenshot shows the Apache Hive interface in a browser. The command-line interface is used to run a query:

```

0: jdbc:hive2://localhost:10000> SELECT STATE, EVENT_TYPE, COUNT(*) AS event_count
FROM static_data
GROUP BY STATE, EVENT_TYPE
ORDER BY event_count DESC;
INFO : Compiling command(queryId:hive_20250425174949_ee9f80ba-4321-44f6-b0b1-be8ed0e6dcf1): SELECT STATE, EVENT_TYPE, COUNT(*) AS event_count
FROM static_data
GROUP BY STATE, EVENT_TYPE
ORDER BY event_count DESC;
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Semantic Analysis Completed (retryal = false)
INFO : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:state, type:string, comment:null), FieldSchema(name:event_type, type:string, comment:null), FieldSchema(name:event_count, type:bigint, comment:null)], properties:null)
INFO : Completed compiling command(queryId:hive_20250425174949_ee9f80ba-4321-44f6-b0b1-be8ed0e6dcf1); Time taken: 0.155 seconds
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Executing command(queryId:hive_20250425174949_ee9f80ba-4321-44f6-b0b1-be8ed0e6dcf1): SELECT STATE, EVENT_TYPE, COUNT(*) AS event_count
FROM static_data
GROUP BY STATE, EVENT_TYPE
ORDER BY event_count DESC
INFO : Query ID = hive_20250425174949_ee9f80ba-4321-44f6-b0b1-be8ed0e6dcf1
INFO : Tez session hasn't been created yet. Opening session
INFO : Starting task [Stage-1:MAPRED] in serial mode
INFO : Subscribed to counters: [] for queryid: hive_20250425174949_ee9f80ba-4321-44f6-b0b1-be8ed0e6dcf1
INFO : Tez session hasn't been created yet. Opening session
INFO : Dag name: SELECT STATE, EVENT_TYPE, COUNT(*)...DESC (Stage-1)
INFO : Status: Running (Executing on YARN cluster with App id application_1745349774470_0024)

VERTICES      MODE      STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED
Map 1 ..... container SUCCEEDED   1    1    0    0    0    0
Reducer ..... container SUCCEEDED   7    7    0    0    0    0
Reducer 3 ..... container SUCCEEDED   1    1    0    0    0    0

VERTICES: 03/03 [=====>>>] 100% ELAPSED TIME: 16.26 s
INFO : Completed executing command(queryId:hive_20250425174949_ee9f80ba-4321-44f6-b0b1-be8ed0e6dcf1); Time taken: 24.285 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock manager
+-----+
| state | event_type | event_count |
+-----+
| TEXAS | Thunderstorm Wind | 49933
| KANSAS | Thunderstorm Wind | 36540
| GEORGIA | Thunderstorm Wind | 32473
| OKLAHOMA | Thunderstorm Wind | 31225
| PENNSYLVANIA | Thunderstorm Wind | 30318
| OHIO | Thunderstorm Wind | 30262
| ILLINOIS | Thunderstorm Wind | 29575
| TENNESSEE | Thunderstorm Wind | 28309
| NORTH CAROLINA | Thunderstorm Wind | 28517
| VIRGINIA | Thunderstorm Wind | 27945
| MISSOURI | Thunderstorm Wind | 27817

```

Figure 9.2. HiveQL query requesting data on the number of storm events per storm event type, grouped by storm type and state and ordered by the number of events in descending order. Screenshot taken by Isabella Pham. (Not Pictured: total elapsed time 24.554 seconds)

*Figure 9.3.* HiveQL query summarizing the total cost of (estimated) property damage in USD, grouped by storm event type and organized in descending order by damage amount. Screenshot taken by Isabella Pham.

```

0: jdbc:hive2://localhost:10000> SELECT STATE, SUM(CAST(INJURIES_DIRECT AS int) + CAST(INJURIES_INDIRECT AS int)) AS total_injuries
   FROM `static_data` 
  GROUP BY STATE
 ORDER BY total_injuries DESC
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Semantic Analysis Completed (retention: false)
INFO : Executing command(queryId=hive_20250425175214_493c2a7b-4dc5-4930-9101-65e48b4261f2): SELECT STATE, SUM(CAST(INJURIES_DIRECT AS int) + CAST(INJURIES_INDIRECT AS int)) AS total_injuries
FROM `static_data` 
GROUP BY STATE
ORDER BY total_injuries DESC
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Semantic Analysis Completed (retention: false)
INFO : Executed compiling command(queryId=hive_20250425175214_493c2a7b-4dc5-4930-9101-65e48b4261f2); Time taken: 0.152 seconds
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Executing command(queryId=hive_20250425175214_493c2a7b-4dc5-4930-9101-65e48b4261f2): SELECT STATE, SUM(CAST(INJURIES_DIRECT AS int) + CAST(INJURIES_INDIRECT AS int)) AS total_injuries
FROM `static_data` 
GROUP BY STATE
ORDER BY total_injuries DESC
INFO : Query ID = hive_20250425175214_493c2a7b-4dc5-4930-9101-65e48b4261f2
INFO : Total jobs = 1
INFO :Launching Job 1 out of 1
INFO : Starting Map [Stage-1] in serial mode
INFO : Subscribed to counters: [] for queryid: hive_20250425175214_493c2a7b-4dc5-4930-9101-65e48b4261f2
INFO : Session is already open
INFO : Dag name: SELECT STATE, SUM(CAST(INJURIES_DIRECT.. DESC (Stage-1)
INFO : Status: Running (Executing on YARN cluster with App id application_1745349774478_0024)

----- VERTICES ----- MODE ----- STATUS ----- TOTAL ----- COMPLETED ----- RUNNING ----- FENDING ----- FAILED ----- KILLED -----
Map 1 ..... container SUCCEEDED 1 1 0 0 0 0
Reducer 2 ..... container SUCCEEDED 7 7 0 0 0 0
Reducer 3 ..... container SUCCEEDED 1 1 0 0 0 0
----- VERTICES: 03/03 [=====>>>>>] 100% ELAPSED TIME: 17.56 s

INFO : Completed executing command(queryId=hive_20250425175214_493c2a7b-4dc5-4930-9101-65e48b4261f2); Time taken: 18.271 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock manager
-----+-----+-----+
| state | total injuries |
-----+-----+-----+
| TEXAS | 17086 |
| ALABAMA | 12556 |
| MISSISSIPPI | 8898 |
| OKLAHOMA | 8221 |
| MISSOURI | 7772 |
| ARKANSAS | 7405 |
| TENNESSEE | 7226 |
| ILLINOIS | 6374 |
| GEORGIA | 6341 |
| FLORIDA | 6305 |
-----+-----+-----+

```

*Figure 9.4.* HiveQL query summarizing the total number of injuries (both direct and indirect) caused by severe storms; grouped by state and organized in descending order by number of injuries. Screenshot taken by Isabella Pham. (Not Pictured: total elapsed time 18.463 seconds)

```

0: jdbc:hive2://localhost:10000> SELECT STATE, SUM(CAST(DEATHS_DIRECT AS int) + CAST(DEATHS INDIRECT AS int)) AS total_deaths
   . . .
   . . . > FROM `static data`
   . . . > GROUP BY STATE
   . . . > ORDER BY total_deaths DESC
   . . . > LIMIT 10;
INFO : Compiling command(queryId=hive_20250425175350_19f69641-7c75-42c9-9d5e-a8d6cb9dca02): SELECT STATE, SUM(CAST(DEATHS_DIRECT AS int) + CAST(DEATHS INDIRECT AS int)) AS total_deaths
FROM `static data`
GROUP BY STATE
ORDER BY total_deaths DESC
LIMIT 10
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Semantic Analysis Completed (retryal = false)
INFO : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:state, type:string, comment:null), FieldSchema(name:total_deaths, type:bigrint, comment:null)], properties:null)
INFO : Completed compiling command(queryId=hive_20250425175350_19f69641-7c75-42c9-9d5e-a8d6cb9dca02); Time taken: 0.116 seconds
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Executing command(queryId=hive_20250425175350_19f69641-7c75-42c9-9d5e-a8d6cb9dca02): SELECT STATE, SUM(CAST(DEATHS_DIRECT AS int) + CAST(DEATHS INDIRECT AS int)) AS total_deaths
FROM `static data`
GROUP BY STATE
ORDER BY total_deaths DESC
LIMIT 10
INFO : Query ID = hive_20250425175350_19f69641-7c75-42c9-9d5e-a8d6cb9dca02
INFO : Total jobs = 1
INFO : Launching Job 1 out of 1
INFO : Starting task [Stage-1:MAPRED] in serial mode
INFO : Subscribed to counters: [] for queryid: hive_20250425175350_19f69641-7c75-42c9-9d5e-a8d6cb9dca02
INFO : Session is already open
INFO : Dag name: SELECT STATE, SUM(CAST(DEATHS DIRECT AS...10 (Stage-1)
INFO : Status: Running (Executing on YARN cluster with App id application_1745349774478_0024)

----- VERTICES MODE STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED -----
Map 1 ..... container SUCCEEDED 1 1 0 0 0 0
Reducer 2 ..... container SUCCEEDED 7 7 0 0 0 0
Reducer 3 ..... container SUCCEEDED 1 1 0 0 0 0
----- VERTICES: 03/03 [=====>] 100% ELAPSED TIME: 16.41 s
----- VERTICES: 03/03 [=====>] 100% ELAPSED TIME: 16.41 s
INFO : Completed executing command(queryId=hive_20250425175350_19f69641-7c75-42c9-9d5e-a8d6cb9dca02); Time taken: 17.085 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock manager
-----+-----+-----+
| state | total_deaths |
-----+-----+
| LOUISIANA | 1154 |
| ALABAMA | 1132 |
| TEXAS | 1124 |
| MISSISSIPPI | 862 |
| MISSOURI | 744 |
| TENNESSEE | 636 |
| ARKANSAS | 580 |
| FLORIDA | 564 |
| OKLAHOMA | 551 |
| INDIANA | 415 |
-----+-----+
10 rows selected (17.229 seconds)

