FatalCrash Explorer:

Project Report

Group members:

Shivani Jariwala (018284188),

Sakshat Patil (018318287),

Parth Patel (1008280535)

Class: CMPE180B

Prof: Georghi Guzun

Due date: May 1st, 2025

Table of Contents

Introduction / Motivation / Problem Definition:

We selected the NHTSA Traffic Fatalities dataset from BigQuery, which contains nationwide records of fatal motor vehicle crashes from 2015–2020. Our project, FatalCrash Explorer, is an interactive web application that visualizes crash patterns by time, location, demographics, driver behavior, and accident factors.

This dataset is compelling due to its real-world relevance: traffic fatalities are a major public safety issue, and exploring this data can help inform safer road policies and interventions. Our target audience include traffic safety officials, law enforcement, EMS coordinators, urban planners, and researchers, who would use the application to identify high-risk trends, optimize resources, and support public safety initiatives.

The project is challenging due to the dataset's complex schema (100+ tables), missing and inconsistent data, and the need for multiple view creations for efficient query searches in order to stay under BigQuery's limits. We used SQL for exploratory data analysis and Google Looker Studio for visualizations, selecting these tools for their effectiveness in handling large datasets and enabling interactive, user-friendly insights.

Related Work

Numerous studies and applications have explored traffic fatalities using large-scale public datasets, particularly those maintained by the National Highway Traffic Safety Administration (NHTSA). One of the most prominent tools is the Fatality Analysis Reporting System (FARS) Encyclopedia, an official NHTSA interface that enables users to query fatal crash data. While comprehensive in scope, the FARS tool is limited in flexibility—it does not support custom, multi-dimensional queries or offer interactive visual exploration capabilities.

Several academic studies have taken a predictive approach, employing machine learning and statistical models to estimate crash severity and identify high-risk zones. For instance, Zhao et al. (2020) applied logistic regression and random forests to analyze contributing factors such as weather conditions, road types, and driver demographics. However, these models typically prioritize prediction accuracy over user-friendly data exploration, making them less accessible to policymakers, urban planners, or first responders who lack technical expertise.

City-level dashboards such as NYC's Vision Zero View and Austin's Crash Data Explorer have also been developed to visualize traffic safety data. While valuable for local policy implementation, these platforms offer only localized insights and lack nationwide coverage or the ability to customize views across variables like vehicle types, violation categories, or emergency response patterns.

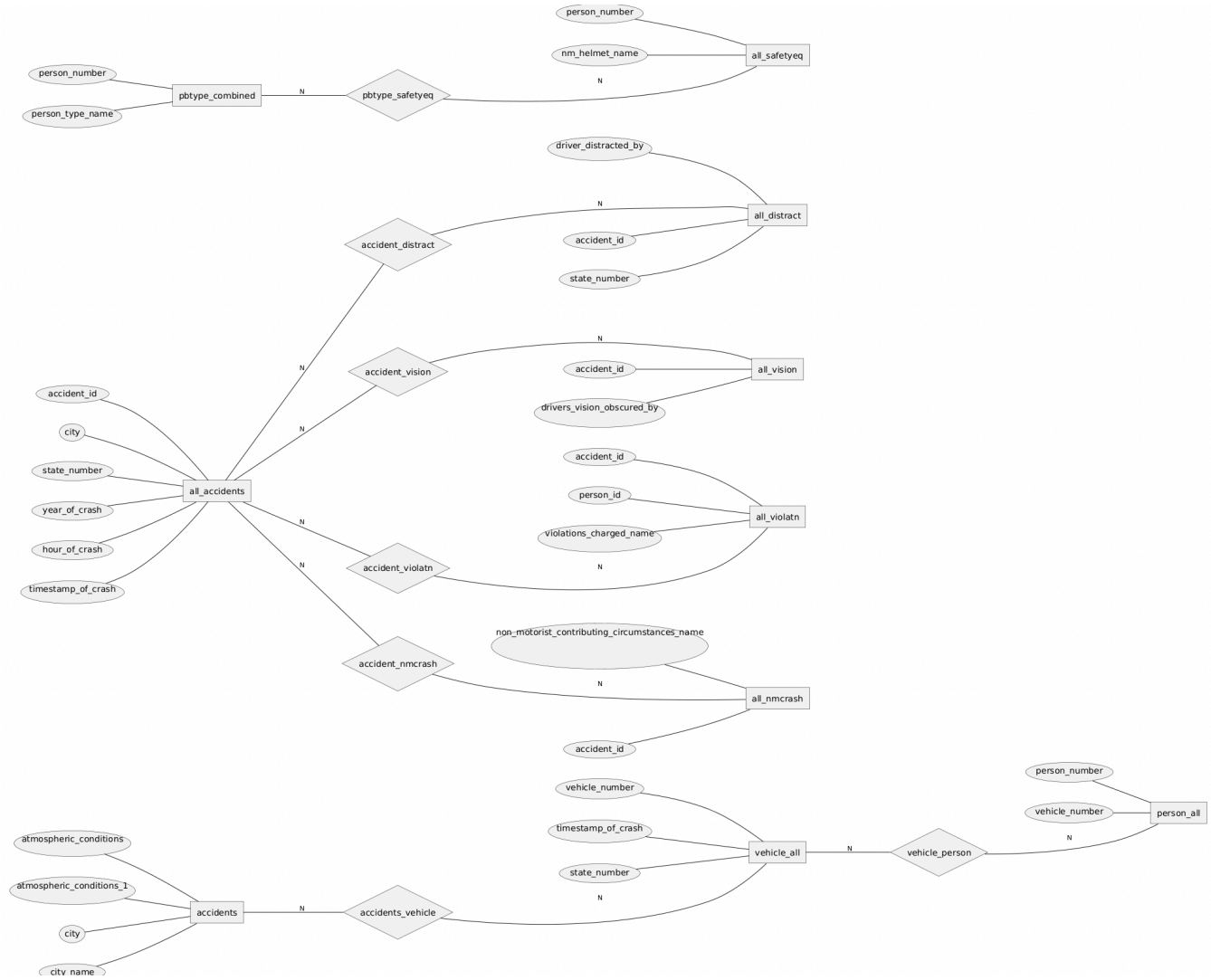
FatalCrash Explorer addresses these gaps by focusing on explorability and actionability rather than prediction alone. Key enhancements include:

- Granular Temporal Analysis:
  Unlike many tools that restrict temporal resolution to annual summaries, this project provides hourly, monthly, and yearly breakdowns, enabling stakeholders to detect patterns such as peak crash times or seasonal trends.

- Safety Equipment Insights:
  The dashboard highlights helmet usage and non-motorist safety gear adoption, dimensions rarely emphasized in existing public or academic tools, offering a more comprehensive view of safety compliance.

Methods Description

ER Diagram



Data Preparation

Due to the large number of tables in the dataset, we grouped tables by type to simplify querying and improve efficiency. For example, the dataset includes yearly tables such as accident_2015, accident_2016, accident_2017, and so on for each year. To streamline our analysis, we used UNION operations to merge tables of the same type into a single consolidated table. This approach reduced the number of working tables from 100+ to 18, making the querying process significantly more manageable. The queries used to perform these merges can be found on GitHub.

Application Features

To implement the key features of the FatalCrash Explorer dashboard, we developed several SQL queries to gain insights into the NHTSA Traffic Fatalities dataset. Each query targeted a unique aspect of fatal crash analysis, enabling the generation of interactive visualizations through Looker Studio. We incorporated the following features into our dashboard:

1. Accidents over the Past 5 Years (Parth)
2. Monthly Variation in Accident Counts (Sakshat)
3. Hourly Distribution of Crashes (Parth)
4. Statewide and City-Level Accident Breakdown (Parth)
5. Vehicle Types Involved and Helmut Usage (Sakshat)
6. EMS Response Time Distribution (Parth)
7. Types of Violations Charged (Parth)
8. Top Causes of Driver Distraction by State (Shivani)
9. Common Factors in Non-Motorist Fatal Crashes(Shivani)
10. Types of Vision Obstructions Reported(Shivani)

The SQL queries can be found here.

