

University of Arkansas – CSCE Department
Capstone I – Preliminary Report – Spring 2026

Formula 1 Telemetry Insights Visual Dashboard

Samuel Brooks, Nicholas Calhoun, Blake Dismore, Eli Rose, Kolby Stoll, Caden Williamson

Abstract

This research proposal outlines the development of a visual dashboard for F1 racing telemetry data, designed to intake, cluster, and visualize driver-specific telemetry data for evaluating performance and making comparisons. In modern Formula 1, massive amounts of data are collected every millisecond, but transforming this raw information into actionable decisions is difficult without advanced analytical tools. A visual dashboard addresses this by translating the large dataset into a clear, interactive webpage, simplifying complex telemetry data into something intuitive and accessible. The development of this dashboard serves as the primary objective of this proposal, enabling data-driven insights into potential differences between drivers and identifying the key aspects that make each driver's style of racing unique. Our design focuses on creating a clustering algorithm to group and analyze our telemetry datasets. Our preliminary analysis initially focused on clustering individual parameters such as throttle, speed, RPM, and braking tendencies. This analysis allowed us to validate our analytical framework and gain a grounded sense of the data's nuances. Building on this foundation, we then transitioned to multi-variable clustering. By analyzing throttle, speed, RPM, braking tendencies, gear, and acceleration (X, Y, Z), the characteristics of optimal F1 racing behavior could be identified. These clustering results have been integrated into an interactive interface that showcases a driver's unique "style" signature, allowing for a direct comparison with other drivers who exhibit similar racing behaviors. Within the scope of this project, we aimed to produce a proof of concept and focus on developing a clustering algorithm that prioritizes accuracy within a narrow, interpretable scope. This simplification of the raw telemetry data aims to make F1 performance analysis more accessible for teams and fans, while creating a more meaningful understanding of the sport. Because clustering driver behavior from raw telemetry is an open problem, our methodology was designed to remain flexible and exploratory. Rather than committing to a single algorithm, we iteratively tested multiple approaches to ensure that the final clustering process is driven by verifiable performance rather than untested assumptions. This refinement of our multivariate methodology ultimately transformed dense racing data into accessible visuals,

empowering fans to explore F1 data for themselves and discover what styles or tendencies lead to a drivers success.

1.0 Problem

In the high-stakes environment of Formula 1, performance optimization is critical for success. In a sport where so many aspects are heavily regulated, data analytics provide "the X factor for every competitor trying to go faster and shatter records" [1]. These data points allow teams to make better decisions before, during, and after the race. These decisions include the design of the vehicle, tire selection and timing, and the optimal strategy for a pit stop. However, among all these insights, one potentially critical factor is often overlooked: the impact of the driver on performance. Specifically, is there a certain way to drive (i.e., a particular driving style) that is better than others? While F1 teams make countless strategic choices, the way the driver interacts with the car can often have a much higher impact on the outcome of a race. Some drivers may choose to gradually steer into a particular turn, while others may brake late and turn sharply. Some drivers may gradually apply the throttle as they come out of a turn, and others may accelerate as quickly as possible. These variations in driver input may seem subtle, but they could potentially have a significant impact on lap speed, tire life, consistency, and ultimately, race outcomes.

While a driver's input has always been a factor of performance in F1, it has often been overlooked during data analysis due to its difficulty to quantify. Driving style has traditionally been treated as a fundamentally subjective feature of the sport. For years, fans and analysts have debated what driving styles there are (if any), how to determine them, and how much influence they have over results. Without an objective methodology for quantifying these inputs, the full potential of driving style analysis to inform performance decisions and strategy cannot be fully realized. In summary, the analysis of driving style has been an unexplored facet of F1, and the knowledge of optimal style could offer teams a competitive advantage over their rivals.

2.0 Objective

This project addressed the currently unquantified concept of "driving styles" using machine learning and analytics on measured F1 telemetry data. This project determined a clustering-based methodology used to categorize all drivers' telemetry driving tendencies on all tracks from the 2025 racing season. Clustering such a massive amount of telemetry poses a significant challenge, so our team developed a data pipeline to process, store, and analyze data, as well as a full-stack web application to present our results.