```

Figure 9.5. HiveQL query requesting a list of the 10 U.S. states with the highest death tolls caused by severe storms; organized in descending order by the number of deaths. Screenshot taken by Isabella Pham.

```

0: jdbc:hive2://localhost:10000> SELECT YEAR, MONTH_NAME, COUNT(*) AS event_count, SUM(CAST(INJURIES_DIRECT AS int) + CAST(INJURIES INDIRECT AS int)) AS total_injuries, SUM(CAST(DEATHS_DIRECT AS int) + CAST(DEATHS INDIRECT AS int)) AS total_deaths, SUM(CAST(DAMAGE_PROPERTY AS int)) AS total_property_damage, SUM(CAST(DAMAGE_CROPS AS int)) AS total_crops_damage
   . . .
   . . . > FROM `static data`
   . . . > GROUP BY YEAR, MONTH_NAME
   . . . > ORDER BY event_count DESC
   . . . > LIMIT 100;
INFO : Compiling command(queryId=hive_20250425175545_ad1c5593-aa87-4d77-aa0b-b746ade7e0df): SELECT YEAR, MONTH_NAME, COUNT(*) AS event count, SUM(CAST(INJURIES DIRECT AS int) + CAST(INJURIES INDIRECT AS int)) AS total injuries, SUM(CAST(DEATHS DIRECT AS int) + CAST(DEATHS INDIRECT AS int)) AS total deaths, SUM(CAST(DAMAGE PROPERTY AS int)) AS total_property_damage, SUM(CAST(DAMAGE CROPS AS int)) AS total_crops_damage
FROM `static data`
GROUP BY YEAR, MONTH NAME
ORDER BY event_count DESC
LIMIT 100
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Semantic Analysis Completed (retryal = false)
INFO : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:year, type:string, comment:null), FieldSchema(name:month_name, type:string, comment:null), FieldSchema(name:event_count, type:bigrint, comment:null), FieldSchema(name:total_injuries, type:bigrint, comment:null), FieldSchema(name:total_deaths, type:bigrint, comment:null), FieldSchema(name:total_property_damage, type:bigrint, comment:null), FieldSchema(name:total_crops_damage, type:bigrint, comment:null)], properties:null)
INFO : Completed compiling command(queryId=hive_20250425175545_ad1c5593-aa87-4d77-aa0b-b746ade7e0df); Time taken: 0.119 seconds
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Executing command(queryId=hive_20250425175545_ad1c5593-aa87-4d77-aa0b-b746ade7e0df): SELECT YEAR, MONTH_NAME, COUNT(*) AS event_count, SUM(CAST(INJURIES_DIRECT AS int) + CAST(INJURIES INDIRECT AS int)) AS total_injuries, SUM(CAST(DEATHS_DIRECT AS int) + CAST(DEATHS INDIRECT AS int)) AS total_deaths, SUM(CAST(DAMAGE_PROPERTY AS int)) AS total_property_damage, SUM(CAST(DAMAGE_CROPS AS int)) AS total_crops_damage
FROM `static data`
GROUP BY YEAR, MONTH NAME
ORDER BY event_count DESC
LIMIT 100
INFO : Query ID = hive_20250425175545_ad1c5593-aa87-4d77-aa0b-b746ade7e0df
INFO : Total jobs = 1
INFO : Launching Job 1 out of 1
INFO : Starting task [Stage-1:MAPRED] in serial mode
INFO : Subscribed to counters: [] for queryid: hive_20250425175545_ad1c5593-aa87-4d77-aa0b-b746ade7e0df
INFO : Session is already open
INFO : Dag name: SELECT YEAR, MONTH_NAME, COUNT(*) AS e...100 (Stage-1)
INFO : Status: Running (Executing on YARN cluster with App id application_1745349774478_0024)

----- VERTICES MODE STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED -----
Map 1 ..... container SUCCEEDED 1 1 0 0 0 0
Reducer 2 ..... container SUCCEEDED 7 7 0 0 0 0
Reducer 3 ..... container SUCCEEDED 1 1 0 0 0 0
----- VERTICES: 03/03 [=====>] 100% ELAPSED TIME: 17.74 s
----- VERTICES: 03/03 [=====>] 100% ELAPSED TIME: 17.74 s
INFO : Completed executing command(queryId=hive_20250425175545_ad1c5593-aa87-4d77-aa0b-b746ade7e0df); Time taken: 18.423 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock manager
-----+-----+-----+-----+-----+
| year | month_name | event_count | total_injuries | total_deaths | total_property_damage | total_crops_damage |
-----+-----+-----+-----+-----+
| 2008 | June | 377 | 56 | 2099460900 | 25697000 |
| 2012 | July | 10604 | 226 | 46 | 118776500 | 1985000 |
-----+-----+

```

Figure 9.6. HiveQL query requesting a list of the top 100 dates (by year and month) in which the most storm events occurred, along with associated data relating to the total number of injuries, deaths, the (estimated) value of property damage, and the (estimated) value of crop damage. All crop and property damages are measured in USD. Screenshot taken by Isabella Pham. (Not Pictured: total elapsed time 18.571 seconds)

```

0: jdbc:hive2://localhost:10000> WITH target_years AS (SELECT EXTRACT(YEAR FROM CURRENT_DATE()) - 1 AS last_year, EXTRACT(YEAR FROM CURRENT_DATE()) - 10 AS ten_years_ago), filtered_data AS (SELECT * FROM target_years UNION ALL SELECT * FROM target_years WHERE YEAR IN (SELECT last_year FROM target_years UNION ALL SELECT ten_years_ago FROM target_years))
          . . . . . > SELECT YEAR, COUNT(*) AS event_count, SUM(total_deaths) AS total_deaths, SUM(total_injuries) AS total_injuries, SUM(property_damage) AS total_property_damage
          . . . . . , SUM(crop_damage) AS total_crop_damage
          . . . . . FROM filtered_data
          . . . . . GROUP BY YEAR
          . . . . . ORDER BY YEAR;
INFO : Compiling command(queryId=hive-20250425175729_59badf5-b841-4785-8910-d95e0ad190ed): WITH target_years AS (SELECT EXTRACT(YEAR FROM CURRENT_DATE()) - 1 AS last_year, EXTRACT(YEAR FROM CURRENT_DATE()) - 10 AS ten_years_ago), filtered_data AS (SELECT * FROM target_years UNION ALL SELECT * FROM target_years WHERE YEAR IN (SELECT last_year FROM target_years UNION ALL SELECT ten_years_ago FROM target_years))
          . . . . . > SELECT YEAR, COUNT(*) AS event_count, SUM(total_deaths) AS total_deaths, SUM(total_injuries) AS total_injuries, SUM(property_damage) AS total_property_damage, SUM(crop_damage) AS total_crop_damage
          . . . . . FROM filtered_data
          . . . . . GROUP BY YEAR
          . . . . . ORDER BY YEAR;
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Statistics Analysis Completed (trial= false)
INFO : Returning Hive schema: Schema(fieldschemas:[FieldSchema(name:year, type:string, comment:null), FieldSchema(name:event count, type:bigrint, comment:null), FieldSchema(name:total_property_damage, type:bigrint, comment:null), FieldSchema(name:total_crops_damage, type:bigrint, comment:null)], properties:null)
INFO : Completed compiling command(queryId=hive-20250425175729_59badf5-b841-4785-8910-d95e0ad190ed): Time taken: 0.263 seconds
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Executing command(queryId=hive-20250425175729_59badf5-b841-4785-8910-d95e0ad190ed): WITH target_years AS (SELECT EXTRACT(YEAR FROM CURRENT_DATE()) - 1 AS last_year, EXTRACT(YEAR FROM CURRENT_DATE()) - 10 AS ten_years_ago), filtered_data AS (SELECT * FROM target_years UNION ALL SELECT * FROM target_years WHERE YEAR IN (SELECT last_year FROM target_years UNION ALL SELECT ten_years_ago FROM target_years))
          . . . . . > SELECT * FROM filtered_data
          . . . . . GROUP BY YEAR
          . . . . . ORDER BY YEAR;
INFO : Query ID = hive_20250425175729_59badf5-b841-4785-8910-d95e0ad190ed
INFO : Total jobs = 1
INFO :Launching Job 1 out of 1
INFO : Running a local job since 1-WARERED in serial mode
INFO : Subscribed to counters: [] for queryId: hive_20250425175729_59badf5-b841-4785-8910-d95e0ad190ed
INFO : Session is already open
INFO : Dag name: WITH target years AS (SELECT EXTRACT(...YEAR (Stage-1)
INFO : Setting tez.task.scale.memory.reserve_fraction to 0.30000000192092896
INFO : Status: Running (Executing on YARN cluster with App id application_1745349774478_0024)

----- VERTICES MODE STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED -----
Map 1 ..... container SUCCEEDED 1 1 0 0 0 0
Reducer 2 ..... container SUCCEEDED 8 8 0 0 0 0
Reducer 3 ..... container SUCCEEDED 1 1 0 0 0 0
Map 4 ..... container SUCCEEDED 1 1 0 0 0 0
Map 5 ..... container SUCCEEDED 1 1 0 0 0 0
Map 6 ..... container SUCCEEDED 0 0 0 0 0 0

----- VERTICES 05/05 [=====>] 100% ELAPSED TIME: 21.35 s -----

INFO : Completed executing command(queryId=hive_20250425175729_59badf5-b841-4785-8910-d95e0ad190ed): Time taken: 22.21 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock manager
+-----+-----+-----+-----+-----+-----+-----+
| year | event count | total deaths | total injuries | total_property_damage | total_crop_damage |
+-----+-----+-----+-----+-----+-----+
| 2015 | 19322 | 1238 | 639578768 | 24295600 |
| 2024 | 24428 | 147 | 924 | 7451362769 | 1558027410 |
+-----+-----+-----+-----+-----+-----+
2 rows selected (22.495 seconds)
0: jdbc:hive2://localhost:10000>
```

*Figure 9.7.* HiveQL query comparing summary data between two years roughly a decade apart, 2024 and 2015, starting with data from 2015. All (estimated) values of property and crop damage are measured in USD. Screenshot taken by Isabella Pham.

HiveQL was used to effectively extract, transform, and summarize large quantities of storm event structured data, with a specific focus on data measuring impact zones and the aftereffects of dangerous U.S.-based storms (e.g. fatalities, injuries, property damage, etc.). For the purposes of optimizing legibility and organization, explicit data types and column mappings - which aligned the Hive table with the structure of the data source - were defined using HiveQL. Beyond this, we are able to tailor queries to the unique needs of the Red Cross through the application of both HiveQL and Apache Spark, which will prove especially useful for future visualization and data analysis efforts.

## Apache Spark: Static Data

```
spark-sql> SELECT STATE, EVENT_TYPE, COUNT(*) AS event_count
  > FROM `static_data`
  > GROUP BY STATE, EVENT_TYPE
  > ORDER BY event_count DESC;
25/04/25 19:11:50 WARN org.apache.hadoop.hive.session.SessionState: METASTORE_FILTER_HOOK will be ignored, since hive.security.authorization.manager is set to instance of HiveAuthorizerFactory.
+-----+-----+-----+
| STATE | EVENT_TYPE | event_count |
+-----+-----+-----+
| TEXAS | Thunderstorm Wind | 49933 |
| KANSAS | Thunderstorm Wind | 36540 |
| GEORGIA | Thunderstorm Wind | 32473 |
| OKLAHOMA | Thunderstorm Wind | 31225 |
| PENNSYLVANIA | Thunderstorm Wind | 30318 |
| OHIO | Thunderstorm Wind | 30262 |
| ILLINOIS | Thunderstorm Wind | 29875 |
| TENNESSEE | Thunderstorm Wind | 29309 |
| NORTH CAROLINA | Thunderstorm Wind | 28517 |
| VIRGINIA | Thunderstorm Wind | 27945 |
| MISSOURI | Thunderstorm Wind | 27817 |
| IOWA | Thunderstorm Wind | 27770 |
| ALABAMA | Thunderstorm Wind | 26172 |
| NEW YORK | Thunderstorm Wind | 25468 |
| IOWA | Thunderstorm Wind | 25382 |
| MISSISSIPPI | Thunderstorm Wind | 25275 |
| ARKANSAS | Thunderstorm Wind | 23904 |
| SOUTH CAROLINA | Thunderstorm Wind | 23440 |
| INDIANA | Thunderstorm Wind | 23120 |
| FLORIDA | Thunderstorm Wind | 20588 |
| MINNESOTA | Thunderstorm Wind | 19152 |
| MICHIGAN | Thunderstorm Wind | 18967 |
| WISCONSIN | Thunderstorm Wind | 18413 |
| NEBRASKA | Thunderstorm Wind | 17923 |
| LOUISIANA | Thunderstorm Wind | 16863 |
| SOUTH DAKOTA | Thunderstorm Wind | 16465 |
| TEXAS | Tornado | 12464 |
| MARYLAND | Thunderstorm Wind | 10844 |
| WEST VIRGINIA | Thunderstorm Wind | 10417 |
| MONTANA | Thunderstorm Wind | 9218 |
| NORTH DAKOTA | Thunderstorm Wind | 9014 |
| COLORADO | Winter Storm | 8937 |
| MINNESOTA | Winter Storm | 8074 |
| NEW JERSEY | Thunderstorm Wind | 7665 |
| NEBRASKA | Winter Storm | 6793 |
| KANSAS | Tornado | 6702 |
| OKLAHOMA | Tornado | 6220 |
| MICHIGAN | Winter Storm | 6133 |
| NEW YORK | Winter Storm | 6080 |
| COLORADO | Thunderstorm Wind | 6018 |
| MASSACHUSETTS | Thunderstorm Wind | 5995 |
| IOWA | Winter Storm | 5982 |
| WISCONSIN | Winter Storm | 5618 |
| VIRGINIA | Winter Storm | 5581 |
| ARIZONA | Thunderstorm Wind | 5456 |
| WYOMING | Winter Storm | 5399 |
+-----+-----+-----+
```