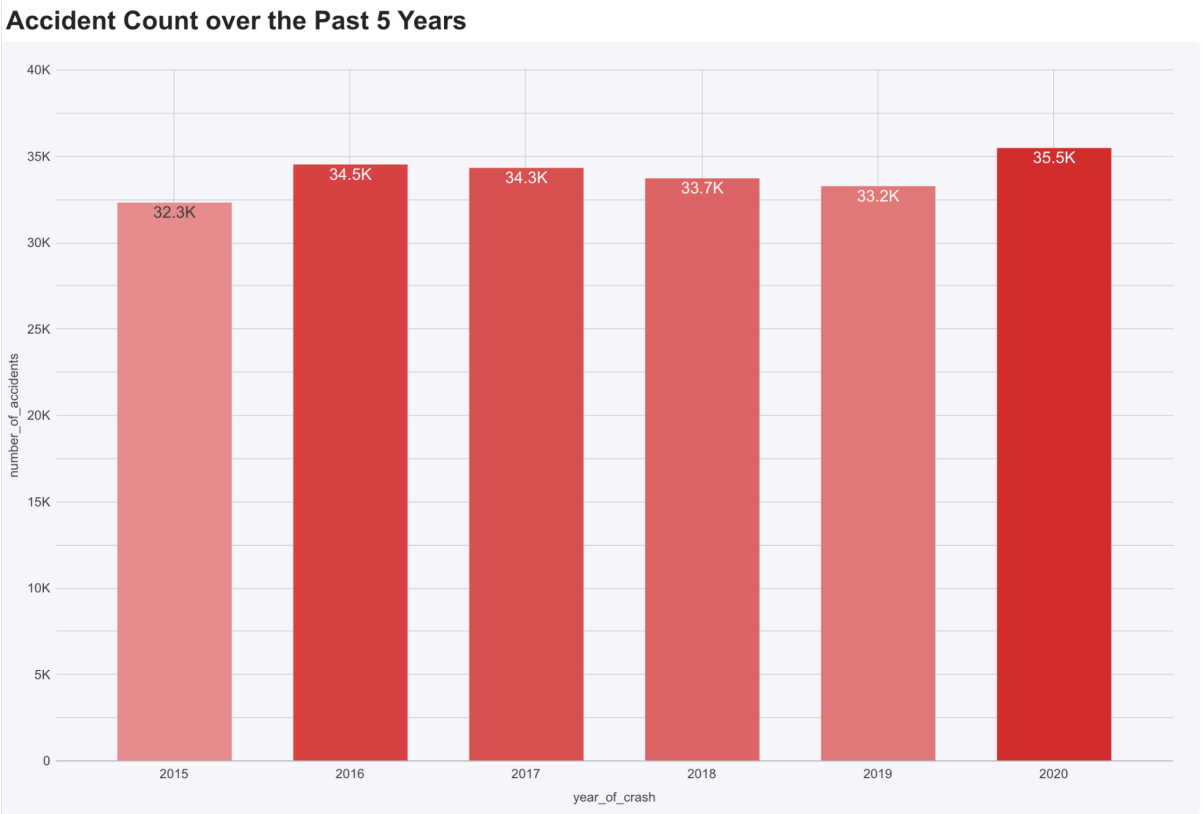
The visualizations in Looker Studio can be found here.

Feature Descriptions

Accidents over the Past 5 Years (Parth)

To understand long-term trends in fatal motor vehicle crashes, we aggregated crash data by year from 2015 to 2020. This visualization highlights the annual variation in the number of fatal accidents, providing insights into whether traffic safety conditions have improved, worsened, or remained stable over time.

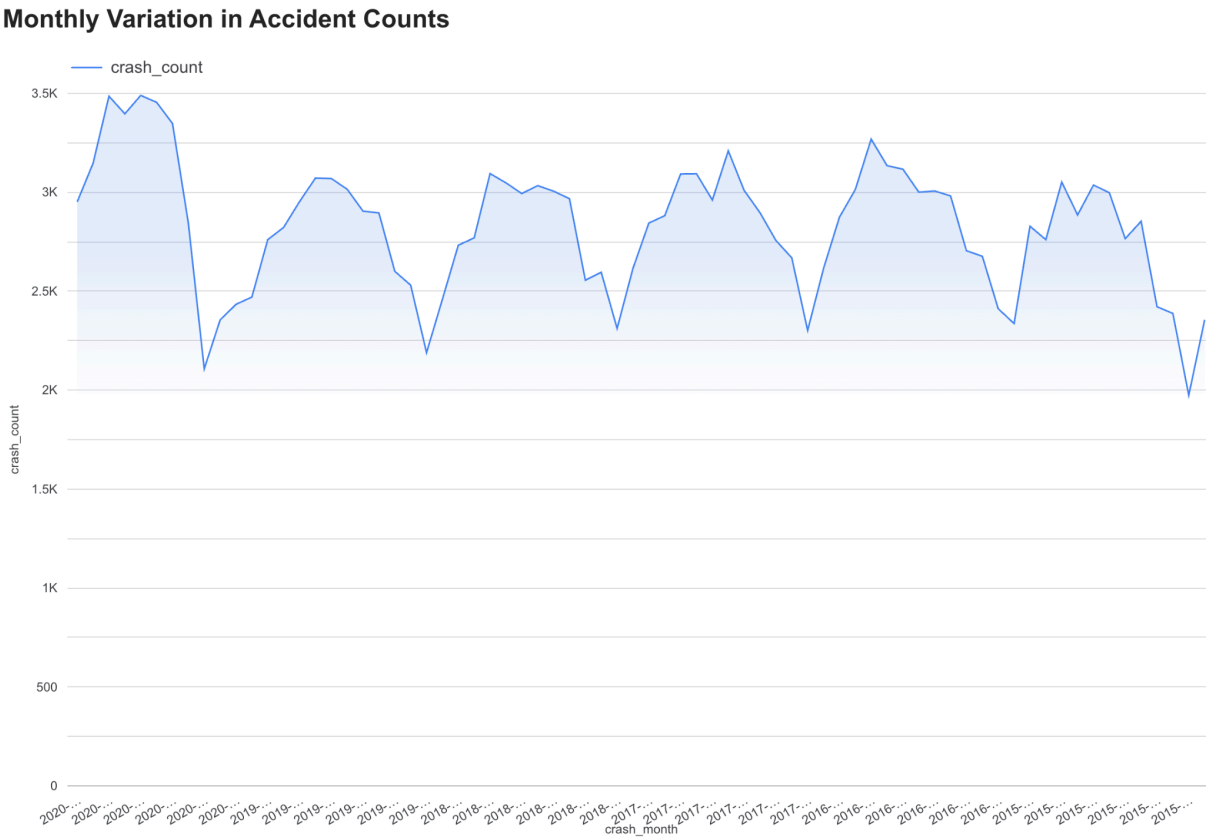
Query:
https://github.com/parthpatelsjsu/FatalCrashExplorer/blob/main/SQL%20Queries/yearly_accidents.sql



Monthly Variation in Accident Counts (Sakshat)

To further analyze temporal trends, we extracted the month and year of each crash and aggregated the total number of crashes by month. This approach allowed us to visualize seasonal patterns and observe fluctuations in fatal crash counts over time, highlighting periods with higher or lower incident rates.

Query:
https://github.com/parthpatelsjsu/FatalCrashExplorer/blob/main/SQL%20Queries/accidents_per_month.sql



Hourly Distribution of Crashes (Parth)

To analyze the relationship between the time of day and fatal crash occurrences, we extracted the hour from each recorded crash event and aggregated the number of crashes for each hour. This visualization reveals patterns in crash frequency throughout a 24-hour period, helping to identify peak hours of risk.

Query:
https://github.com/parthpatelsjsu/FatalCrashExplorer/blob/main/SQL%20Queries/hourly_accidents.sql

Hourly Distribution of Crashes

	hour_of_crash_name	number_of_crashes
1.	0:00am-0:59am	
2.	1:00am-1:59am	
3.	2:00am-2:59am	
4.	3:00am-3:59am	
5.	4:00am-4:59am	
6.	5:00am-5:59am	
7.	6:00am-6:59am	
8.	7:00am-7:59am	
9.	8:00am-8:59am	
10.	9:00am-9:59am	
11.	10:00am-10:59am	
12.	11:00am-11:59am	
13.	12:00pm-12:59pm	
14.	1:00pm-1:59pm	
15.	2:00pm-2:59pm	
16.	3:00pm-3:59pm	
17.	4:00pm-4:59pm	
18.	5:00pm-5:59pm	
19.	6:00pm-6:59pm	
20.	7:00pm-7:59pm	
21.	8:00pm-8:59pm	
22.	9:00pm-9:59pm	
23.	10:00pm-10:59pm	
24.	11:00pm-11:59pm	

Statewide and City-Level Accident Breakdown (Parth)

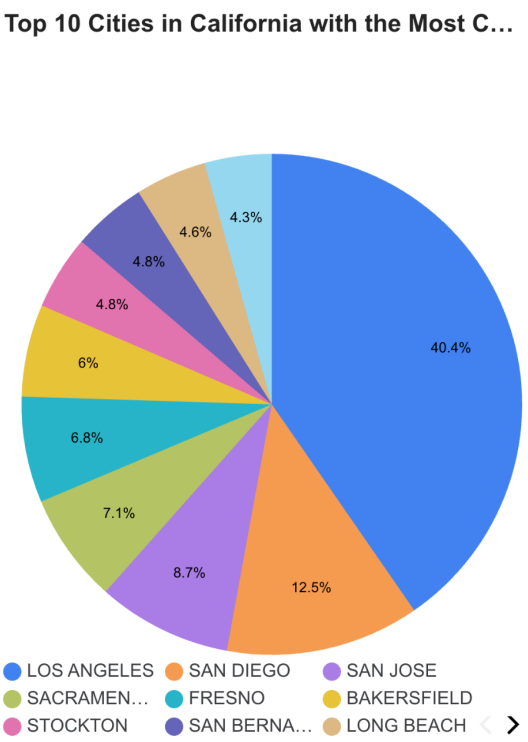
To explore the geographic distribution of fatal crashes, we aggregated accident counts at the state level. The first visualization highlights differences in fatal crash counts across states. Additionally, the page dives deeper into the city-level distribution within California, the state with the highest number of fatal accidents. Together, these visualizations help identify areas with higher concentrations of fatal traffic incidents, providing insights that can support targeted safety initiatives and policy development at both the state and city levels.