The central focus of this project was to reveal the intricacies of driving style without relying on predetermined style categories through the development of a data and clustering pipeline. Although fans (and previous work attempting to define driving styles) may describe certain drivers as "passive" or "aggressive," imposing these labels with no empirical evidence could bias the results and reduce the quality of the output if the labels did not correlate with the data. Instead, our approach allowed the data to speak for itself and for users to determine these labels themselves based on provided empirical evidence. The clustering algorithms were also designed to withstand outliers. Laps or turns that deviated too far from a driver's normal behavior caused

unnecessary skew amongst clusters and needed to be discarded. Additionally, the approach had to be scalable. In other words, we had to be able to alter the data pipeline to be functional for other tracks in the Formula One racing season (with the ultimate goal of covering all tracks from the 2025 F1 season).

A secondary but important component of this project was the creation of a telemetry database. Although our scope essentially only required a static output to the final webpage, it took many iterations of algorithms, pre-processing methods, and general experimenting to come up with our final solution. Therefore, to speed up runs, we decided to implement a full-scale database, to ensure the time costly web scraping only must be performed once. Our aim for this database was to provide a structured and query-efficient database, which stores processed telemetry data pulled from the Tracing Insights JSON files, and served as a foundation for any additional features and analysis to be done beyond this project's scope.

Finally, the interface for this tool is a web-based dashboard that allows users to select multiple drivers across teams and view the results of the clustering analysis for each side-by-side for easy comparison. Outputs are visualized through both bar charts and radar charts that summarize the multivariate clustering results. There is a track drop down menu to select the desired track for clustering results and analysis, which also generates a track view that can be found below the analysis. Overall, the dashboard delivers a user-friendly experience that conveys our findings relating to distinct driving styles amongst F1 drivers.

3.0 Background

3.1 Key Concepts

Modern Formula 1 cars generate extremely large and dense telemetry streams, and our analysis relies on this data sourced through Tracing Insights, an open-source repository built on top of the FastF1 Python library. Unlike FastF1, which stores its data in deeply nested objects that are quite difficult to parse, Tracing Insights provides well-structured, easy to interpret JSON files for each driver's telemetry. Tracing Insights was created specifically to make F1 telemetry easier to work with, converting FastF1's nested objects into clean, structured JSON files. Each file contains a driver's telemetry for a single lap and includes complete race coverage dating back to 2018 (although, for this project, we utilize the 2025 data). Tracing Insights also performs fuel corrections over time, accounting for the lap-by-lap reduction in car weight due to fuel burn-off so that speed and throttle related comparisons remain consistent throughout the dataset. This consistent formatting, built-in preprocessing, and open-source nature make Tracing Insights the most accessible and reliable source for F1 data analysis.

Clustering algorithms are used to find patterns in the collected telemetry data. Clustering is an unsupervised machine learning technique that groups data points exhibiting similar behavior. In this project, clustering helps find patterns in how drivers race. These groups are compared to speed metrics to identify faster or slower driving styles and provide actionable insights into optimal race strategies for F1 drivers. To establish a foundation for the project, we began with several standard clustering algorithms as baseline models before moving on to more specialized

techniques. These initial experiments helped us understand the structure and variability of the telemetry data and gave a starting point for evaluating more advanced approaches.

Our approach to these clustering techniques was dynamic and iterative. Because of the complex nature of the data, determining the optimal clustering technique before starting the trial-and-error process was extremely difficult. Because of this, we selected a group of starter clustering algorithms to assess their performance and identify the optimal approach. However, we did not limit ourselves to these algorithms and continued to expand our testing to a variety of different options while searching for the best solution to this difficult problem. For each clustering algorithm, we clustered on the entirety of a driver's laps (discarding outliers) and grouped each singular lap into a specific style. From there, we determined a probability distribution of a driver's tendency towards a specific grouping, ultimately obtaining a driver's "profile" when it comes to telemetry metrics.

K-means clustering was the first algorithm we used to group our drivers. It works by grouping similar data points into clusters. The process starts by selecting initial centroids, then assigning each data point to the closest one. After that, the centroids are updated using the average of the points in each cluster. These steps are repeated until the centroids stop changing.

Hierarchical clustering is the next clustering algorithm we are tested, and it groups data by building a tree-like structure called a dendrogram. It starts without using a preselected number of clusters and instead makes each point its own cluster, then repeatedly merges with the closest cluster based on distance or similarity. This algorithm creates a hierarchy that organizes data into progressively larger, more meaningful clusters.