*Figure 10.1.* Spark SQL query extracting data regarding the total number of events per storm type, grouped by state and organized in descending order. Screenshot taken by Isabella Pham. (Not Pictured: total elapsed time 23.685 seconds)

```
spark-sql> SELECT EVENT_TYPE, SUM(CAST(DAMAGE_PROPERTY AS int)) AS total_property_damage
  > FROM `static_data`
  > GROUP BY EVENT_TYPE
  > ORDER BY total_property_damage DESC;
25/04/25 17:35:06 WARN org.apache.hadoop.hive.session.SessionState: METASTORE_FILTER_HOOK will be ignored, since hive.security.authorization.manager is set to instance of HiveAuthorizerFactory.
+-----+-----+
| EVENT_TYPE | total_property_damage |
+-----+-----+
| Hurricane (Typhoon) | 98675768618 |
| Tornado | 96757868717 |
| Hurricane | 41151109000 |
| Thunderstorm Wind | 23589627962 |
| Winter Storm | 5445706510 |
| EVENT_TYPE | NULL |
+-----+-----+
Time taken: 26.389 seconds, Fetched 6 row(s)
```

*Figure 10.2.* Spark SQL query summarizing the total value of (estimated) property damage in USD per storm event type, organized in descending order according to damage value. Screenshot taken by Isabella Pham.

```
spark-sql> SELECT STATE, SUM(CAST(INJURIES_DIRECT AS int) + CAST(INJURIES INDIRECT AS int)) AS total_injuries
    > FROM `static_data`
    > GROUP BY STATE
    > ORDER BY total_injuries DESC;
STATE  total_injuries
TEXAS 17089
ALABAMA 12556
MISSISSIPPI 8898
OKLAHOMA 8221
MISSOURI 7772
ARKANSAS 7405
TENNESSEE 7286
ILLINOIS 6374
GEORGIA 6341
FLORIDA 6305
OHIO 5686
INDIANA 5542
KENTUCKY 5065
NORTH CAROLINA 4580
MICHIGAN 4274
KANSAS 4108
LOUISIANA 3887
IOWA 3578
PENNSYLVANIA 2580
MINNESOTA 2549
WISCONSIN 2482
SOUTH CAROLINA 2135
MASSACHUSETTS 1998
VIRGINIA 1784
NEBRASKA 1591
UTAH 1269
NEW YORK 1101
GUAM 970
SOUTH DAKOTA 950
CONNECTICUT 829
MARYLAND 775
NEW JERSEY 673
COLORADO 655
NORTH DAKOTA 551
ARIZONA 536
WYOMING 513
WASHINGTON 375
WEST VIRGINIA 364
CALIFORNIA 339
```

*Figure 10.3.* Spark SQL query summarizing the total number of injuries (both direct and indirect) caused by severe storms; organized by state and in descending order according to the number of injuries per state. Screenshot taken by Isabella Pham. (Not Pictured: total elapsed time 18.170 seconds)

```
spark-sql> SELECT STATE, SUM(CAST(DEATHS_DIRECT AS int) + CAST(DEATHS INDIRECT AS int)) AS total_deaths
    > FROM `static_data`
    > GROUP BY STATE
    > ORDER BY total_deaths DESC
    > LIMIT 10;
25/04/25 19:18:44 WARN org.apache.hadoop.util.concurrent.ExecutorHelper: Thread (Thread[GetFileInfo #1,5,main]) interrupted:
java.lang.InterruptedException
    at com.google.common.util.concurrent.AbstractFuture.get(AbstractFuture.java:510)
    at com.google.common.util.concurrent.FluentFuture$TrustedFuture.get(FluentFuture.java:88)
    at org.apache.hadoop.util.concurrent.ExecutorHelper.logThrowableFromAfterExecute(ExecutorHelper.java:48)
    at org.apache.hadoop.util.concurrent.HadoopThreadPoolExecutor.afterExecute(HadoopThreadPoolExecutor.java:90)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1157)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
    at java.lang.Thread.run(Thread.java:750)
STATE  total_deaths
LOUISIANA 1154
ALABAMA 1132
TEXAS 915
MISSISSIPPI 862
MISSOURI 744
TENNESSEE 636
ARKANSAS 580
FLORIDA 564
OKLAHOMA 551
INDIANA 415
Time taken: 8.502 seconds, Fetched 10 row(s)
spark-sql> []
```

*Figure 10.4.* Spark SQL query requesting a list of the 10 U.S. states with the highest number of deaths caused by severe storms; organized in descending order according to the number of deaths per state. Screenshot taken by Isabella Pham.

YEAR	MONTH NAME	event_count	total_injuries	total_deaths	total_property_damage	total_crops_damage
2008	June	11769	377	56	2099460900	25697500
2012	July	10604	226	46	118776500	1985000
1998	June	10500	1000	18	757918920	79682600
2011	June	10300	586	28	644536300	87364000
2009	June	9950	506	40	7000000	30360000
2011	April	9752	7524	788	12171912698	30360000
2009	June	8994	140	18	171323800	5454000
2008	July	8265	132	12	68841900	15450000
2003	July	8164	150	22	501927000	98854600
2011	July	8000	88	14	141183800	94772000
2005	June	7954	396	6	5700000	134000000
2013	June	7935	72	14	920329200	13229100
2004	May	7156	428	8	674140000	10820500
2010	July	6944	114	8	167072500	15925400
2006	July	6746	214	8	439970980	7108000
2009	August	6582	152	16	141183800	4140000
2007	June	6659	86	6	160526500	3283000
2011	May	6580	3350	378	653446300	3934000
2023	July	6529	62	9	433638820	16768410
2012	June	6398	114	36	342587200	14062800
2011	August	6326	198	32	256113900	177394400
1999	May	6300	1344	42	5300000	8700000
2005	July	6180	74	18	3731102800	10857000
2004	June	5912	144	10	137095500	9956000
2004	July	5680	152	8	132443000	21315000
2002	July	5666	114	6	202332180	74048000
2016	July	5609	96	12	59816480	1806000
1993	July	5572	236	30	5330000	8703400
2003	May	5478	1410	84	1912665000	21338000
2001	June	5468	202	10	252522530	35735000
2010	August	5460	278	16	84967300	5214600
2009	July	5376	124	10	164513500	21892000
1997	July	5309	508	34	5000000	2300000
2024	May	5302	351	50	600342149	5055000
2008	May	5274	1422	96	1620470800	15470000
2013	July	5176	66	6	78688600	28404000
2000	July	5096	112	14	135732040	8216000
2013	May	5074	501	90	4832234300	11593500
2004	July	4970	156	0	1300000	1000000
2000	May	4820	248	18	230975480	8244200
2006	June	4786	62	4	76845420	2288000
2023	June	4781	197	15	266463950	17251300
2007	July	4700	46	10	87294300	325006200
2009	August	4662	196	6	16049200	7879000
2000	August	4624	152	6	15212000	2000000
2019	June	4584	66	10	42169920	89400

Figure 10.5. Spark SQL query requesting a list of 100 dates (by year and month) in which the most storm events occurred, including associated data relating to the number of injuries, deaths, (estimated) value of property damages, and (estimated) value of crop damages. All crop and property damages are measured in USD. Screenshot taken by Isabella Pham. (Not Pictured: total elapsed time 9.264 seconds)

YEAR	event_count	total_deaths	total_injuries	total_property_damage	total_crop_damage
2015	19332	104	1238	639578768	24295600
2024	24441	147	924	7451362769	1558027410

Figure 10.6. Spark SQL query comparing summary data from two years roughly a decade apart (2024 and 2015), starting with data from 2015. All (estimated) values of property and crop damage are measured in USD. Screenshot taken by Isabella Pham.

Similar to our prior applications of HiveQL, Spark SQL was used to parse, transform, and summarize large quantities of structured data relating to severe storm events documented within the United States in a highly efficient manner. Specific focus was placed on data measuring impact zones and the lingering, measurable effects/damages caused by severe storms, such as fatalities, injuries, estimated property damage, and estimated crop damages. Queried data is returned in a simple format which can be easily be applied to future visualization and data analysis efforts, and has the potential to be used to train machine learning algorithms, which can be used to predict the potential after effects of future storms.

## Hive & Spark Comparisons

### Static Dataset

Query #	Time Elapsed	
	Apache Hive	Apache Spark
1	24.554 seconds	23.685 seconds
2	17.004 seconds	26.389 seconds
3	18.463 seconds	18.170 seconds
4	17.229 seconds	8.502 seconds
5	18.571 seconds	9.264 seconds
6	22.495 seconds	9.671 seconds
Average Total Time	<i>19.72 seconds</i>	<i>15.95 seconds</i>

### Streaming Dataset

Query #	Time Elapsed	
	Apache Hive	Apache Spark
1	7.005 seconds	8.952 seconds
2	8.133 seconds	8.673 seconds
3	8.094 seconds	2.248 seconds
4	7.868 seconds	7.775 seconds
5	7.128 seconds	5.788 seconds
Average Total	<i>7.65 seconds</i>	<i>6.69 seconds</i>