Queries:

- https://github.com/parthpatelsjsu/FatalCrashExplorer/blob/main/SQL%20Queries/accidents_by_state.sql
- https://github.com/parthpatelsjsu/FatalCrashExplorer/blob/main/SQL%20Queries/california_city_accidents.sql

Accidents By State

	state_...	total_accidents
1.	California	20,308
2.	Texas	20,093
3.	Florida	17,522
4.	Georgia	8,480
5.	North Ca...	8,015
6.	Pennsyl...	6,416
7.	Ohio	6,363
8.	Illinois	5,866
9.	New York	5,729
1.	Tennessee	5,670
1.	South C...	5,635
1.	Michigan	5,627
1.	Arizona	5,264
1.	Alabama	5,193
1.	Missouri	5,097
1.	Indiana	4,709
1.	Virginia	4,565
1.	Louisiana	4,221
1.	Kentucky	4,216
2.	Mississippi	3,700
2.	Oklahoma	3,545
2.	Colorado	3,372
2.	New Jer...	3,277
2.	Wisconsin	3,171



Vehicle Types Involved and Helmet Usage (Sakshat)

1.Safety Equipment Usage among Non-Motorists

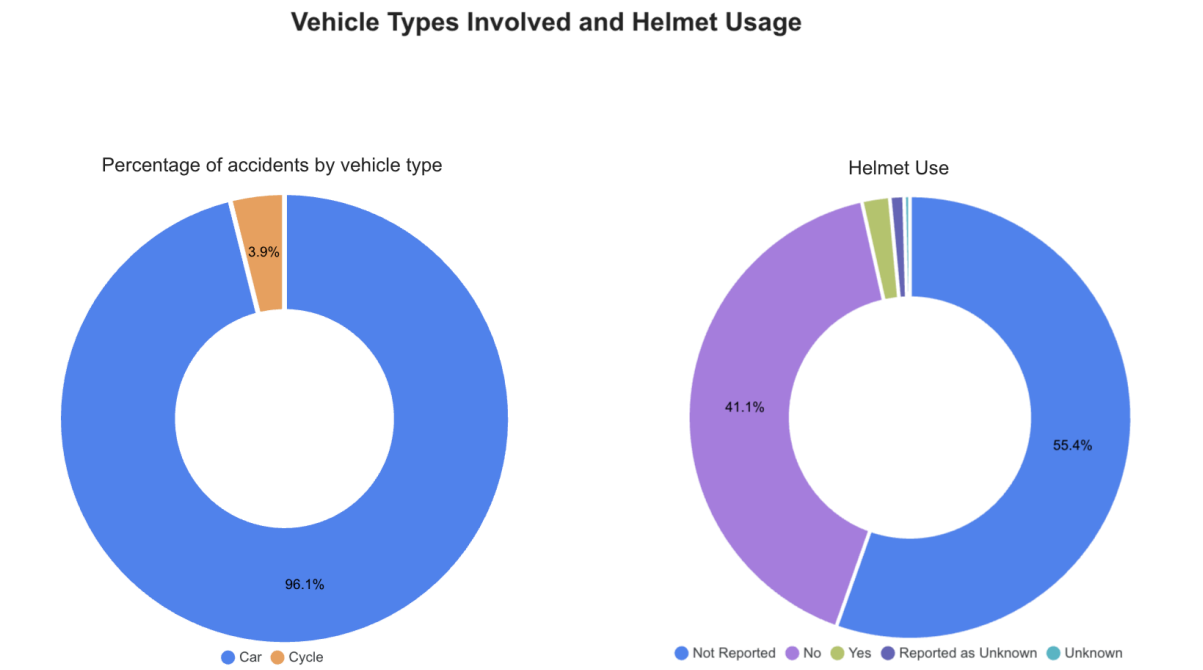
To analyze the usage of **helmets and other safety equipment** among pedestrians and cyclists involved in crashes, we joined the all_pbtype and all_safetyeq tables. This query identifies the relationship between helmet usage and the types of non-motorists affected.

Query:
https://github.com/parthpatelsjsu/FatalCrashExplorer/blob/main/SQL%20Queries/helmet_use.sql

2.Vehicle Type Involvement across Time Dimensions

To examine the vehicle distribution, we aggregated accidents based on vehicle type and found that only a minority of accidents involve motor vehicles.

Query:
https://github.com/parthpatelsjsu/FatalCrashExplorer/tree/main/SQL%20Queries#:~:text=yesterday-,vehicle_type,-Create%20vehicle_type

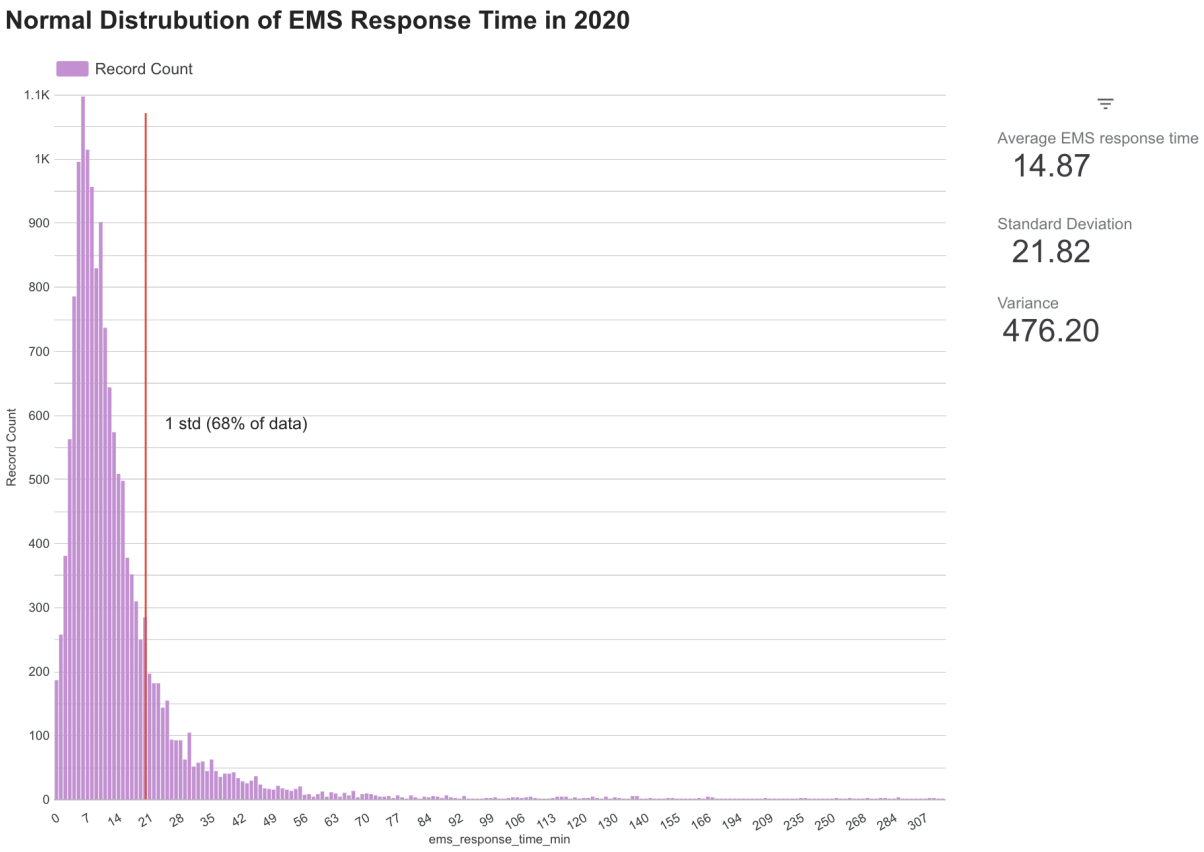


EMS Response Time Distribution (Parth)

This page analyzes the distribution of Emergency Medical Services (EMS) response times following fatal crashes of 2020. The histogram displays EMS response time in minutes (X-axis) against the number of incidents (Y-axis), illustrating how quickly emergency services typically arrive at crash scenes. To ensure a clean and reliable analysis, extreme outliers were removed based on a ±2 standard deviation filtering method.