Gaussian Mixture Model (GMM) is another clustering approach we explored. This algorithm represents the data as a combination of several normal distributions. Instead of placing each point into a single group like K-means, GMM gives every point a probability of belonging to different clusters. Because each cluster has its own mean and covariance, GMM can form more flexible groups that reflect laps blending multiple tendencies. We initially expected it to perform well due to the conversion of our data to z-scores in the preprocessing stage. However, its reliance on normal distribution assumptions resulted in poor clustering performance, and it did not effectively cluster our data.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) was also explored as part of our baseline batch of clustering algorithms. Unlike algorithms that require a predefined number of clusters, DBSCAN identifies groups by locating dense regions of data points and discarding everything else as noise. This makes it well suited for busy telemetry, where laps may contain irregularities such as lock-ups, unusual traffic, or mechanical issues which won't be present in other laps. By automatically detecting outliers and allowing clusters to be arbitrary in shape, DBSCAN provides a flexible approach to clustering not available in algorithms like k-means and hierarchical clustering.

Principal Component Analysis (PCA) is a data preprocessing technique that reduces the dimensionality of data. In datasets with large amounts of features (in our case, roughly 3500), it can compress these down into often just 5-10 features. It does this by applying the PCA algorithm, which heavily relies on linear algebra. Essentially, it is a way to drastically reduce the

dimensionality of the data that is being clustered upon, thus producing much more insightful clustering results.

3.2 Related Work

There are several online tools that work with Formula 1 data, but most focus on visualizations instead of clustering driver behavior.

Tracing Insights [3] gives users the ability to look at lap comparisons that show speed, throttle, and braking data. While it offers clear visuals, it is often slow and does not summarize data or classify driver styles. Users can see the information but must interpret it on their own.

The Formula 1 Telemetry Tool [4] on GitHub is another open-source project that lets users visualize telemetry from races. It is helpful for plotting things like throttle and brake pressure but only serves as a viewer. It does not analyze or group drivers based on performance or behavior.

F1nsight [5] provides a broader set of race statistics, including braking zones, throttle control, and lap performance. It gives useful comparisons between drivers but still focuses on individual statistics rather than patterns across drivers. It does not use clustering or machine learning to find similarities or differences in style.

The Formula 1 website [6] also provides driver and team information in a clear and professional format. It focuses on standings, records, and short summaries but does not allow comparisons using telemetry data.

The Formula 1 Telemetry Insights Visual Dashboard builds on these ideas by combining visualization with data analysis. Instead of showing raw data, we use clustering algorithms to group drivers based on shared racing patterns. This approach allows users to see how driving style impacts performance and offers a new perspective that current tools do not provide.

4.0 Design

4.1 Use Cases

The dashboard has one user type. There are no login-protected views, admin roles, or permission tiers. Every user sees the same interface and has access to the same controls.

4.1.1 Track Selection

The first thing a user does is pick a track. The dropdown in the map visualization panel lists all 24 circuits from the 2025 season, each labeled by city, country, and circuit name. Once a selection is made, the panel renders a map image of that circuit and the lap ingestion summary updates to show total laps ingested, usable laps, and dropped laps. A cluster overview section also populates, where the user can select any of the five clusters from a dropdown to see a horizontal bar chart showing how that cluster's average telemetry deviates from the overall mean across eight metrics: RPM, speed, throttle, brake, gear, and the three acceleration axes.

4.1.2 Driver Selection

A user can click any driver button in the header to open that driver's statistics block. Drivers are grouped by team, with both team members shown as clickable buttons on each team card. The header scrolls horizontally on smaller screens so all ten teams stay reachable. Clicking a driver highlights their button and opens their statistics block below. Clicking the same driver again closes it. Multiple drivers can be open at once. Their blocks stack in the order they were opened, with horizontal scrolling available in the statistics panel if enough are active to overflow it. Switching tracks while drivers are already selected refreshes all open blocks for the new circuit without clearing the driver selection.

4.1.3 Statistics Exploration

Each driver's statistics block shows their name as a header and contains two charts: a stacked bar chart and a radar chart, both labeled "Lap Count by Cluster" and "Cluster % Distribution" respectively.