Time		
------	--	--

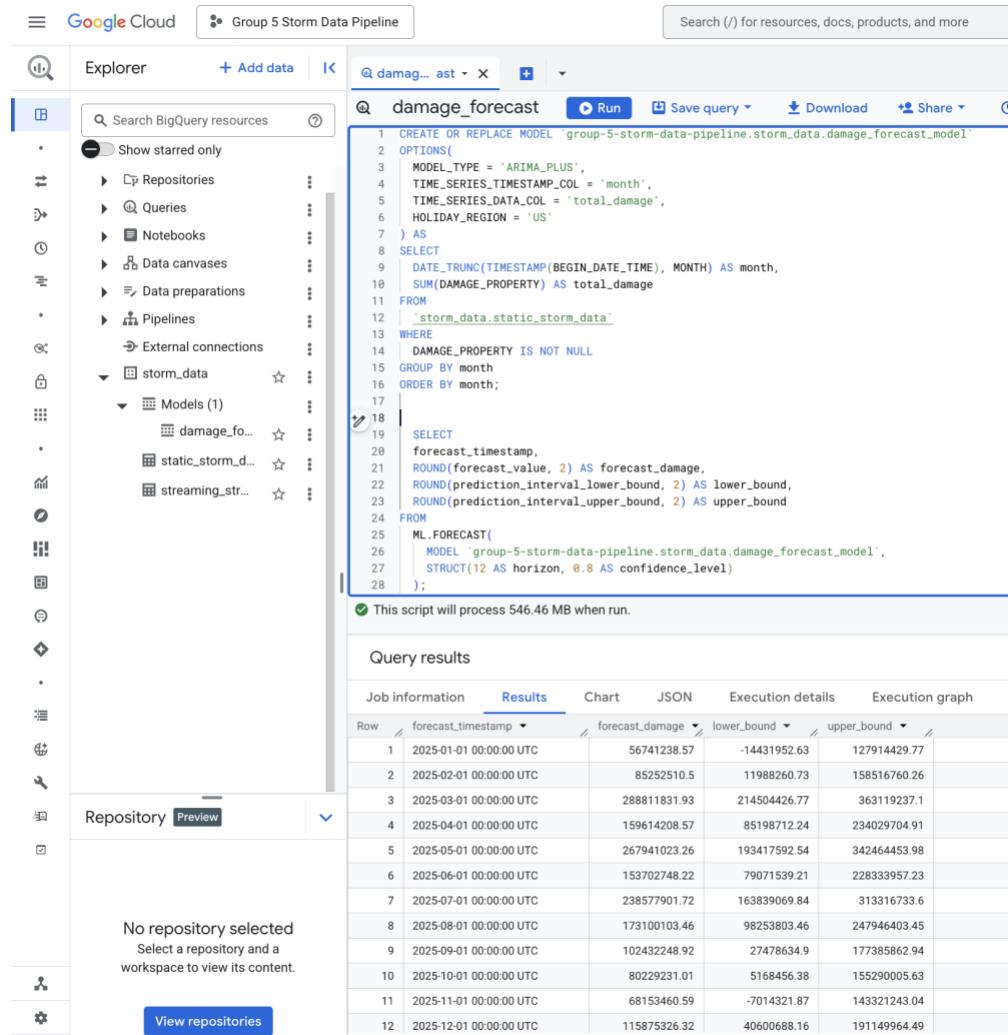
### ***Comparison Conclusion***

Apache Spark was consistently faster than Apache Hive when evaluating both static and streaming datasets, with a total average processing time of 11.74 seconds for Spark and 14.23 seconds for Hive. This is likely because of the differences in core functionalities between the two Hadoop sub-modules: While Hive is a scalable data warehousing solution that stores data on a computer's data storage device (i.e. its disk) - similar to how data is stored in a relational database management system - and processes each query by reading from and writing to the storage device, Spark is a framework with no dedicated storage; instead, it loads data into memory (RAM), processes everything in-memory, and only writes the final results to the computer's storage. (Berman, 2019). Since Hive is dependent on slower disk-based operations, while Spark processes data directly into memory through Spark, Spark achieves faster processing speed as compared to Hive.

Overall, the static and streaming queries were constructed using similar aggregations and groupings. However, static data queries resulted in longer query times due to the sheer size of the static dataset. The smaller size of the streaming dataset - combined with the computing power of Spark - resulted in Spark surpassing Hive entirely in terms of processing times. Based on these findings, we have concluded that the architecture of Spark made it better suited for both real-time and batch processing in this project.

## Machine Learning (BONUS)

### *Damage Forecast For the Year 2025*



The screenshot shows the Google Cloud BigQuery interface. The left sidebar lists various Google Cloud services like Google Cloud Storage, Google Sheets, and Google BigTable. The main area shows a query named "damage\_forecast" in the "storm\_data" dataset. The query code is as follows:

```

1 CREATE OR REPLACE MODEL `group-5-storm-data-pipeline.storm_data.damage_forecast_model`
2 OPTIONS(
3   MODEL_TYPE = 'ARIMA_PLUS',
4   TIME_SERIES_TIMESTAMP_COL = 'month',
5   TIME_SERIES_DATA_COL = 'total_damage',
6   HOLIDAY_REGION = 'US'
7 ) AS
8 SELECT
9   DATE_TRUNC(TIMESTAMP(BEGIN_DATE_TIME), MONTH) AS month,
10  SUM(DAMAGE_PROPERTY) AS total_damage
11 FROM
12  `storm_data.static_storm_data`
13 WHERE
14  DAMAGE_PROPERTY IS NOT NULL
15 GROUP BY month
16 ORDER BY month;
17
18
19 SELECT
20   forecast_timestamp,
21   ROUND(forecast_value, 2) AS forecast_damage,
22   ROUND(prediction_interval_lower_bound, 2) AS lower_bound,
23   ROUND(prediction_interval_upper_bound, 2) AS upper_bound
24 FROM
25  ML.PREDICT(
26    MODEL `group-5-storm-data-pipeline.storm_data.damage_forecast_model`,
27    STRUCT(12 AS horizon, 0.8 AS confidence_level)
28 );

```

A note at the bottom of the query editor says: "This script will process 546.46 MB when run."

Below the query editor, there's a "Query results" section. It has tabs for "Job information", "Results" (which is selected), "Chart", "JSON", "Execution details", and "Execution graph". The "Results" tab displays a table with 12 rows of data:

Row	forecast_timestamp	forecast_damage	lower_bound	upper_bound
1	2025-01-01 00:00:00 UTC	56741238.57	-14431952.63	127914429.77
2	2025-02-01 00:00:00 UTC	85252510.5	11988260.73	158516760.26
3	2025-03-01 00:00:00 UTC	288811831.93	214504426.77	363119237.1
4	2025-04-01 00:00:00 UTC	159614208.57	85198712.24	234029704.91
5	2025-05-01 00:00:00 UTC	267941023.26	193417592.54	342464453.98
6	2025-06-01 00:00:00 UTC	153702748.22	79071539.21	228333957.23
7	2025-07-01 00:00:00 UTC	238577901.72	163839069.84	313316733.6
8	2025-08-01 00:00:00 UTC	173100103.46	98253803.46	247946403.45
9	2025-09-01 00:00:00 UTC	102432248.92	27478634.9	177385862.94
10	2025-10-01 00:00:00 UTC	80229231.01	5168456.38	155290005.63
11	2025-11-01 00:00:00 UTC	68153460.59	-7014321.87	143321243.04
12	2025-12-01 00:00:00 UTC	115875326.32	40600688.16	191149964.49

The interface also includes a "Repository" section with a "View repositories" button and a note: "No repository selected Select a repository and a workspace to view its content."

Figure 11. Screenshot taken by Cezane Karki.

We used Google Big Query to train the machine learning model to forecast damage that might occur in the year 2025 by seeing the damage pattern from the year 1950 to present. This tool helps to provide the future information about the damages and potential upper and lower bound of the damages which can be useful for planning and preparation for the potential storm impact. Similar to this, looking at the patterns of the storm and its occurrence time, potential forecasts can be done for the storm type, impact and size that might occur in the future.

## Visualizations (BONUS)

### Static Dataset

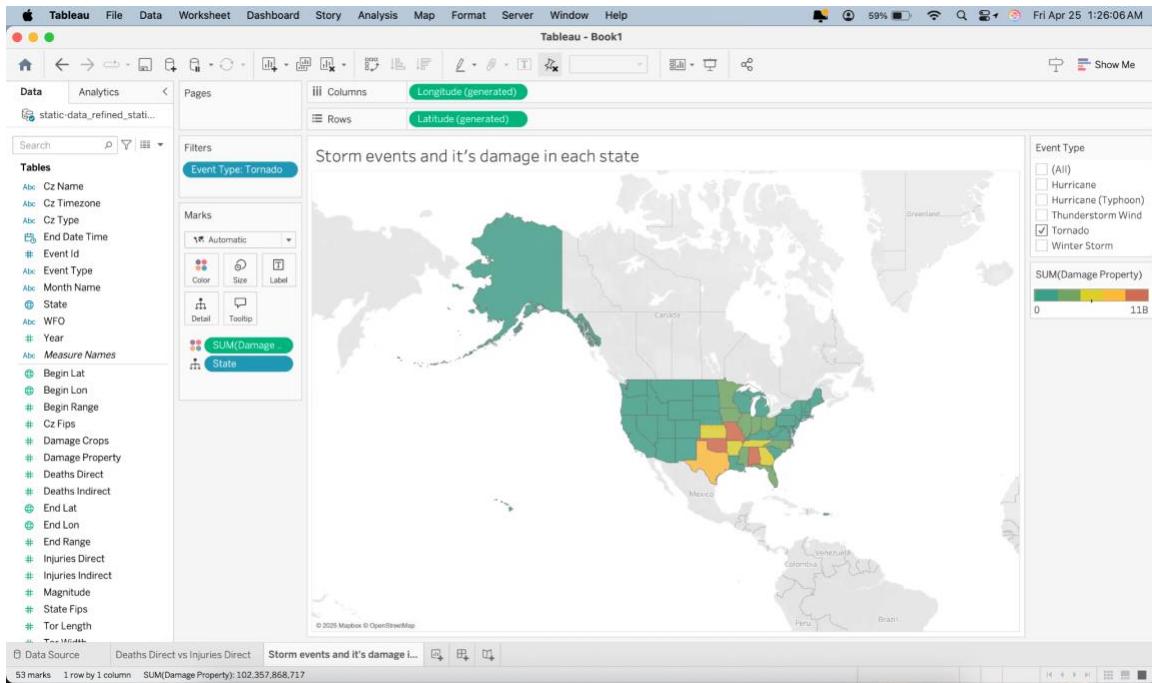


Figure 12.1. Screenshot taken by Cezane Karki.

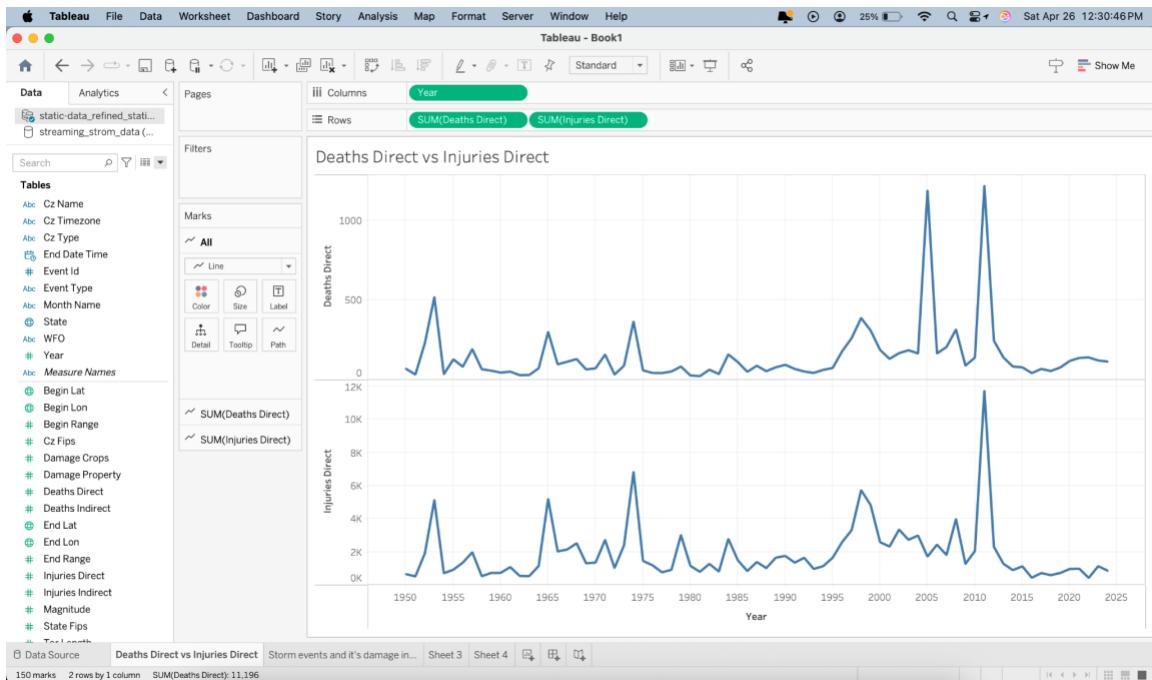


Figure 12.2. Screenshot taken by Cezane Karki.