Key statistical metrics—including the mean, standard deviation, and variance—are displayed alongside the chart to provide further context. A reference line marks one standard deviation from the mean, capturing approximately 68% of the data and highlighting the typical range of EMS response times. This analysis helps identify whether response times are consistent or vary significantly, offering potential insights for emergency response optimization.

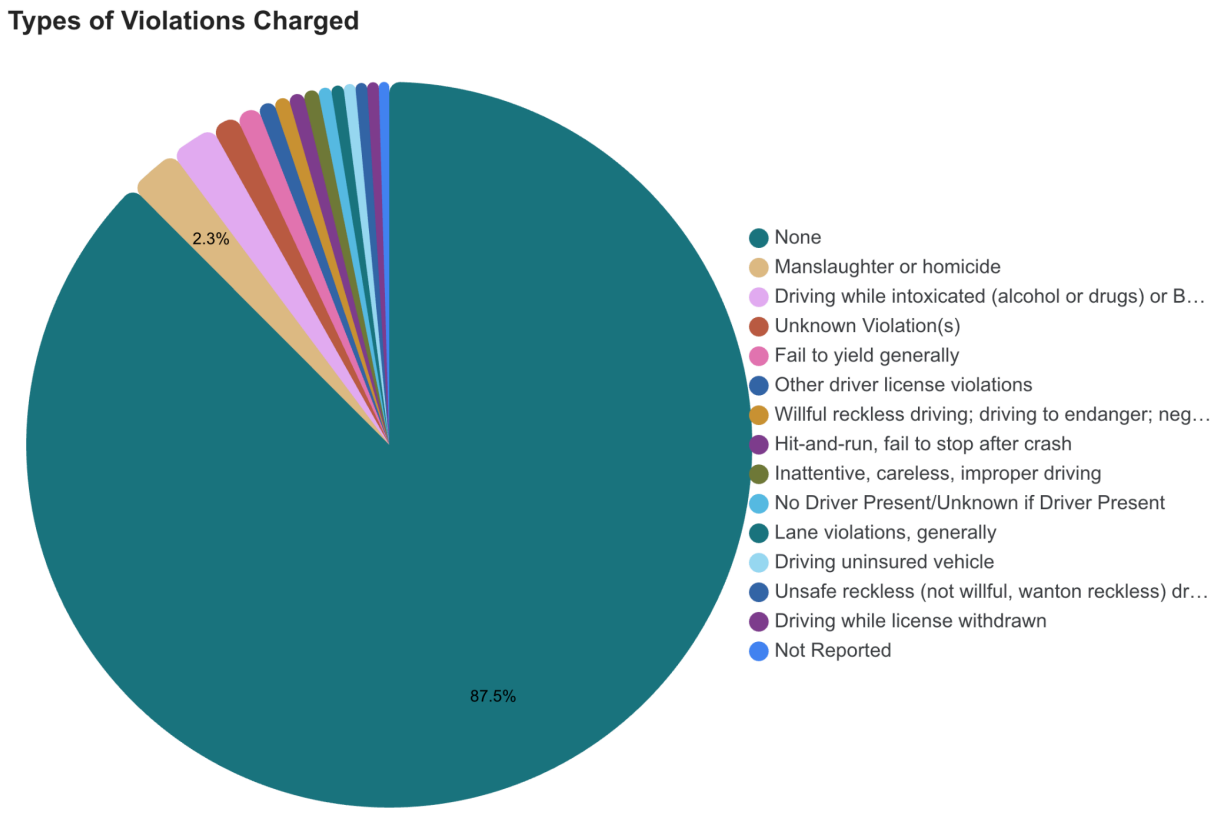
Query:
https://github.com/parthpatelsjsu/FatalCrashExplorer/blob/main/SQL%20Queries/ems_response_time_optimized.sql



Types of Violations Charged (Parth)

This page presents an analysis of the different traffic violations charged in fatal crash incidents. A pie chart displays the distribution of violation types, allowing for a quick visual comparison of the most common contributing violations, such as speeding, impaired driving, and failure to yield. It is important to note that a significant portion of records are categorized as "None," since many crashes involve minor violations such as parking infractions or speeding that were not formally recorded as contributing factors to the fatality.

Query:
https://github.com/parthpatelsjsu/FatalCrashExplorer/blob/main/SQL%20Queries/violation_types.sql



Top Causes of Driver Distraction by State (Shivani)

This analysis identifies the most frequent driver distraction involved in fatal crashes for each U.S. state. The query filters out ambiguous categories such as "Not Reported" and "Unknown" to highlight meaningful distraction causes like cell phone use, passenger interference, or external distractions. The output presents a clear, state wise breakdown that reveals how distraction types vary across different regions.

Query:

https://github.com/parthpatelsjsu/FatalCrashExplorer/blob/main/SQL%20Queries/distractions_by_state.sql

Top Causes of Driver Distraction by State

	state_name	top_reason_for_distraction	distraction_case_count
1.	Texas	Inattention (Inattentive), Details Unknown	1,264
2.	Florida	Inattention (Inattentive), Details Unknown	890
3.	New Mexico	Inattention (Inattentive), Details Unknown	730
4.	New Jersey	Inattention (Inattentive), Details Unknown	716
5.	Kentucky	Inattention (Inattentive), Details Unknown	649
6.	North Carolina	Inattention (Inattentive), Details Unknown	544
7.	Kansas	Inattention (Inattentive), Details Unknown	418
8.	Louisiana	Inattention (Inattentive), Details Unknown	328
9.	Washington	Inattention (Inattentive), Details Unknown	280
10.	Virginia	Other Distraction	272
11.	Arizona	Looked But Did Not See	196
12.	Tennessee	Other Cellular Phone Related	173
13.	California	Looked But Did Not See	142
14.	Idaho	Inattention (Inattentive), Details Unknown	123
15.	Maryland	Looked But Did Not See	114
16.	Ohio	Distracted by Outside Person, Object or Event	113
17.	Oklahoma	Inattention (Inattentive), Details Unknown	101
18.	Hawaii	Inattention (Inattentive), Details Unknown	98
19.	Colorado	Inattention (Inattentive), Details Unknown	90
20.	Michigan	Distracted by Outside Person, Object or Event	84
21.	Wisconsin	Inattention (Inattentive), Details Unknown	79
22.	Massachusetts	Other Distraction	63