The stacked bar chart breaks down that driver's laps by cluster across a single horizontal bar, color-coded across all five clusters. Red represents Cluster 1, blue Cluster 2, yellow Cluster 3, teal Cluster 4, and purple Cluster 5. The length of each color segment corresponds to how many of that driver's laps fell into that cluster, giving a quick read on which driving styles showed up most often at the selected circuit.

The radar chart displays the same information as a percentage distribution across a five-axis shape, with each axis corresponding to one cluster. Rather than raw lap counts, it normalizes the data so the polygon reflects the relative weight of each cluster in that driver's profile. A driver whose laps concentrated heavily in one or two clusters will show a skewed, elongated shape, while a more evenly distributed driver will produce something closer to a balanced pentagon. Together the two charts give both an absolute count and a proportional view of the same underlying data.

Outside of track and driver selection, the dashboard has no other interactive surfaces. No forms, no accounts, and no way to adjust clustering parameters at runtime. All model training runs offline before the page loads, and the site serves results statically.

4.2 Requirements

This will be split into three sections. Functional requirements, which discusses each aspect of our pipeline necessary for functionality in a very broad overview. Interface requirements, which does the same but for the frontend aspect of our project. And finally, the performance requirement section, which discusses the performance considerations needed when approaching this project.

4.2.1 Functional Requirements

The requirements for this project can be split into a few broad categories. Each of these requirements is outlined in the schedule, which gave us a defined approach for implementation.

Data Ingestion. For this requirement, we needed our program to be able to obtain the telemetry data files from the TracingInsights GitHub. These are divided by year of F1 data, then by race,

then by stages (Q1, Q2, Q3, race (we will only be using race)), then by driver, and finally by lap. We needed the ability to intake the data from these JSON files and store them for further manipulation.

Database for telemetry. The database needed to be able to run queries quickly to retrieve telemetry data and be able to handle a massive number of records due to the size of our telemetry dataset. This was the focus of our performance requirements and was the primary source of a bottleneck when dealing with this amount of data.

Initial data processing. This occurred directly after the telemetry is retrieved from the database, and provided normalization and standardization of the telemetry, as well as resampling based on fixed distance points on the track to facilitate Euclidean distance comparisons.

Clustering-specific preprocessing. We dedicated significant time in the schedule to exploring methods of reducing and refining telemetry data before it is passed into the clustering algorithms. This was done through methods such as PCA and DTW (Dynamic Time Warping). The goal was to improve the quality and interpretability of the resulting driver style groups.

Clustering algorithm. Our implementation of this involved experimenting with multiple algorithms described in Section 3.0 Background, although we did not limit ourselves to these models. Our clustering algorithm needed to be able to produce clear, grounded results that highlight differences in driver's tendencies when evaluating a particular metric. It should give a fuzzy output, meaning it gives a distribution of style and behavior instead of lumping a driver into a singular and potentially limiting style (this is why laps were ultimately chosen to cluster upon, because a lap distribution fulfills this requirement).

Cluster validation. The outputs of our clustering algorithm needed to be analyzed to compare results across the different approaches we used. We needed to be able to, through a variety of different methods (visualizations of clustering, silhouette scores, Dunn index, Davies-Bouldin Index, etc.) determine the validity and accuracy of each clustering methodology. This evaluation was critical to determining the ultimate best way to cluster the data.

Style analysis. The final clusters from our clustering algorithm were not manually labeled, since cluster meanings changed from race to race and labeling them could be misleading. Instead, we focused on showing the results through a cluster overview visualization. This approach shows the distribution of telemetry features within each cluster, allowing users to interpret differences in driver behavior without forcing fixed labels. By displaying the clustered metrics speed, throttle, RPM, and acceleration side by side, users can see how clusters differ without oversimplifying the data.

Server. The system uses a server that runs both the front-end web application and the telemetry database. This server needed to be accessible to both the capstone team and General Motors.

Website. To display the results of our clustering and analysis, the system needed to provide a front-end interface with interactive visualizations and relevant statistical summaries. This front-end needed to be visually appealing, insightful, and useful to F1 teams, presenting driver-style insights in a clean and intuitive manner.