## Streaming Dataset

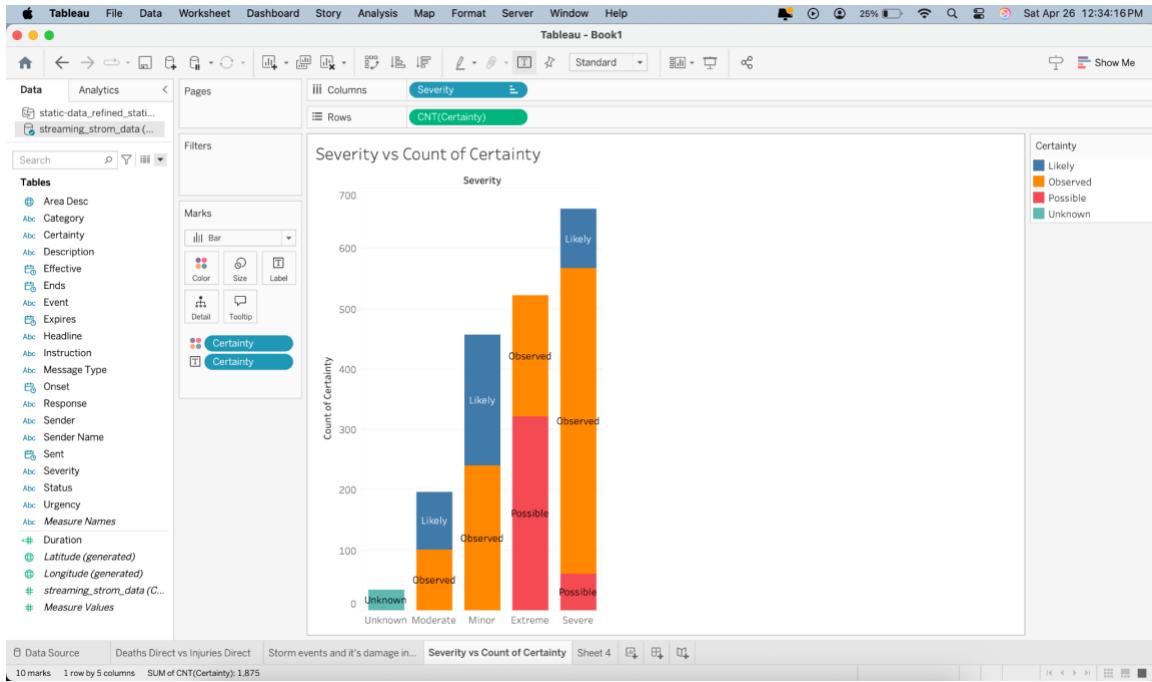


Figure 13.1. Screenshot taken by Cezane Karki.

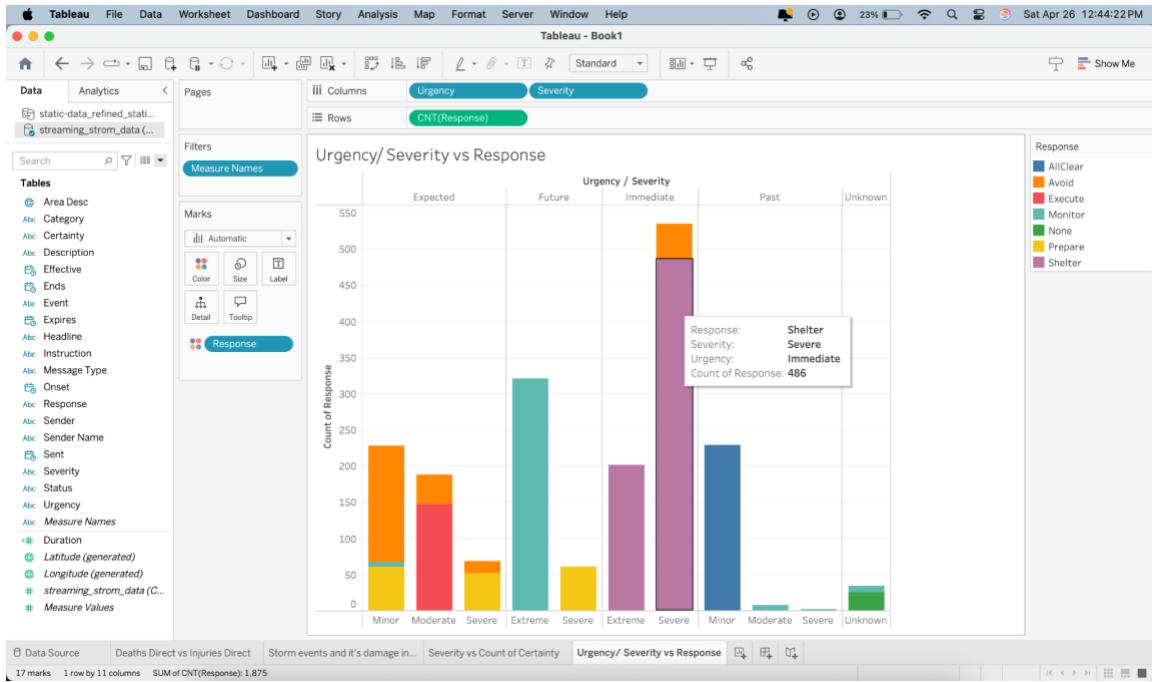


Figure 13.2. Screenshot taken by Cezane Karki.

## Interpretation (BONUS)

Our interpretation of storm event data processed through Big Query, Apache Hive, and Apache Spark has thrown up several actionable recommendations essential to the disaster response efforts of the American Red Cross. The static data analysis (from 1950 to 2024) realistically identified states most frequently impacted by storms. Texas, Kansas, and Oklahoma have the highest recorded number of impacts, of storm events, deaths, and crop damages, and property damages. (Figure 6.5 & 12.1) Some storm types such as tornadoes and thunderstorms contributed significantly to injuries and deaths over the decades, highlighting the vulnerabilities in different regions.

The streaming data acquired on April 20, 2025 indicated a distinct difference in the urgency and severity labeling of storm alerts across different parts of the United States. For example, high-urgency alerts have been given out most frequently by the states of Florida, Alabama, and Louisiana for tornadoes, severe thunderstorms, and flash floods. Word frequency analysis pointed to predominance of certain emergency communication words such as "Warning," "Severe," "Tornado," and "Flash Flood," suggesting the many hazards needing immediate attention by the public. Stream data analysis emphasized the need for better standardization and clarity in messaging practices, such that affected populations interpret correctly the level of threat and response.

The machine learning functionality and visualizations further enhanced our understanding and analysis of the data and its interpretation. Predictive damage forecasts with BigQuery machine learning functionality suggested 2025 might again trend upwards in property damage from previous years; this would be consistent with general predictions about climate change. The visualizations built from static data and the streaming data have further communicated trends like the frequency of different storm events, the severity distribution, and daily alert patterns clearly. These machine learning and visualization tools have enabled us to better present complex relationships in the data and make insights into actions for the American Red Cross.

Overall, the interpretation of our findings will help the American Red Cross in identifying the vulnerable areas, improving the channels of emergency communication, and initiating future more predictive, data-driven efforts at disaster response.

## Conclusion

### Discussion

In this project we utilized the data life cycle approach to derive a complete picture of the nature of the storm phenomenon. Using a systematic approach that included the generation, collection, processing, storage, management, and analysis of data, it revealed meaningful patterns in storms and created a strong framework to support real-time decision-making for disaster relief by the American Red Cross.

The joining of static data from 1950 to 2024 with streaming data gathered on April 20, 2025, facilitated comparative analysis between historical perspectives and real-time data, further elevating the validity and applicability of the findings. The use of tools such as OpenRefine, Apache Spark, Hive, and Google BigQuery was essential in effectively cleaning the data, transforming it, and conducting in-depth analytical processing. The coordination between classical SQL-based analysis and modern big data processing software was key in handling the magnitude and frequency of the storm data (Hashem et al., 2015).

The BigQuery analysis also yielded major trends, such as the number and kind of storms, the level of urgency, and the changing patterns of response over time. By classifying alert data in severity and geographic location categories, the areas having most at-risk of severe weather events were clear. Further, the ability of Apache Hive and Spark to derive insights from vast data make it better suited for making faster decisions. Other similar projects have used the tools of NLP to extract insights from real-time data such as social media and official announcements in crisis (Imran et al., 2014).

One of the most evident omissions to emerge in the analysis was the lack of tropical cyclone data, something that could compromise the accuracy of the forecasts in areas hit hardest by such storms. Another challenge we faced while completing this project was the dataset. Most of the columns of the dataset required to be altered to be suitable for the tools that we used in the processing stage. Also, the size of the dataset created hurdles with OpenRefine, as it interfered with the performance of the tool and its response while working. Hence, we had to spend additional time on planning of data management before initiating the actual process.

To conclude, by merging historical and live storm data using high-performance tools like BigQuery, Apache Hive, and Apache Spark, this project offered an end-to-end analytical system

to better prepare and respond for the American Red Cross. The enhanced visuals, derived from static and streaming data, compellingly illustrated significant patterns, like storm type occurrences, intensity trends, and emergency message development, that would remain elusive from raw data. These graphical tools summarized complex datasets to insightful information, proving that visuals are an important element to enhance public communications strategies and detection of at-risk areas. Furthermore, the Spark versus Hive comparison unearthed Spark's competitiveness regarding processing time, crucial for up-to-the-minute analysis for alerts. Having said that, Hive would be a better choice from an economic viewpoint as the processing time difference between the two services was minor compared to the size of the dataset. By making big data for storm events accessible and understandable, this project makes it easier for future data scientists and analysts to develop more predictive, concise, and user-friendly disaster management for the Red Cross organization.

## **Recommendations**

### ***1. Forecasting Storm Effects Using Predictive Modeling***

Based on an in-depth analysis of historical storm data from 1950 to 2024, as well as real-time data from April 2025, the American Red Cross should establish a partnership with data scientists to develop machine learning models for the prediction of the types of storms, their severity, and response needs. The patterns observed in storm frequencies, seasonal patterns, and severity distributions suggest potential for predictive analytics for management of storms. With the use of methods such as deep learning and time series forecasting, early warning systems can be improved, opening the door to proactive emergency planning (Li, Li, et al., 2020).

**Expected Impact:** Improves disaster anticipation and resource readiness, reducing emergency response times.

### ***2. Geospatial Risk Mapping for Resource Deployment***

Visualizations based on static and streaming data show geographic clusters characterized by high storm intensity, high rates of injury, and high property losses. These observations can help the Red Cross prioritize resource allocation by generating dynamic heatmaps and clustering models that are updated in real time (Hashem et al., 2015). Pre-deployment of critical supplies, such as emergency shelters, medical supplies, and rescue personnel, to high-risk areas, can help improve the effectiveness of response activities.