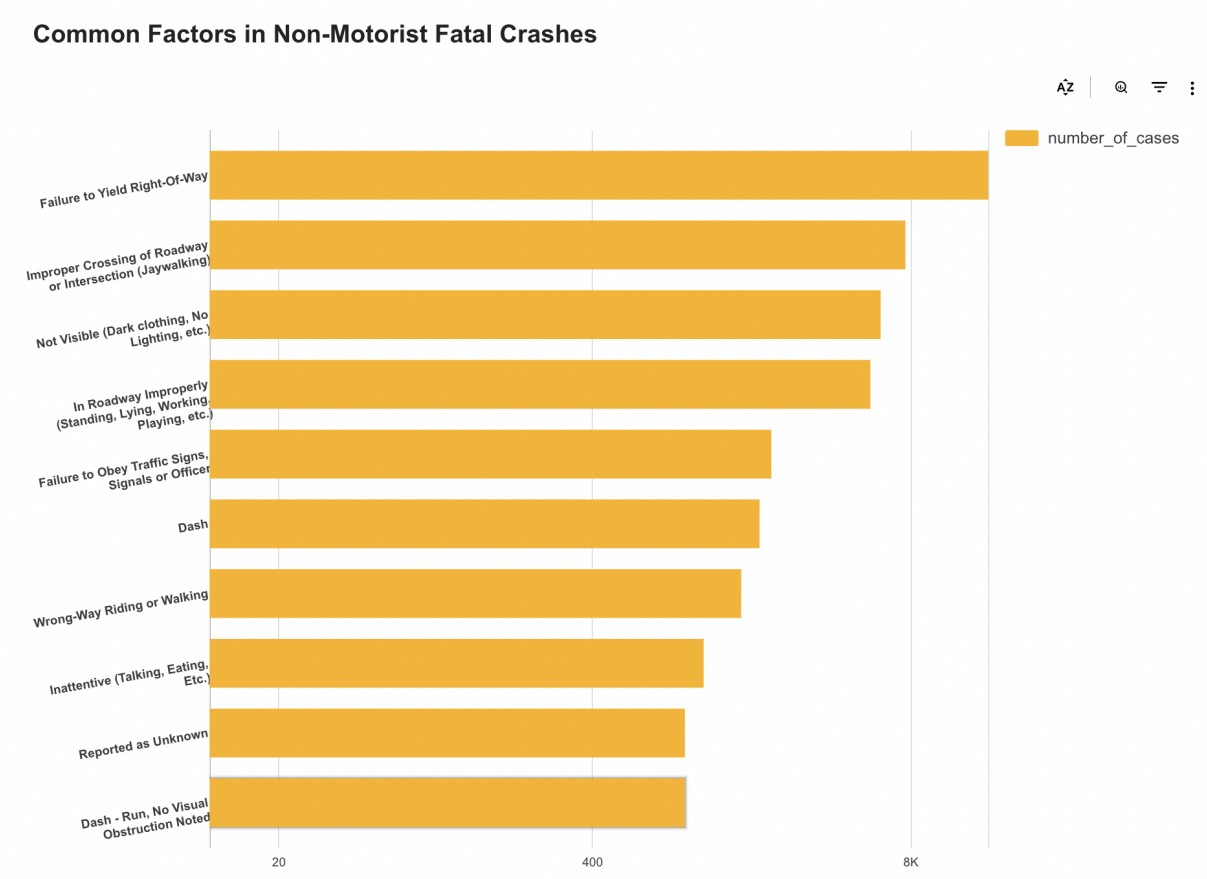
Other Cellular Phone Related

Common Factors in Non-Motorist Fatal Crashes (Shivani)

This analysis highlights the most frequently recorded actions or behaviors of non-motorists—such as pedestrians and cyclists—that contributed to fatal crashes. The query excludes generic or uninformative entries like "Unknown" and "None Noted" to surface meaningful contributing factors such as jaywalking, failure to yield, and poor visibility. The output presents a ranked list of risky behaviors attributed to non-motorists, offering valuable insight into behavioral patterns that often precede fatal collisions.

Query:

https://github.com/parthpatelsjsu/FatalCrashExplorer/blob/main/SQL%20Queries/non-motorist_mistakes.sql

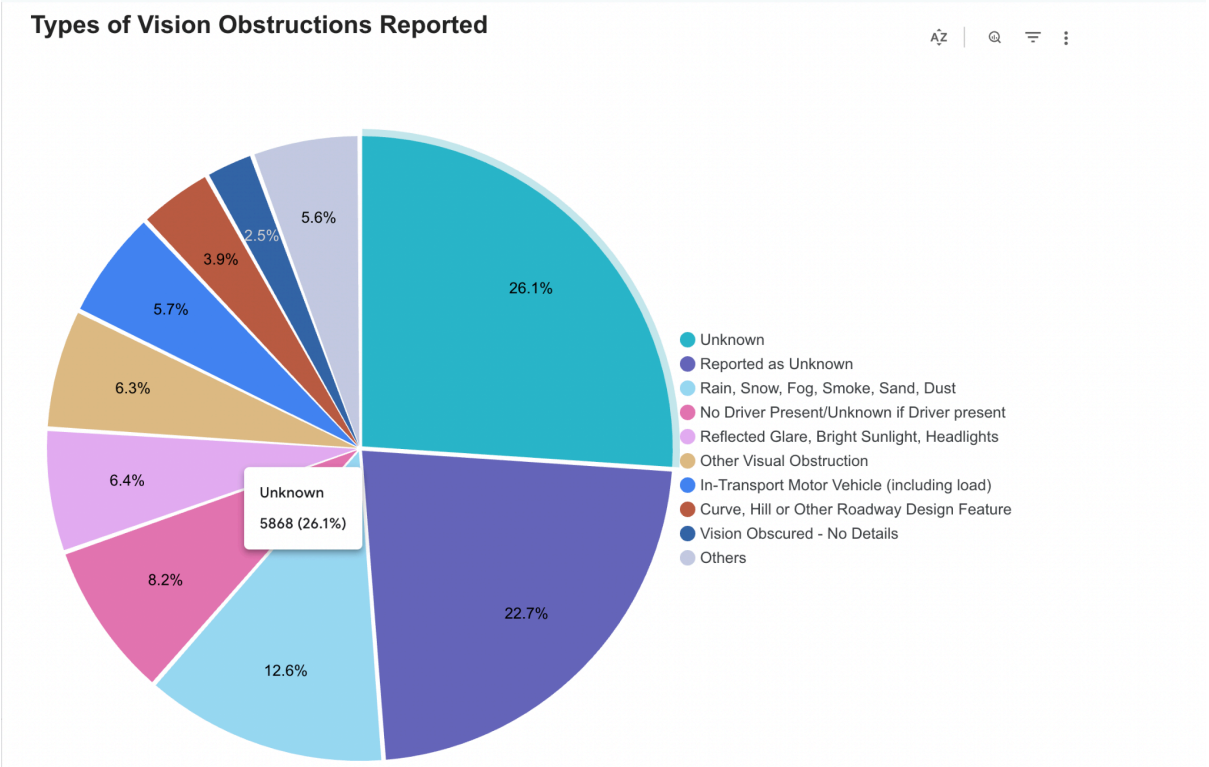


Types of Vision Obstructions Reported (Shivani)

This page explores the different factors that obstructed drivers' vision in crash incidents. The analysis highlights common causes such as broken or dirty windshields, internal vehicle obstructions, and obstructing angles. Notably, a significant portion of records fall under ambiguous categories like "Unknown" or "No Details," which may indicate underreporting or incomplete data collection during crash investigations. The insight emphasizes the importance of clear visibility for drivers and draws attention to both environmental and mechanical risks on the road.

Query:

https://github.com/parthpatelsjsu/FatalCrashExplorer/blob/main/SQL%20Queries/vision_obstructions.sql



Query Analysis

In our FatalCrash Explorer application, three queries that uncovered the most important to our application were:

Monthly Variation in Accident Counts (Sakshat)

This query was responsible for generating the “Monthly Variation in Accident Counts” feature of the FatalCrash Explorer dashboard. It provides a time-series summary of accident frequency, enabling analysts to identify seasonal trends, patterns, or anomalies in crash occurrence across different years. The query aggregates crash data on a monthly basis, making it easier to interpret long-term trends rather than dealing with raw daily values. This aids in decision-making for road safety policy and resource allocation by highlighting months or seasons with higher accident rates.

The query used the accidents table and involved the following attributes:

- crash_date (timestamp or date field indicating when the crash occurred)

The query computes:

- crash_month: A formatted string (e.g., '2021-03') representing the month of the crash.
- crash_count: Total number of crashes in that month

Optimization

The query used for this feature can be optimized for performance through several techniques. Since it relies heavily on time-based attributes, a traditional relational database would benefit from an index on timestamp_of_crash to improve the efficiency of the WHERE clause by avoiding full table scans.

Additionally, the ems_response_time_min value is currently calculated at runtime for each row. To reduce computational overhead, this value can be stored as a precomputed column during data ingestion or preprocessing.

We also optimized the query by selecting only the attributes required for computation rather than using SELECT *. This approach minimizes the number of columns read, reducing I/O cost.

We created a partitioned version of the original table, called accidents_partition, where we partitioned the table by DATE(timestamp_of_crash).

Running the same query on this partitioned table led to noticeable performance improvements, as summarized below:

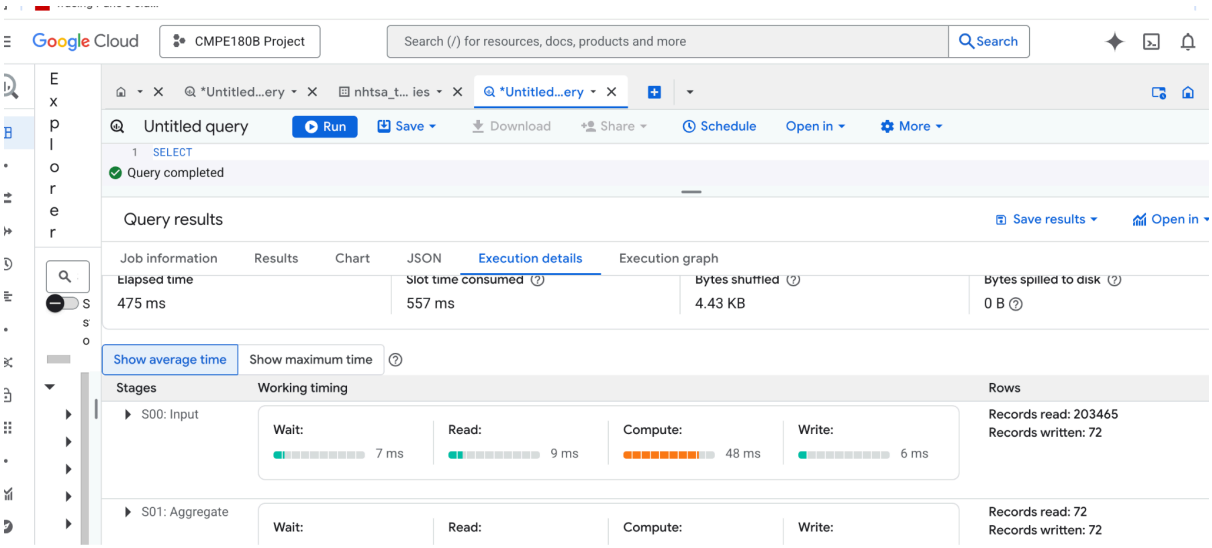
Partition query:
https://github.com/parthpatelsjsu/FatalCrashExplorer/blob/main/SQL%20Queries/monthly_crash_partition.sql