4.2.2 Interface Requirements

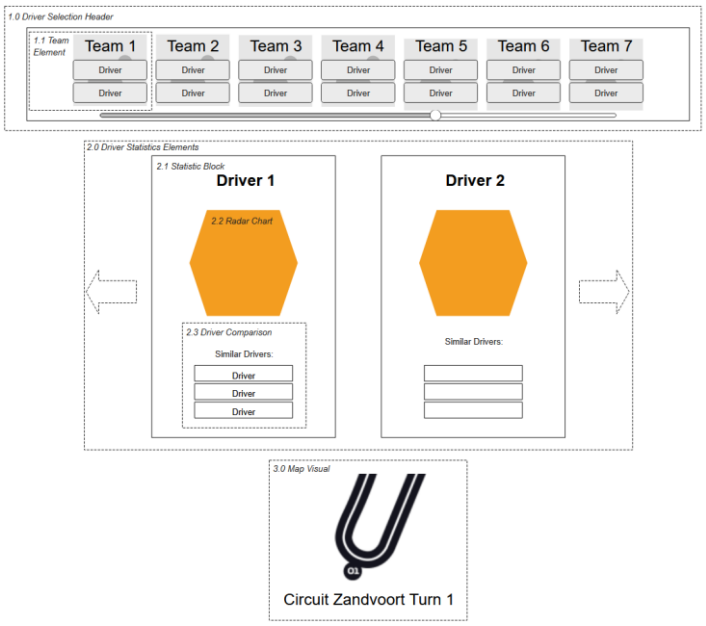


Fig. 1 Visual Dashboard Preliminary Mockup

1. **Driver Selection Header**
   The driver selection header is located at the top of the webpage and contains every team found within the Tracing Insights database. The main header displays team elements in a single row, only visualizing a limited portion of the teams with horizontal scrolling to view the others when displayed on smaller screens.
   1.1. **Team Element**
   A team element has the name of the team at the top, two buttons that contain the names of that team's drivers below the team's name, which update based on which track has been selected, and the team's colors as the background. When a driver button is clicked by the user, it is highlighted and a statistics block for that specific driver appears under the driver selection header.
2. **Driver Statistics Elements**
   Found below the driver selection header, this area only populates with statistic blocks when a driver's button has been clicked on. If that driver's button is clicked again, the block disappears. When multiple drivers are selected, their blocks will appear vertically stacked (in rows of two, if the user's screen is big enough). This allows a user to select as many drivers as they would like and still be able to view their information.
   2.1. **Statistic Block**
   A driver's statistic block originally contained the driver's name at the top, a radar chart comprising the data generated by the clustering algorithm, and then three drivers found to be most similar to the driver of this statistic block. During implementation, we decided to use a stacked horizontal bar chart for clustering information, as well as a single radar chart for each driver that represents a summary of their clustering information.

2.2. **Radar Chart**

The radar chart takes in the percentile data created by the clustering algorithm for the driver and creates a visual showing what different driving styles the driver falls into. A polygon is created around a central origin point with the number of vertices equal to how many different styles the clustering algorithm finds the driver to be a part of. Each vertex has an equal distance from each other but has a varied distance from the center origin point equal to its percentile value for that specific driving style. This will then create a unique shape for each driver, showing which driving style groupings they are dominant in.

2.3. **Cluster Overview**

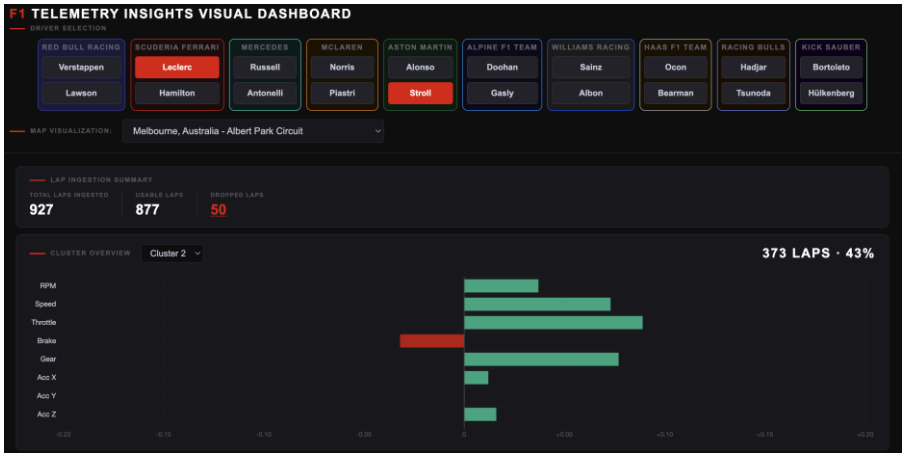
The Cluster Overview shows, for each race and cluster, how each of the single telemetry categories for that cluster vary from the races average. A dropdown menu allows you to select from each of the five clusters to view the relevant statistics.