**Expected Impact:** Enhances resource effectiveness and increases community resilience in high-risk areas.

### ***3. Real-Time, Automated Alert Systems***

Since Spark and BigQuery have shown success handling streaming alert data, the Red Cross should create a real-time monitoring dashboard. This dashboard would systematically filter alerts according to severity, urgency, and response type, supplemented with visualization aids like graphical frequency trends and urgency heatmaps (Imran, et al., 2014). It would enable the decision-making processes of field officers to be faster, data-driven, enabling the latter to prepare societies proactively for approaching storms.

**Expected Impact:** It provides real-time situational awareness, enabling faster, more accurate responses as disasters occur.

### ***4. Optimization of Public Messaging***

Analysis of often repeated key terms "Warning," "Severe," and "Tornado" from streaming alerts holds great promise for improving public communication. The Red Cross needs to concentrate efforts towards creating standardized alert messages that utilize explicit and imperative templates. Through testing the templates with groups of people of different backgrounds, we can provide assurance that messages get quickly understood and evince appropriate public actions, thereby reducing fatalities as well as misunderstandings of disasters (Imran et al., 2014).

**Expected Impact:** Increases public awareness and compliance in response to emergency notifications, thus creating better safety outcomes.

## Supporting Documentation

### Data Sheet

We used two data sets - one static and one streaming - of data for the analysis of severe storms based in the United States.

The static dataset, [Storm Events Database](#), was obtained from the National Oceanic and Atmospheric Administration's (NOAA). The dataset contains data from 1950 until 2024, and focuses on the location, storm type/severity, and impact (financial and otherwise) of each documented storm. We chose this dataset because it is accessible (i.e. free for public use), the data are both formatted consistently and despite spanning several decades, the quality of each observation is generally reliable.

The streaming data was obtained from the [National Weather Service \(NWS\) API](#). The live data was collected through the API on the 20th of April, 2025 at different times. Collected data contained information about the severity, onset, and offset of recorded storms. We chose this data because we believe that its use can help the Red Cross to predict severe storms and prepare affected communities ahead of time, as to minimize damages and casualties. The data is also current (i.e. up-to-date), timely, and - similarly to the streaming dataset obtained from NOAA - publicly accessible.

### Meeting Notes

#### *Meeting 1*

<b>Meeting Information</b>	<p><b>Date:</b> April 15, 2025  <b>Times:</b> <ul style="list-style-type: none"> <li>• Start: 1:40 PM</li> <li>• End: 3:52 PM</li> </ul> <b>Attendees:</b> <ul style="list-style-type: none"> <li>• Shivangi Vipulbhai Borad</li> <li>• Isabella Pham</li> <li>• Kiran Kuchekar</li> <li>• Ankur Bhattacharjee</li> <li>• Cezane Karki</li> </ul> <b>Note-Taker:</b> <ul style="list-style-type: none"> <li>• Isabella Pham</li> </ul> </p>
<b>Notes</b>	<ul style="list-style-type: none"> <li>• Bimala wasn't able to meet due to a personal emergency, but was kept updated via WhatsApp message and assisted with searching for datasets</li> <li>• In order to facilitate collaborative work, the following were created and shared among the group: <ul style="list-style-type: none"> <li>◦ Shared Google Drive folder</li> <li>◦ Google Document</li> <li>◦ Google Slides presentation</li> <li>◦ Jira dashboard</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>Isabella updated and shared project overview notes (taken during class) with the group</li> <li>Discussed group availability for the week and scheduled the next meeting (April 18)</li> </ul> <p><b>DATASET RESEARCH/DISCUSSION</b></p> <ul style="list-style-type: none"> <li>Discussed and researched potential datasets <ul style="list-style-type: none"> <li>Group ran into issues with certain datasets being unavailable, with datasets being of inadequate size/quality, and with datasets containing useful but not completely relevant information</li> </ul> </li> <li>Located a <u>static dataset</u> on U.S.-based storm events from the National Centers for Environmental Information <ul style="list-style-type: none"> <li>Discussed quality of dataset, and agreed that it was high-quality (easily-accessible, consistent, accurate, and current)</li> <li>Discovered that the dataset contained a wide range of data, <i>except</i> for enough data on tropical cyclones</li> </ul> </li> <li>Agreed that using the above dataset would be ideal if our use case/prompt included other types of storm events</li> </ul> <p><b>PLANNING</b></p> <ul style="list-style-type: none"> <li>Discussed (but didn't fully solidify): <ul style="list-style-type: none"> <li>An outline for our plan of action</li> <li>How work might be divided</li> </ul> </li> <li>Wrote an email to Dr. Schenita Floyd regarding the following: <ul style="list-style-type: none"> <li>Potentially expanding the scope of our use case/prompt to include other types of storm events (e.g. tornadoes, hurricanes)</li> <li>Document format guidelines</li> </ul> </li> </ul>																																								
<b>Decisions</b>	<ul style="list-style-type: none"> <li><b>Static Dataset:</b> <a href="#">Storm Events Database</a> <ul style="list-style-type: none"> <li>Contains storm event information across the U.S. (from 1950 to 2024)</li> <li>Data will be filtered to only include tropical storms, hurricanes, and tornados</li> </ul> </li> <li><b>Streaming Dataset:</b> No decision made yet <ul style="list-style-type: none"> <li>Cezane and Ankur agreed to take shared responsibility for setting up APIs</li> </ul> </li> <li><b>Use Case:</b> Group agreed to expand our use case/prompt to include other storm events (with Dr. Floyd's permission)</li> <li><b>Next Meeting:</b> <ul style="list-style-type: none"> <li>April 18, 2025 - Willis Library</li> <li>Approximate meeting time: 1:00 PM - 5:00 PM</li> </ul> </li> </ul>																																								
<b>Action Items</b>	<table border="1"> <thead> <tr> <th><i>Item #</i></th><th><i>Description</i></th><th><i>Assigned To</i></th><th><i>Due Date</i></th><th><i>Status</i></th></tr> </thead> <tbody> <tr> <td>1</td><td>Email Dr. Floyd</td><td>Isabella Pham, Kiran Kuchekar, Shivangi Borad</td><td>04/15/2025</td><td>Completed</td></tr> <tr> <td>2</td><td>Search for datasets</td><td>Whole Group</td><td>04/18/2025</td><td>Ongoing</td></tr> <tr> <td>3</td><td>Thoroughly check static dataset for quality + compile data</td><td>Cezane Karki</td><td>04/18/2025</td><td>Ongoing</td></tr> <tr> <td>4</td><td>Identify an appropriate streaming dataset for use</td><td>Cezane Karki, Ankur Bhattacharjee</td><td>04/18/2025</td><td>Ongoing</td></tr> <tr> <td>5</td><td>Choose a non-profit organization and determine why we are gathering data for them</td><td>Whole Group</td><td>04/18/2025</td><td>Ongoing</td></tr> <tr> <td>6</td><td>Create GCP project and assign access permissions</td><td>Kiran S. Kuchekar</td><td>04/18/2025</td><td>Not Started</td></tr> <tr> <td>7</td><td>Create GCP bucket &amp; upload raw static data</td><td>Kiran S. Kuchekar</td><td>04/18/2025</td><td>Not Started</td></tr> </tbody> </table>	<i>Item #</i>	<i>Description</i>	<i>Assigned To</i>	<i>Due Date</i>	<i>Status</i>	1	Email Dr. Floyd	Isabella Pham, Kiran Kuchekar, Shivangi Borad	04/15/2025	Completed	2	Search for datasets	Whole Group	04/18/2025	Ongoing	3	Thoroughly check static dataset for quality + compile data	Cezane Karki	04/18/2025	Ongoing	4	Identify an appropriate streaming dataset for use	Cezane Karki, Ankur Bhattacharjee	04/18/2025	Ongoing	5	Choose a non-profit organization and determine why we are gathering data for them	Whole Group	04/18/2025	Ongoing	6	Create GCP project and assign access permissions	Kiran S. Kuchekar	04/18/2025	Not Started	7	Create GCP bucket & upload raw static data	Kiran S. Kuchekar	04/18/2025	Not Started
<i>Item #</i>	<i>Description</i>	<i>Assigned To</i>	<i>Due Date</i>	<i>Status</i>																																					
1	Email Dr. Floyd	Isabella Pham, Kiran Kuchekar, Shivangi Borad	04/15/2025	Completed																																					
2	Search for datasets	Whole Group	04/18/2025	Ongoing																																					
3	Thoroughly check static dataset for quality + compile data	Cezane Karki	04/18/2025	Ongoing																																					
4	Identify an appropriate streaming dataset for use	Cezane Karki, Ankur Bhattacharjee	04/18/2025	Ongoing																																					
5	Choose a non-profit organization and determine why we are gathering data for them	Whole Group	04/18/2025	Ongoing																																					
6	Create GCP project and assign access permissions	Kiran S. Kuchekar	04/18/2025	Not Started																																					
7	Create GCP bucket & upload raw static data	Kiran S. Kuchekar	04/18/2025	Not Started																																					

	8	Format presentation document and create presentation outline	Isabella Pham	04/22/2025	Not Started
--	---	--	---------------	------------	-------------

## Meeting 2

<b>Meeting Information</b>	<p><b>Date:</b> April 18, 2025  <b>Times:</b> <ul style="list-style-type: none"> <li>• Start: 2:14 PM</li> <li>• End: 4:33 PM</li> </ul> <b>Attendees:</b> <ul style="list-style-type: none"> <li>• Shivangi Vipulbhai Borad</li> <li>• Isabella Pham</li> <li>• Kiran Kuchekar</li> <li>• Bimala Joshi</li> <li>• Cezane Karki</li> <li>• Ankur Bhattacharjee</li> </ul> <b>Note-Taker:</b> <ul style="list-style-type: none"> <li>• Isabella Pham</li> </ul> </p>															
<b>Notes</b>	<ul style="list-style-type: none"> <li>• Added tasks and assigned tasks on the Jira project dashboard</li> <li>• Identified two potential streaming data sources and agreed to contact Dr. Floyd for approval: <ul style="list-style-type: none"> <li>◦ <a href="#">Weather.Gov (National Weather Service): API Web Service</a></li> <li>◦ <a href="#">WxData.com: Storm Reports API</a></li> </ul> </li> <li>• Created the following: <ul style="list-style-type: none"> <li>◦ GCP project (added all group members as principals)</li> <li>◦ GCP storage bucket</li> <li>◦ Directory structure within storage bucket (separate containers for raw and refined data for both streaming and static data sets)</li> </ul> </li> <li>• Combed through the static dataset &amp; discussed how to clean it (i.e. which columns to delete, which to keep, etc.)</li> <li>• Discussed presentation format/theme and outline</li> <li>• Discussed which non-profits we can use</li> <li>• Ran into issues with running data from the static dataset through OpenRefine <ul style="list-style-type: none"> <li>◦ Meeting ended without a clear solution</li> </ul> </li> </ul>															
<b>Decisions</b>	<ul style="list-style-type: none"> <li>• Non-Profit <ul style="list-style-type: none"> <li>◦ Chose the American Red Cross for their humanitarian and disaster relief efforts</li> <li>◦ Determined that - within the scope of the project prompt - we were hired to build a pipeline for data which can be used to predict storm events for more efficient and effective disaster aid</li> </ul> </li> <li>• Chose two potential streaming datasets (pending approval from Dr. Floyd)</li> <li>• Scheduled next meetings: <ul style="list-style-type: none"> <li>◦ Meeting 3: <ul style="list-style-type: none"> <li>■ April 22 (Tuesday)</li> <li>■ 12:00 PM - Undetermined End Time</li> <li>■ Curry Hall, Rom 204</li> </ul> </li> <li>◦ Meeting 4: <ul style="list-style-type: none"> <li>■ April 25 (Friday)</li> <li>■ 2:30 PM - 4:30 PM</li> <li>■ Willis Library, Room 250G</li> </ul> </li> </ul> </li> </ul>															
<b>Action Items</b>	<table border="1"> <thead> <tr> <th><b>Item #</b></th> <th><b>Description</b></th> <th><b>Assigned To</b></th> <th><b>Due Date</b></th> <th><b>Status</b></th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Thoroughly check static dataset for quality + compile data</td> <td>Cezane Karki, Ankur Bhattacharjee</td> <td>04/18/2025</td> <td>Completed</td> </tr> <tr> <td>2</td> <td>Email Dr. Floyd</td> <td>Isabella Pham</td> <td>04/18/2025</td> <td>Completed</td> </tr> </tbody> </table>	<b>Item #</b>	<b>Description</b>	<b>Assigned To</b>	<b>Due Date</b>	<b>Status</b>	1	Thoroughly check static dataset for quality + compile data	Cezane Karki, Ankur Bhattacharjee	04/18/2025	Completed	2	Email Dr. Floyd	Isabella Pham	04/18/2025	Completed
<b>Item #</b>	<b>Description</b>	<b>Assigned To</b>	<b>Due Date</b>	<b>Status</b>												
1	Thoroughly check static dataset for quality + compile data	Cezane Karki, Ankur Bhattacharjee	04/18/2025	Completed												
2	Email Dr. Floyd	Isabella Pham	04/18/2025	Completed												