Metrics:

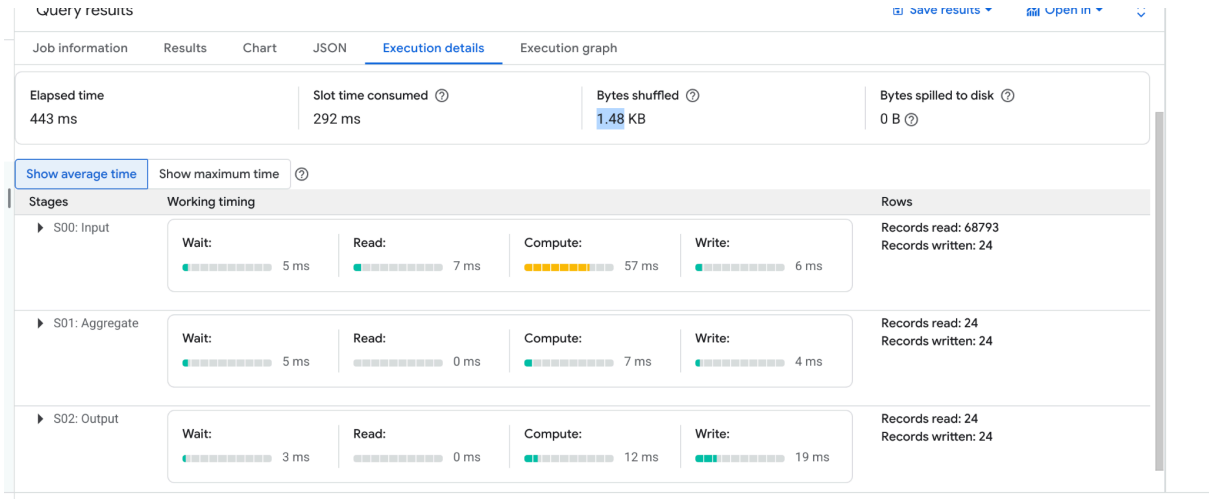
	Unoptimized	**Optimized**
Elapsed Time	475 sec	443 ms
Slot time consumed	557 ms	292 ms
Bytes shuffled	4.43 KB	1.48 KB
Bytes splitted to disk	0 B	0 B

The I/O cost of using the non-partitioned table is significantly higher because it requires a full table scan to evaluate each row. While BigQuery does not expose I/O cost in terms of pages like traditional relational databases, we can compare the **number of records read** to understand the difference in efficiency.

IO cost of non-optimized query:



IO cost of optimized query:



As shown above, the non-optimized query read 203465 records, whereas the optimized version used partitions and read 68793.

EMS Response Time Distribution (Parth)

What features are they responsible for? (remove this line later)

This query was responsible for generating the **“EMS Response Time Distribution”** feature of the FatalCrash Explorer dashboard. It provides a cleaned and statistically meaningful dataset that visualizes how long it takes for emergency medical services to arrive at the crash scene, enabling analysis of EMS efficiency in fatal crash scenarios. The query filters out outliers using a ±2 standard deviation rule and calculates key statistics such as the mean, standard deviation, and variance to support more accurate interpretation and visualization.

This query used the **accidents** table, and used the following attributes:

- Hour_of_arrival_at_scene

- Minute_of_arrival_at_scene

Optimization

The query used for this feature can be optimized for performance through several techniques. Since it relies heavily on time-based attributes, a traditional relational database would benefit from an index on *(hour_of_arrival_at_scene, minute_of_arrival_at_scene)* to improve the efficiency of the WHERE clause by avoiding full table scans. Additionally, the ems_response_time_min value is currently calculated at runtime for each row. To reduce computational overhead, this value can be stored as a pre-computed column during data ingestion or preprocessing.

We also optimized the query by selecting only the attributes required for the computation, rather than using SELECT *, which minimizes the number of columns read and reduces I/O cost.

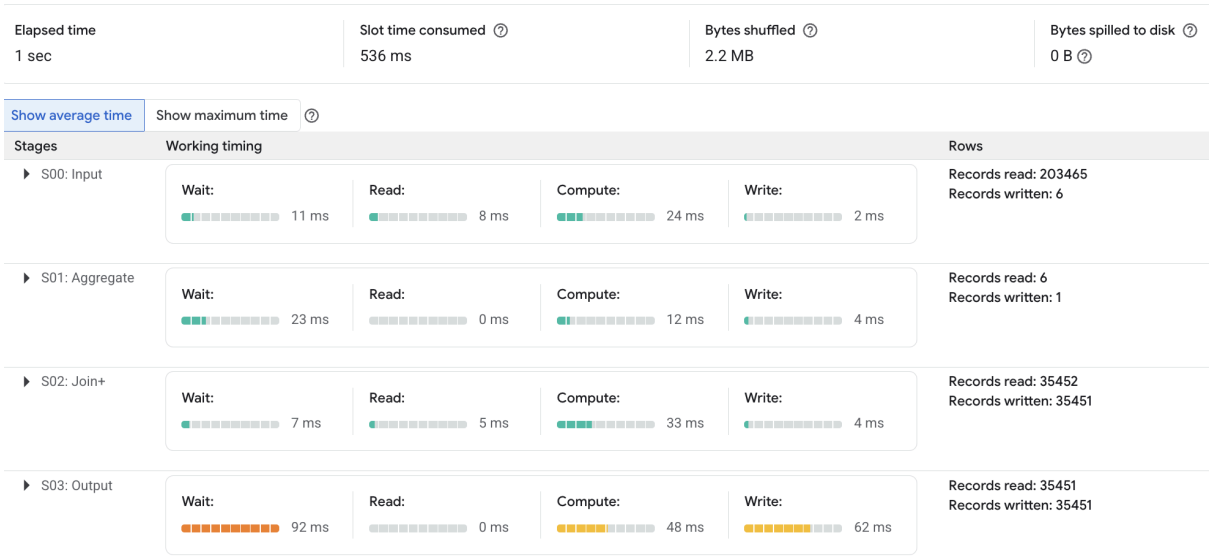
While BigQuery does not support manually creating indexes like traditional databases, it does allow partitioning and clustering. To take advantage of this, we created a partitioned version of the original table called *accidents_partition*, where we synthesized a DATETIME column using *hour_of_arrival_at_scene* and *minute_of_arrival_at_scene*, and partitioned the table on the resulting date. By running the same query on this partitioned table, we observed improved performance as seen below:

Partition query:
https://github.com/parthpatelsjsu/FatalCrashExplorer/blob/main/SQL%20Queries/partition_ems_response_time.sql
Metrics:

	Unoptimized	**Optimized**
Elapsed Time	1 sec	378 ms
Slot time consumed	536 ms	261 ms
Bytes shuffled	2.2 MB	499.32 KB
Bytes splitted to disk	0 B	0 B

The I/O cost of using the non-partitioned table is significantly higher because it requires a full table scan to evaluate each row. While BigQuery does not expose I/O cost in terms of pages like traditional relational databases, we can compare the **number of records read** to understand the difference in efficiency.