2.4. **Lap Ingestion Summary**

The Lap Ingestion Summary shows for each race how many of the laps were included in the clustering results, as well as the number of laps dropped, and the total number of laps ingested. When hovering over the number of dropped laps, a short description appears to show why those laps were dropped.

3. **Map Visualization**

The map visual is comprised of a leaflet map and a drop-down list. The drop-down lists the tracks from the 2025 F1 season. When one of these is selected, the visual updates to show the map of whichever track is selected. When this happens, all information present in the driver statistic blocks updates to display information found for that specific racetrack.



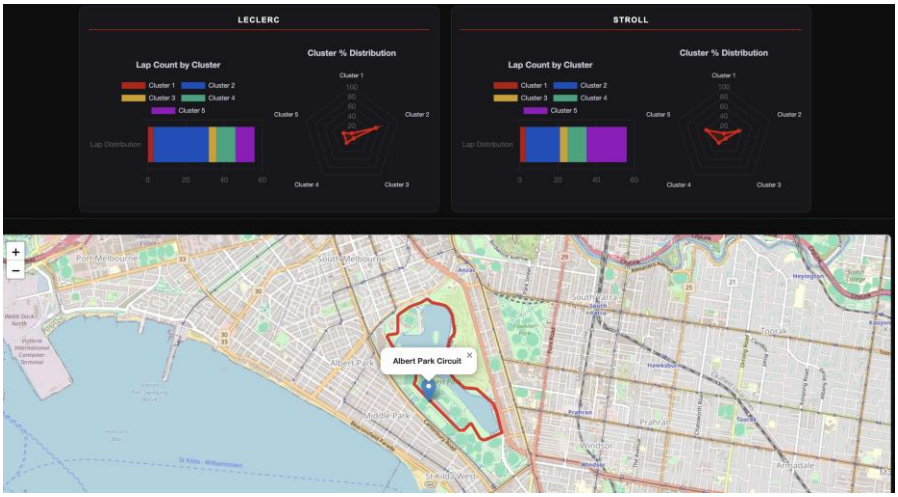


Fig 2. Visual Dashboard Final Version

4.2.3 Performance Requirements

Performance is an important metric to keep in mind, due to the massive amounts of telemetry data being stored in the database, and the high potential for massive SQL query times. For the entire Formula One 2025 racing season, there are over 18 million records. This presents a massive issue for the technique of scanning over the entire table for select statements, and so additional methodologies had to be developed to mitigate the massive waiting times on queries. Our system needed to allow for quick SQL select statement execution and to retrieve data for repeated clustering trials.

While TimescaleDB was initially installed for this project, we found PostgreSQL fully sufficient for our architecture. As TimescaleDB is simply an extension of PostgreSQL, we will be referring to our database as PostgreSQL for the remainder of this paper.

For the front-end, dynamic runtime performance is not a concern. This is because all the training is executed prior to any user interaction, allowing the visual dashboard to serve as a static display of insights realized beforehand. These insights are fetched from static JSON files with relevant outputs from our clustering analysis.