	3	Identify an appropriate streaming dataset for use	Cezane Karki, Ankur Bhattacharjee	04/18/2025	Completed
	4	Thoroughly check static dataset for quality + compile data	Cezane Karki	04/18/2025	Completed
	5	Choose a non-profit organization and determine why we are gathering data for them	Whole Group	04/18/2025	Completed
	6	Create GCP project and assign access permissions	Kiran S. Kuchekar	04/18/2025	Completed
	7	Create GCP bucket & upload raw static data	Kiran S. Kuchekar	04/18/2025	Completed
	8	Format presentation document and create presentation outline	Isabella Pham	04/22/2025	Ongoing
	9	Clean/refine data from static data source using OpenRefine	Ankur Bhattacharjee	04/22/2025	Ongoing
	10	Clean/refine data from streaming data source using OpenRefine	Ankur Bhattacharjee	04/22/2025	Not Started
	11	Create cluster using GCP Dataproc	Bimala Joshi	04/22/2025	Not Started
	12	Write SQL queries in GCP BigQuery for the static dataset	Cezane Karki	04/22/2025	Not Started
	13	Write SQL queries in GCP BigQuery for the streaming dataset	Cezane Karki	04/22/2025	Not Started

### Meeting 3

<b>Meeting Information</b>	<p><b>Date:</b> April 22, 2025  <b>Times:</b> <ul style="list-style-type: none"> <li>• Start: 12:08 PM</li> <li>• End: 8:07 PM</li> </ul> <b>Attendees:</b> <ul style="list-style-type: none"> <li>• Shivangi Vipulbhai Borad</li> <li>• Isabella Pham</li> <li>• Kiran Kuchekar</li> <li>• Bimala Joshi</li> <li>• Cezane Karki</li> <li>• Ankur Bhattacharjee</li> </ul> <b>Note-Taker:</b> <ul style="list-style-type: none"> <li>• Isabella Pham</li> </ul> </p>
<b>Notes</b>	<ul style="list-style-type: none"> <li>• Solved previous issues with OpenRefine &amp; finished cleaning both static and streaming data</li> <li>• Created presentation outline, began writing speaker notes, &amp; completed multiple slides</li> <li>• Uploaded refined streaming and static data to GCP bucket</li> <li>• Created cluster using GCP Dataproc + VM instances (1 manager node, 2 worker nodes)</li> <li>• Wrote multiple queries using GCP BigQuery for both streaming and static data</li> <li>• Copied the refined versions of both streaming and static datasets to HDFS and created tables for each</li> <li>• Wrote and sent queries for the streaming data table using Spark SQL</li> </ul>

	<ul style="list-style-type: none"> <li>Began writing and sending queries for the static data table using HiveQL</li> </ul>				
<b>Decisions</b>	<ul style="list-style-type: none"> <li>N/A</li> </ul>				
<b>Action Items</b>	<i>Item #</i>	<i>Description</i>	<i>Assigned To</i>	<i>Due Date</i>	<i>Status</i>
	1	Format presentation document and create presentation outline	Isabella Pham, Shivangi Vipulbhai Borad	04/22/2025	Completed
	2	Clean/refine static data using OpenRefine	Ankur Bhattacharjee	04/22/2025	Completed
	3	Set up APIs for streaming data	Cezane Karki, Ankur Bhattacharjee	04/22/2025	Completed
	4	Clean/refine streaming data using OpenRefine	Ankur Bhattacharjee	04/22/2025	Completed
	5	Create cluster using GCP Dataproc	Bimala Joshi	04/22/2025	Completed
	6	Write SQL queries in GCP BigQuery for the static dataset	Cezane Karki	04/22/2025	Completed
	7	Write SQL queries in GCP BigQuery for the streaming dataset	Cezane Karki	04/22/2025	Completed
	8	Copy static data from GCP bucket to HDFS; create table	Isabella Pham	04/22/2025	Completed
	9	Copy streaming data from GCP bucket to HDFS; create table	Shivangi Vipulbhai Borad	04/22/2025	Completed
	10	Query data using Spark SQL queries	Shivangi Vipulbhai Borad	04/22/2025	Completed
	11	Query data using HiveQL queries	Isabella Pham	04/25/2025	Ongoing
	12	Update document outline & fill in documentation	Bimala Joshi, Group	04/25/2025	Ongoing
	13	Format, revise, and edit document before submission	Isabella Pham, Group	04/28/2025	Not Started

## Meeting 4

<b>Meeting Information</b>	<u>Date:</u> April 26, 2025 <u>Times:</u> <ul style="list-style-type: none"> <li>Start: 9:00 PM</li> <li>End: 10:40 PM</li> </ul> <u>Attendees:</u> <ul style="list-style-type: none"> <li>Shivangi Vipulbhai Borad</li> <li>Isabella Pham</li> </ul>
----------------------------	--

	<ul style="list-style-type: none"> <li>• Kiran Kuchekar</li> <li>• Bimala Joshi</li> <li>• Cezane Karki</li> <li>• Ankur Bhattacharjee</li> </ul> <p><b>Note-Taker:</b></p> <ul style="list-style-type: none"> <li>• Isabella Pham</li> </ul>																																																																	
<b>Notes</b>	<ul style="list-style-type: none"> <li>• Reviewed document to: <ul style="list-style-type: none"> <li>○ Update and change written content according to group consensus</li> <li>○ Ensure that all written content and formatting was both factual and correct</li> <li>○ Ensure that all group members are on the same page</li> </ul> </li> <li>• Updated document contents according to group discussion and agreement</li> <li>• Assigned presentation slides to group members</li> </ul>																																																																	
<b>Decisions</b>	<ul style="list-style-type: none"> <li>• Assigned presentation slides to group members</li> <li>• Agreed to meet at 9 PM on Monday evening to practice presenting</li> </ul>																																																																	
<b>Action Items</b>	<table border="1"> <thead> <tr> <th><i>Item #</i></th><th><i>Description</i></th><th><i>Assigned To</i></th><th><i>Due Date</i></th><th><i>Status</i></th></tr> </thead> <tbody> <tr> <td>1</td><td>Query data using HiveQL queries</td><td>Isabella Pham</td><td>04/25/2025</td><td>Completed</td></tr> <tr> <td>2</td><td>Update document outline &amp; fill in documentation</td><td>Bimala Joshi, Group</td><td>04/25/2025</td><td>Completed</td></tr> <tr> <td>3</td><td>Format, revise, and edit document before submission</td><td>Isabella Pham</td><td>04/28/2025</td><td>Completed</td></tr> <tr> <td>4</td><td>Complete comparison chart &amp; interpretation for Apache Hive and Spark</td><td>Isabella Pham, Shivangi Vipulbhai Borad</td><td>04/28/2025</td><td>Completed</td></tr> <tr> <td>5</td><td>Create table of contents</td><td>Isabella Pham</td><td>04/28/2025</td><td>Completed</td></tr> <tr> <td>6</td><td>Complete &amp; upload data visualizations</td><td>Cezane Karki</td><td>04/28/2025</td><td>Completed</td></tr> <tr> <td>7</td><td>Complete written content for the “Interpretation” section</td><td>Shivangi Vipulbhai Borad</td><td>04/28/2025</td><td>Completed</td></tr> <tr> <td>8</td><td>Complete writing discussion and actionable recommendations</td><td>Kiran S. Kuchekar</td><td>04/28/2025</td><td>Completed</td></tr> <tr> <td>9</td><td>Complete written content for the conclusion (discussion &amp; recommendations)</td><td>Kiran S. Kuchekar</td><td>04/28/2025</td><td>Ongoing</td></tr> <tr> <td>10</td><td>Complete data sheet</td><td>Bimala Joshi</td><td>04/28/2025</td><td>Completed</td></tr> <tr> <td>11</td><td>Update slides according to contents of the written document</td><td>Whole Group</td><td>04/28/2025</td><td>Ongoing</td></tr> <tr> <td>12</td><td>Assign presentation slides</td><td>Whole Group</td><td>04/27/2025</td><td>Completed</td></tr> </tbody> </table>	<i>Item #</i>	<i>Description</i>	<i>Assigned To</i>	<i>Due Date</i>	<i>Status</i>	1	Query data using HiveQL queries	Isabella Pham	04/25/2025	Completed	2	Update document outline & fill in documentation	Bimala Joshi, Group	04/25/2025	Completed	3	Format, revise, and edit document before submission	Isabella Pham	04/28/2025	Completed	4	Complete comparison chart & interpretation for Apache Hive and Spark	Isabella Pham, Shivangi Vipulbhai Borad	04/28/2025	Completed	5	Create table of contents	Isabella Pham	04/28/2025	Completed	6	Complete & upload data visualizations	Cezane Karki	04/28/2025	Completed	7	Complete written content for the “Interpretation” section	Shivangi Vipulbhai Borad	04/28/2025	Completed	8	Complete writing discussion and actionable recommendations	Kiran S. Kuchekar	04/28/2025	Completed	9	Complete written content for the conclusion (discussion & recommendations)	Kiran S. Kuchekar	04/28/2025	Ongoing	10	Complete data sheet	Bimala Joshi	04/28/2025	Completed	11	Update slides according to contents of the written document	Whole Group	04/28/2025	Ongoing	12	Assign presentation slides	Whole Group	04/27/2025	Completed
<i>Item #</i>	<i>Description</i>	<i>Assigned To</i>	<i>Due Date</i>	<i>Status</i>																																																														
1	Query data using HiveQL queries	Isabella Pham	04/25/2025	Completed																																																														
2	Update document outline & fill in documentation	Bimala Joshi, Group	04/25/2025	Completed																																																														
3	Format, revise, and edit document before submission	Isabella Pham	04/28/2025	Completed																																																														
4	Complete comparison chart & interpretation for Apache Hive and Spark	Isabella Pham, Shivangi Vipulbhai Borad	04/28/2025	Completed																																																														
5	Create table of contents	Isabella Pham	04/28/2025	Completed																																																														
6	Complete & upload data visualizations	Cezane Karki	04/28/2025	Completed																																																														
7	Complete written content for the “Interpretation” section	Shivangi Vipulbhai Borad	04/28/2025	Completed																																																														
8	Complete writing discussion and actionable recommendations	Kiran S. Kuchekar	04/28/2025	Completed																																																														
9	Complete written content for the conclusion (discussion & recommendations)	Kiran S. Kuchekar	04/28/2025	Ongoing																																																														
10	Complete data sheet	Bimala Joshi	04/28/2025	Completed																																																														
11	Update slides according to contents of the written document	Whole Group	04/28/2025	Ongoing																																																														
12	Assign presentation slides	Whole Group	04/27/2025	Completed																																																														