IO cost of non-optimized query:

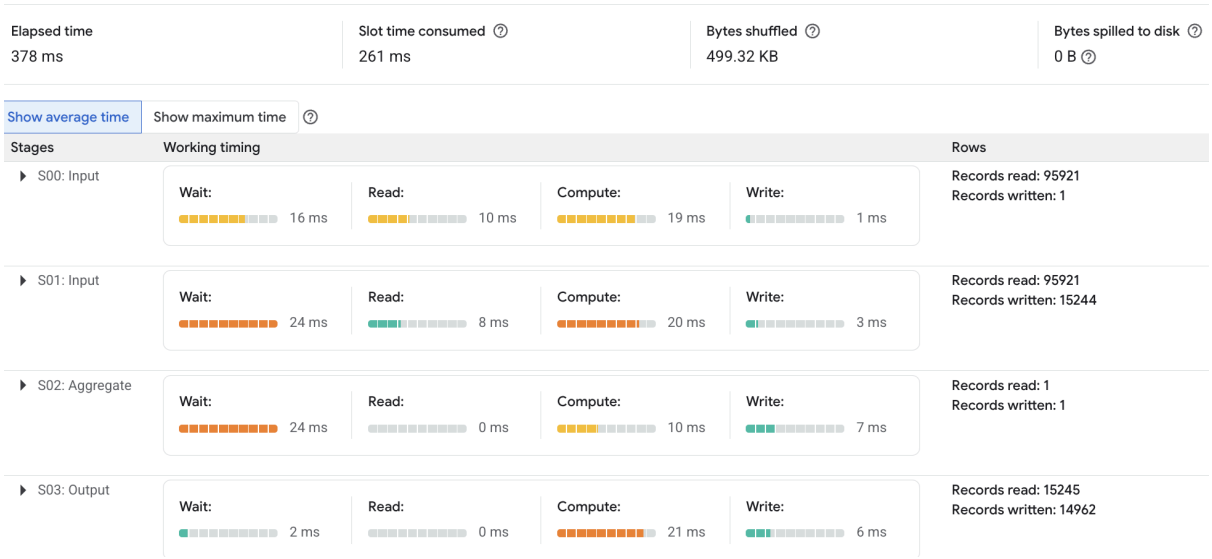


Elapsed time	Slot time consumed	Bytes shuffled	Bytes spilled to disk
1 sec	536 ms	2.2 MB	0 B

Show average time | Show maximum time

Stages	Working timing	Rows
S00: Input	Wait: 11 ms, Read: 8 ms, Compute: 24 ms, Write: 2 ms	Records read: 203465 Records written: 6
S01: Aggregate	Wait: 23 ms, Read: 0 ms, Compute: 12 ms, Write: 4 ms	Records read: 6 Records written: 1
S02: Join+	Wait: 7 ms, Read: 5 ms, Compute: 33 ms, Write: 4 ms	Records read: 35452 Records written: 35451
S03: Output	Wait: 92 ms, Read: 0 ms, Compute: 48 ms, Write: 62 ms	Records read: 35451 Records written: 35451

IO cost of optimized query:



Elapsed time	Slot time consumed	Bytes shuffled	Bytes spilled to disk
378 ms	261 ms	499.32 KB	0 B

Show average time | Show maximum time

Stages	Working timing	Rows
S00: Input	Wait: 16 ms, Read: 10 ms, Compute: 19 ms, Write: 1 ms	Records read: 95921 Records written: 1
S01: Input	Wait: 24 ms, Read: 8 ms, Compute: 20 ms, Write: 3 ms	Records read: 95921 Records written: 15244
S02: Aggregate	Wait: 24 ms, Read: 0 ms, Compute: 10 ms, Write: 7 ms	Records read: 1 Records written: 1
S03: Output	Wait: 2 ms, Read: 0 ms, Compute: 21 ms, Write: 6 ms	Records read: 15245 Records written: 14962

As shown above, the non-optimized query read 203,465 records, whereas the optimized version used partitions and read 95,921 records, resulting in a 52% reduction in records scanned.

The reason for such a drastic reduction in rows is because the table was partitioned on the ems_response_time, and the final query in the CTE contains a WHERE condition on this field. Of all the different partitions attempted, such as *year_of_crash*, and *state_name*, this resulted in the greatest reduction in both IO and execution time.

Top Causes of Driver Distraction by State (Shivani)

This query powers the “Top Driver Distraction Causes by State” feature of the FatalCrash Explorer dashboard. It analyzes fatal crash data to identify the most common cause of driver distraction in each U.S. state. By filtering out vague or non-informative categories such as

“Not Reported” and “Unknown,” the dataset provides meaningful, state-specific insights into driver behavior and its role in fatal accidents

The query uses the **all_distract** table and relies on the following attributes:

- state_name
- Driver_distracted_by_name

Using a common windowing technique (ROW_NUMBER partitioned by `state_name`), the query selects only the highest-ranked distraction cause per state based on frequency, producing a clean and concise summary across all 50 states.

Optimization

To optimize performance and cost-efficiency, we created a partitioned version of the original table (all_distract_partition). Since the base table lacks a natural timestamp column, we appended a synthetic ingestion timestamp (CURRENT_TIMESTAMP() AS ingestion_time) and partitioned by DATE(ingestion_time). We also clustered the table by state_name to speed up region-specific filtering and aggregation.
Additionally, the query avoids using SELECT * and retrieves only the required columns, reducing scan costs and improving performance in BigQuery.

By running the query against the partitioned and clustered table, we observed noticeable reductions in execution time and data scanned, making it more suitable for dashboard refreshes and interactive filtering.

Partition query:
https://github.com/parthpatelsjsu/FatalCrashExplorer/blob/main/SQL%20Queries/distractions_by_state_partition.sql

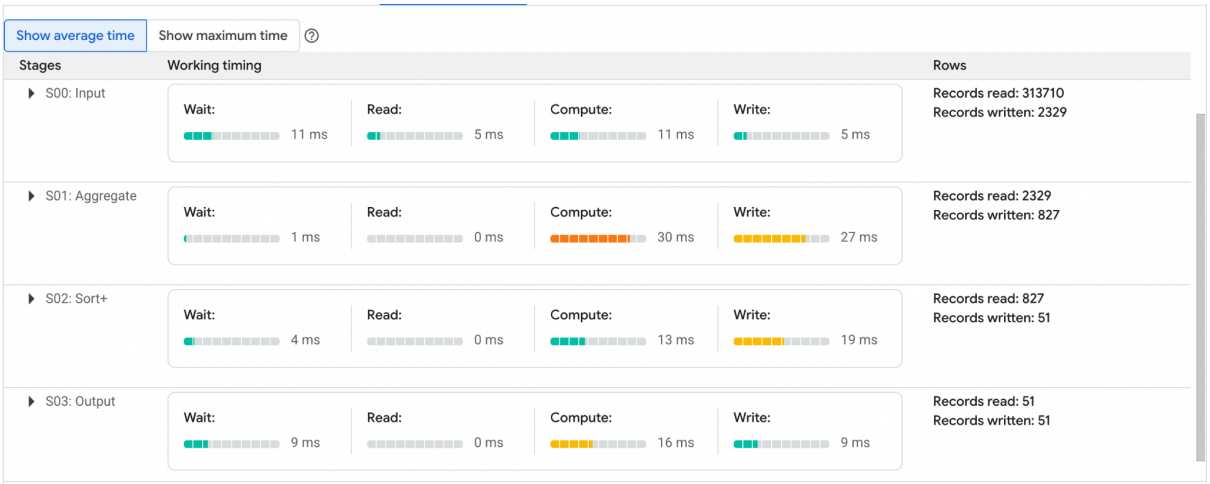
Metrics:

	Unoptimized	**Optimized**
Elapsed Time	2 sec	717 ms
Slot time consumed	42 sec	9 sec
Bytes shuffled	193.51 KB	103.14 MB
Bytes splitted to disk	0 B	0 B

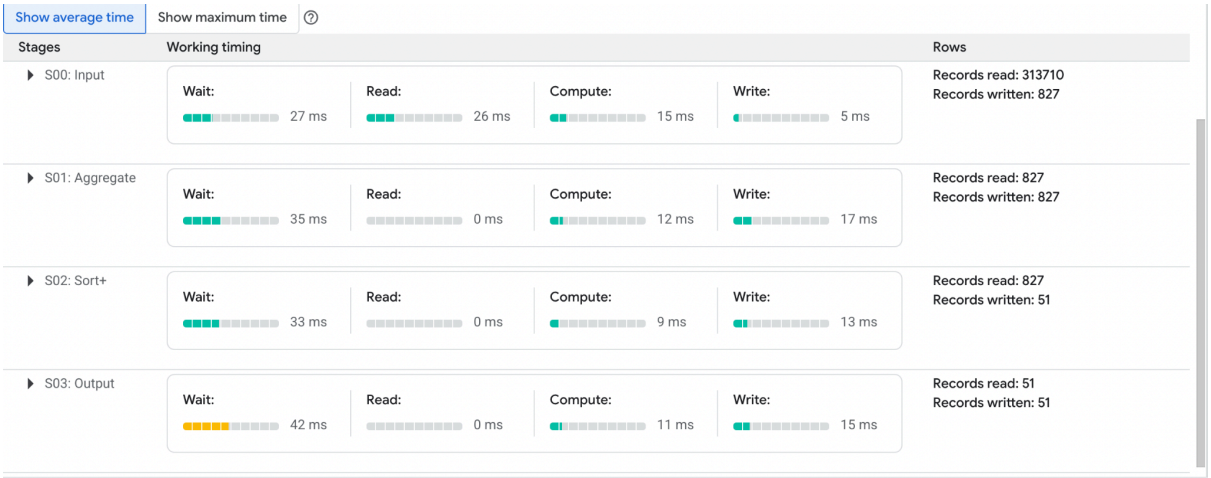
The I/O cost of querying the non-partitioned table is significantly higher because BigQuery must scan the entire dataset to compute results. While BigQuery doesn't expose I/O cost in terms of traditional page reads, it does report the number of bytes scanned for each query. By comparing bytes scanned between the original and partitioned tables, we can measure the

efficiency gain from partition pruning and clustering. This not only reduces query cost but also improves response time, especially when filtering by state or analyzing large datasets.