4.3 High-Level Architecture

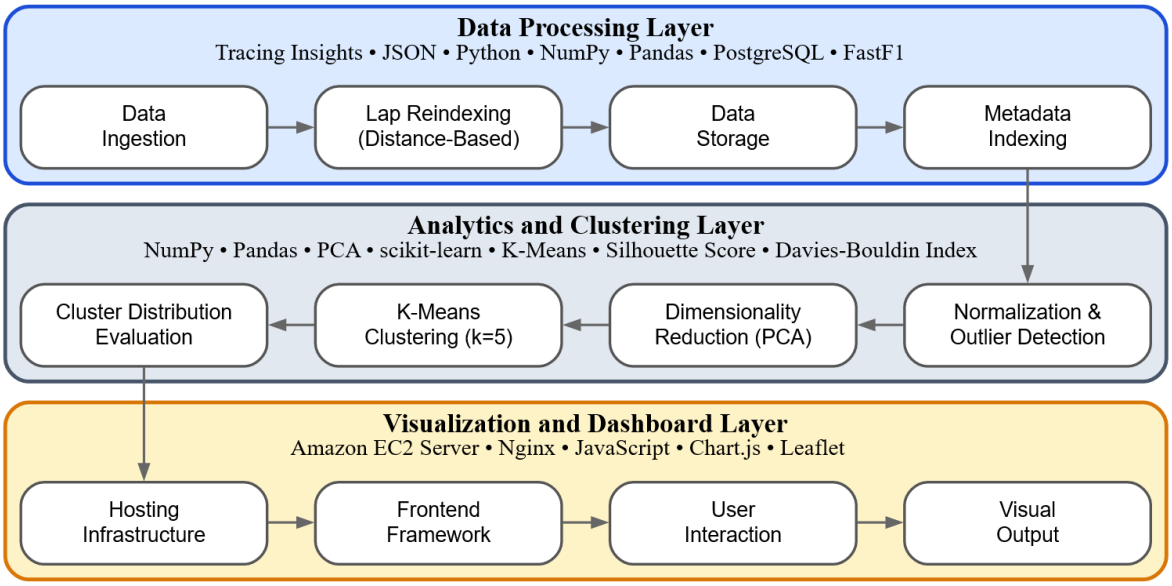


Fig 3. High-Level Architecture of the System

The system architecture is organized into three layers.

Data Processing Layer. This layer takes every JSON file from TracingInsights and processes it. It breaks the laps down into single measurements, and stores two records for each one. One in a metadata table for faster querying, and the other in the telemetry table for the storage of the actual data.

Analytics and Clustering Layer. This layer converts the massive number of records provided and feeds them to a clustering algorithm, transforming the data into driver styles. The data preprocessing is also done on this layer, and is utilized to improve clarity of clustering results.

Visualization and Dashboard Layer. This layer provides visualizations of data and statistics realized through this process and provides both technical and non-technical F1 enthusiasts with interesting insights into the drivers in the sport.

4.4 Detailed Architecture

Data Ingestion. The data ingestion process is done through a simple Python script. This script utilizes the requests library, which allows downloads of the JSON files stored in the TracingInsights repository through building and executing links through the script scrape.py. Once the script is run, all the data from a specified race is stored locally in JSON files. These files can then be parsed into the PostgreSQL database by running the Python script database_intake.py, which has an incredibly long runtime, but only must be executed once.

Database for processed telemetry. The database layout is thoroughly outlined in 4.5 Data Management and Storage. PostgreSQL was used for this task (although the extension TimeScaleDB was also installed but never used).

Initial data processing. Stored in the data_analysis folder. This step is handled in its entirety in the RaceData class, which is stored in race_data.py. First, the data is retrieved from the PostgreSQL database using the TelemetryDatabase from query_db.py. It is then reindexed to fit each lap on a grid of 500 points. This is required for Euclidean distance comparison, as equivalent indexes are vital, and the telemetry is measured at varying and inconsistent intervals. The data is then interpolated to fit on the grid using native Pandas dataframe methods. The data is then normalized. This is done by retrieving each driver's local minimum and maximum values for each metric, and then, using these values, we can convert each data point to a consistent 0-1 scale. This normalization per driver serves the purpose of normalizing mechanical differences, allowing for clean comparisons that only encompass differences between drivers and not cars. Also, consistent clustering requires every metric to be on the same scale for equal weighting. The minimum and maximum values used for normalization are recomputed in 5 lap chunks for each driver. This was done to prevent mechanical alterations throughout a race damaging a driver's calculated scale. Finally, we need to detect outlier laps. This was done by calculating the average speed of all laps during a race for a driver and dropping all laps below a 1.5 x IQR range of this metric. IQR serves as an incredible metric here, as we found that this consistently detects anomalous laps with incredible precision, allowing us to not be limited to hardcoding speed values (which does not translate algorithmically to other tracks).