## Meeting 5

<b>Meeting Information</b>	<p><b>Date:</b> April 28, 2025</p> <p><b>Times:</b></p> <ul style="list-style-type: none"> <li>• Start: 9:00 PM</li> <li>• End: 10:09 PM</li> </ul> <p><b>Attendees:</b></p> <ul style="list-style-type: none"> <li>• Shivangi Vipulbhai Borad</li> <li>• Isabella Pham</li> <li>• Kiran Kuchekar</li> <li>• Bimala Joshi</li> <li>• Cezane Karki</li> <li>• Ankur Bhattacharjee</li> </ul> <p><b>Note-Taker:</b></p> <ul style="list-style-type: none"> <li>• Isabella Pham</li> </ul>					
<b>Notes</b>	<ul style="list-style-type: none"> <li>• Practiced presenting (with a specific focus on remaining within 8 minutes)</li> <li>• Reviewed/double-checked presentation and written document for content and any errors</li> </ul>					
<b>Decisions</b>	<ul style="list-style-type: none"> <li>• N/A</li> </ul>					
<b>Action Items</b>	<table border="1"> <thead> <tr> <th><i>Item #</i></th><th><i>Description</i></th><th><i>Assigned To</i></th><th><i>Due Date</i></th><th><i>Status</i></th></tr> </thead> </table>	<i>Item #</i>	<i>Description</i>	<i>Assigned To</i>	<i>Due Date</i>	<i>Status</i>
<i>Item #</i>	<i>Description</i>	<i>Assigned To</i>	<i>Due Date</i>	<i>Status</i>		
<table border="1"> <tbody> <tr> <td>1</td><td>Complete written content for the conclusion (discussion &amp; recommendations)</td><td>Kiran S. Kuchekar</td><td>04/28/2025</td><td>Completed</td></tr> </tbody> </table>	1	Complete written content for the conclusion (discussion & recommendations)	Kiran S. Kuchekar	04/28/2025	Completed	
1	Complete written content for the conclusion (discussion & recommendations)	Kiran S. Kuchekar	04/28/2025	Completed		
<table border="1"> <tbody> <tr> <td>2</td><td>Update slides according to contents of the written document</td><td>Whole Group</td><td>04/28/2025</td><td>Completed</td></tr> </tbody> </table>	2	Update slides according to contents of the written document	Whole Group	04/28/2025	Completed	
2	Update slides according to contents of the written document	Whole Group	04/28/2025	Completed		

## Access & Permission Policy

The screenshot shows the Google Cloud IAM (Identity and Access Management) interface. The left sidebar has a tree view with 'IAM & Admin' selected under 'IAM'. The main area is titled 'Allow' and lists users with their email addresses, roles, and permissions. The roles include Editor, BigQuery Admin, Owner, and All User Role. The permissions listed are IAP-secured Tunnel Destination Group Editor, IAP-secured Tunnel User, IAP-secured Web App User, and various Google services like Compute Engine, Storage, and BigQuery.

User Email	Role	Permissions
424037099608-compute@developer.gserviceaccount.com	Editor	Default compute service account
abcharjee17@gmail.com	Editor	IAP-secured Tunnel Destination Group Editor IAP-secured Tunnel User IAP-secured Web App User
bimalajoshi14@gmail.com	BigQuery Admin	IAP-secured Tunnel Destination Group Editor IAP-secured Tunnel User IAP-secured Web App User
isabellam.pham@gmail.com	All User Role	IAP-secured Tunnel Destination Group Editor IAP-secured Tunnel User IAP-secured Web App User
karkicezane123@gmail.com	Owner	Cezane Karki
kirankuchekar0@gmail.com	Editor	
project-datapipeline@group-5-storm-data-pipeline.iam.gserviceaccount.com	Owner	Project-datapipeline
sborad.us@gmail.com	Editor	IAP-secured Tunnel Destination Group Editor IAP-secured Tunnel User IAP-secured Web App User

Figure 14. IAM policy for all permitted users.

## Jira Dashboard (BONUS)

The screenshot shows a Jira dashboard for a project titled 'Harvesting, Retrieving, and Storing Data'. The left sidebar has sections for 'PLANNING', 'DEVELOPMENT', and 'ARCHIVED WORK ITEMS'. The main area displays a list of work items and a detailed architecture diagram.

**Work Items:**

- KAN-10: Clean the data for static data
- KAN-12: Clean the data using Open Refine
- KAN-11: Store streaming data in the streaming data folder
- KAN-10: Store static files in the static folder
- KAN-9: Create Bucket and Folder Structure
- KAN-6: Fetch data from the bucket and use Open refine to refine the data
- KAN-7: Fetch Data from the API and store in the bucket
- KAN-6: Fetch the data from API and store it in the bucket
- KAN-6: Create an outline for and format the presentation slides
- KAN-4: Search for the static data
- KAN-3: Deadly and damaging tropical cyclones Data Engineering Pipeline

**Architecture Diagram:**

```

graph TD
    subgraph Data_Ingestion [Data Ingestion]
        direction TB
        DSD[Read raw Weather Data Streams] --> CHF[Cloud Function: transform_weather_data]
        SVD[Static Vulnerability Data] --> CSB[Cloud Storage Bucket: static-vulnerability-data]
    end

    subgraph Data_Processing [Data Processing]
        direction TB
        CHF --> DP[Dataflow Job: process_weather_weather]
        CSB --> DP
        DP --> BQ1[BigQuery Table: processed_weather_data]
        DP --> BQ2[BigQuery Table: vulnerability_data]
    end

    subgraph Risk_Assessment [Risk Assessment]
        direction TB
        BQ1 --> RAI[Dataflow Job: integrate_risk_assessment]
        BQ2 --> RAI
        RAI --> BQ3[BigQuery Table: cyclone_risk_index]
    end

    subgraph Data_Visualization [Data Visualization & Alerting]
        direction TB
        BQ3 --> DV[Dataflow Job: visualize_risk_index]
        DV --> DVH[Dataflow Job: publish_risk_index]
        DVH --> CFAB[Cloud Function: Create Alerting Backend]
        CFAB --> EIR[Email Integration: Risk Index]
    end

```

**Architecture Description:** Proposed Architecture: The architecture involves data ingestion from raw weather data streams and static vulnerability data into a cloud storage bucket. This data is then processed using Dataflow jobs to produce processed weather data and vulnerability data. These datasets undergo risk assessment using another Dataflow job. The resulting cyclone risk index is visualized and published to a cloud function, which triggers an alerting backend and sends an email integration.

Figure 15.1 Jira dashboard (Project Epic).

The screenshot shows the Jira interface with the following details:

- Header:** Jira, Your work, Projects, Filters, Dashboards, Teams, Plans, Apps, Create.
- Project:** Harvesting, Retrieving, and Storing Data (Software project).
- View:** All work.
- Task List:**
  - Child work items:
 

Type	Key	Summary	Priority	Assignee	Status
Task	KAN-6	Fetch the data from API and store it in the bucket	Medium	Unassigned	DONE
Task	KAN-15	Create outline for executive summary	Medium	Isabella Ph...	DONE
Task	KAN-4	Search for the static data	Medium	Cezane Karki	DONE
Task	KAN-1	Search for the streaming data and its API and API documentation	Medium	Cezane Karki	DONE
Task	KAN-2	Create a bucket in GCP and add data to the bucket	Medium	KiranKucha...	DONE
Task	KAN-12	Clean the data using Open Refine	Medium	Ankur Bhat...	DONE
Task	KAN-5	Create an outline for and format the presentation slides	Medium	Isabella Ph...	DONE
Task	KAN-16	Write Executive Summary	Medium	Bimala Joshi	DONE
Task	KAN-21	Machine Learning, Visualization and Interpretation	Medium	Cezane Karki	DONE
  - Confluence content: Project plan.
  - Activity: Show: All, Comments, History, Work log.
- Sidebar:** PLANNING (Summary, Timeline, Board, Calendar, List, Forms, Goals, All work), DEVELOPMENT (Code, Project pages, Add shortcut, Project settings, Archived work items NEW).

Figure 15.2 Jira dashboard (Task assigned to Epic).

The screenshot shows the Jira interface with the following details:

- Header:** Jira, Your work, Projects, Filters, Dashboards, Teams, Plans, Apps, Create.
- Project:** Harvesting, Retrieving, and Storing Data (Software project).
- View:** Summary.
- Summary Metrics:**
  - 20 completed (In the last 7 days)
  - 21 updated (In the last 7 days)
  - 8 created (In the last 7 days)
  - 0 due soon (In the next 7 days)
- Status Overview:** 24 Total work items. Breakdown: To Do: 1, Parking Lot: 0, In Progress: 0, In Review: 0, Done: 23.
- Recent Activity:**
  - Cezane Karki commented on KAN-1: Search for the streaming data and its API and API documentation. 10 days ago.
  - Waiting for the approval from the professor.
  - Cezane Karki commented on KAN-1: Search for the streaming data and its API and API documentation. 11 days ago.
- Priority Breakdown:** A bar chart showing priority levels: Highest (~1), High (~1), Medium (~22), Low (~1), Lowest (~0).
- Types of work:** Distribution by type: Epic (100%), Task (~5%), Feature (~5%), Subtask (~5%).
- Team workload:** Assignee: Unassigned. Work distribution: 100%.
- Epic progress:** See how your epics are progressing at a glance. View all epics. KAN-3: Deadly and damaging tropical cyclones Data Engineering Pipeline. Progress: 100%.

Figure 15.3 Jira dashboard (Summary of the Project).



## References

- American Red Cross. (2025). *About us*. <https://www.redcross.org/about-us.html>
- Berman, D. (2019, August 5). *Comparing Apache Hive vs. Spark*. Logz.Io.  
<https://logz.io/blog/hive-vs-spark/>
- Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Khan, S. U. (2015). The rise of “big data” on cloud computing: Review and open research issues. *Information Systems*, 47, 98–115. <https://doi.org/10.1016/j.is.2014.07.006>
- Imran, M., Castillo, C., Lucas, J., Meier, P., & Vieweg, S. (2014). AIDR: Artificial Intelligence for Disaster Response. *Proceedings of the 23rd International Conference on World Wide Web (WWW '14 Companion)*, 159–162.  
<https://doi.org/10.1145/2567948.2577034>
- Li, Z., Li, W., Wang, Y., & Duan, Y. (2020). A survey of deep learning-based human activity recognition in disaster response. *Pattern Recognition Letters*, 129, 120–126.  
<https://doi.org/10.3390/rs11091068>
- Shimizu. (2014, December 9). *American Red Cross logo vector*. BrandLogos.  
<https://brandlogos.net/american-red-cross-25705.html>