IO cost of non-optimized query:



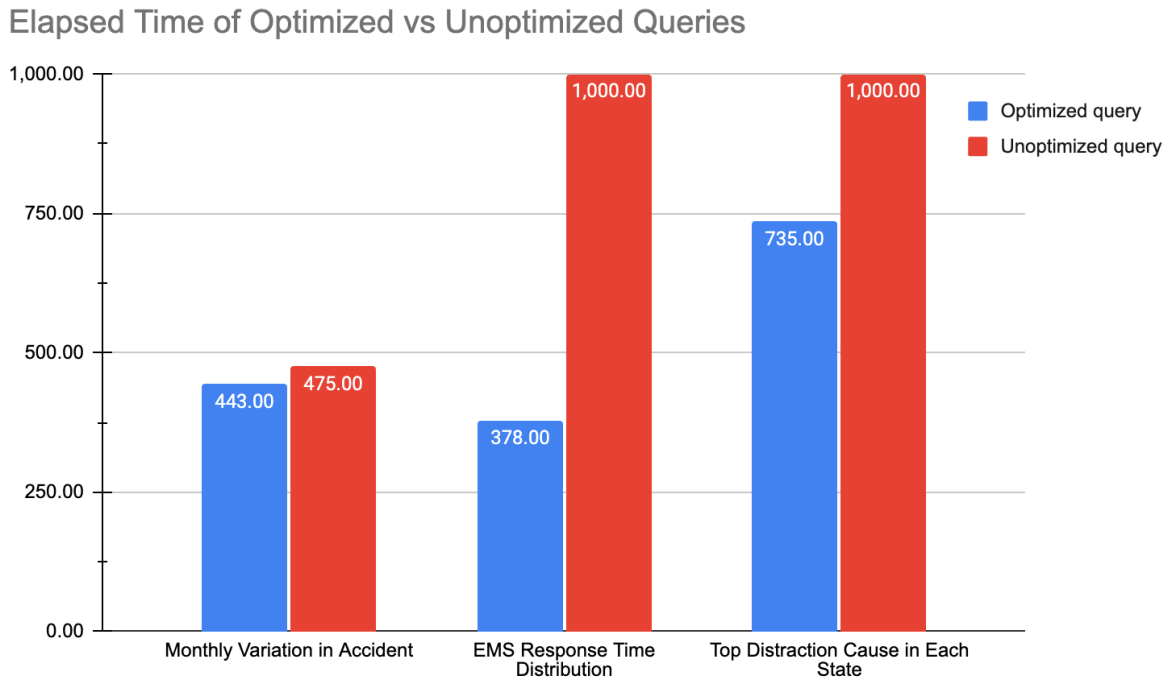
IO cost of optimized query:

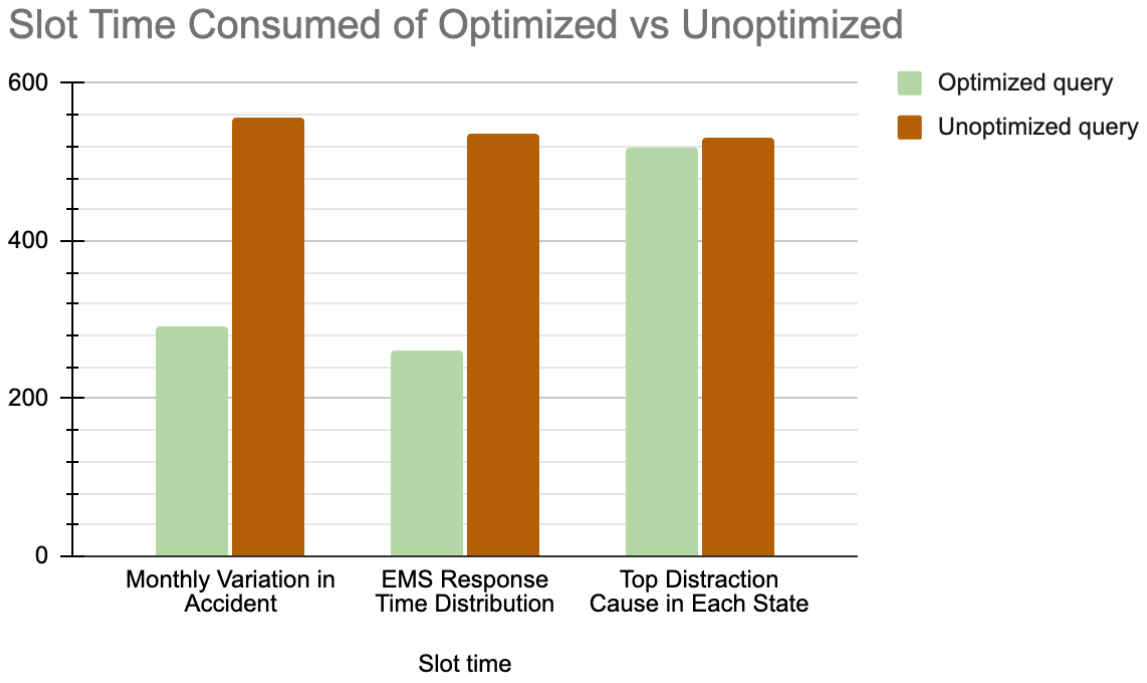


Results and Findings

We measured both the **elapsed time** and **slot time** for three queries, each executed in both optimized and unoptimized forms. While the optimized versions consistently showed improved performance, the reduction in execution time was relatively minor. It is important to note that performance in BigQuery can also be influenced by external factors, such as concurrent system load and resource availability at the time of execution.

A graphical representation of our findings is provided below:





Impact of Optimizations

The query optimizations implemented, such as avoiding SELECT *, precomputing fields using CTE's , and partitioning the dataset, led to measurable improvements in query performance. The most significant impact was seen in the reduction of I/O operations when using the partitioned table. While improvements in elapsed time and slot time were relatively modest on average, this is expected given the small data volume and BigQuery's automatic

resource scaling. Nevertheless, the reduction in bytes shuffled and records scanned demonstrates that these optimizations effectively improved query efficiency, especially in terms of cost and scalability for larger datasets.

The largest performance increase was seen in the EMS response time query. The optimized query filtered on *year_of_crash* early in the CTE rather than at the end, the table was partitioned on ems_response_time, and the WHERE condition filtered on ems_response_time, thus utilizing the benefit of partitioning as shown in the compute time and graphs.

Future Improvements

Future improvements could focus on further optimizing data storage and query efficiency by incorporating clustering on high-cardinality columns such as state_name or city_name, which are commonly used in filters and aggregations. Additionally, introducing a dedicated TIMESTAMP column instead of an INTEGER, at the data ingestion stage (e.g., for EMS arrival time) would allow for more meaningful and effective partitioning. We could also benefit from using a traditional database, where we can add indexes on time-based attributes. Lastly, if the dataset grows by adding more recent years, then implementing incremental data loading and partition pruning strategies will be essential to maintain performance.

Conclusions and Lessons Learned

This project demonstrated the value of combining SQL-based data exploration with cloud-native tools such as BigQuery and Looker Studio. By working with the NHTSA Traffic Fatalities dataset, we developed a comprehensive dashboard, FatalCrash Explorer, that enables users to interactively explore fatal crash trends across time, geography, vehicle types, and behavioral factors. Our work highlighted not only the power of structured querying but also the importance of data transformation, optimization, and visualization in communicating insights effectively.

Throughout the project, we encountered and overcame several challenges. The highly normalized and year-split structure of the dataset made querying cumbersome initially, prompting us to consolidate tables by type using UNION. We also learned the importance of query optimization, such as avoiding unnecessary column reads, reusing expressions, and leveraging BigQuery features like partitioning to reduce I/O and execution costs. While BigQuery automates many performance enhancements, we found that thoughtful query design and data structure choices had a measurable impact on efficiency and usability.

One key lesson was that dashboard performance depends not only on SQL correctness but also on how well the underlying queries and datasets are prepared. Designing for scalability, by pre-aggregating data, reducing compute overhead, and minimizing scan costs, proved essential, even when working with a cloud data warehouse. Additionally, the integration with Looker Studio emphasized the need for clean, well-structured query outputs to support meaningful visualizations.