Clustering-specific preprocessing. PCA is the main clustering-specific preprocessing technique we implemented. Upon initial implementation of PCA, both the silhouette score and the Davies Bouldin index went down, indicating less definitive clusters being found. However, we were able to produce much more discernable cluster distributions for the first time (i.e., the drivers did not have identical lap distributions across the various clusters, and cluster lap amounts were much more consistent). This indicated true driving behavior differences being detected for the first time, and is why we have kept it in our pipeline. The PCA algorithm is applied using sklearn (sklearn.decomposition).

Clustering algorithm. Our finalized clustering pipeline is entirely grounded in K-means. It has been implemented entirely through sklearn. We found it to be the highest performing clustering algorithm, and throughout our many iterations of our algorithm, we found the most definitive results were with it. Hierarchical was implemented as well, but found much worse lap distributions. DBSCAN never made it past the planning stage, as with more research, we found that it does extremely badly with similar data (and with all F1 laps being very similar to each other, we decided against its implementation). Gaussian Mixture Model (or GMM) found identical results to k-means when using its hard labels, and we could not feasibly implement logic for the soft probabilistic clustering in our timeline, so we also decided against it. We ended up landing on K-means with a cluster size of k = 5. This was done through careful analysis of both resulting silhouette scores, Davies Bouldin index scores, lap distributions between all drivers, and the amount of outlier clusters our results found. K = 5 was determined to be optimal, as it most consistently found driver lap distributions that were interesting, and had at most 2 outlier groups (with higher cluster counts just finding more outlier groups, and lower cluster counts destroying a consistent lap distribution).

Cluster validation. Cluster validation will be a critical stage for determining the effectiveness of each clustering methodology. Clustering is incredibly complex, and requires constant experimentation with cluster count, data preprocessing techniques, clustering algorithms, and

more. Our goal is to find the optimal combination of these techniques, but this requires the ability to judge and interpret our outputs. The evaluation process relied on a combination of metrics and visualizations to compare outputs across models. For quantitative metrics, we calculated several values to help us determine the success of clustering. Silhouette score implemented using sklearn.metrics.silhouette_score will measure how similar each lap is to the assigned cluster compared to other clusters. The Davies-Bouldin Index is also available through sklearn.metrics.davies_bouldin_score, and will evaluate how similar each lap is to its assigned cluster compared to other clusters. We also investigated the lap distributions of clusters per driver (which was ultimately a part of the deliverable data visualization), and it helped us determine cases where scores could be misleading. The scores are much higher when clusters found outlier groups exclusively, and these outputs were useless for our use case. Finally, we investigated using some simple Matplotlib visualizations that helped us determine what clusters were more in depth.

Style analysis. This step is largely left up to the user of our dashboard. We aim, through our visualizations and clustering outputs, to give the user a tool to help develop insights and establish patterns across the F1 driver field. Our analysis helps define an entry point into "driver styles," but more analysis is required by the user to find definitive patterns and insights from our tool.

Website Architecture. The architecture of the dashboard is built around a JavaScript based web application that retrieves the clustering results from the Amazon EC2 instance and transforms them into a visualization-ready format for display on the website. The system is comprised of a single HTML page, utilizing nginx to extend Javascript functionality, serve the page over HTTPS, and map incoming domain requests to correct file paths located on the EC2 instance. Accessed through a paid domain provided by Porkbun [15], a user can connect securely instead of through an open IPv4 address like nginx inititally provides. The function of the website is to simply select drivers that have raced on specific tracks, and then pull and visualize their generated clustering data from stored JSON files. The clustering results are stored as a set of pre-generated JSON files organized by race name on the server, including Track_Summary.json, driver_distribution.json, and cluster_overview.json. When a user selects a race, the frontend fetches only the files relevant to that selection rather than loading everything upfront, keeping the page lightweight and non-dynamic. Because clustering results are fixed and do not need to be recomputed on demand, serving these files through nginx is significantly simpler and more reliable than maintaining a live database connection or API backend, and it means the site has no moving backend parts. Visualizations generated from these clustering results are built using Chart.js, rendering statistical graphs and charts instantly. The other main visualization is what track is selected, which is created with Leaflet.js that utilizes OpenStreetMap to display the real-world location of a racetrack.

Deployment:

- **Hosting:** The dashboard is hosted on an Amazon EC2 instance running Ubuntu, chosen for its reliable uptime and straightforward configuration for static file serving. The clustering result JSON files and the HTML page are stored directly on the instance, organized under a web root directory that nginx is configured to serve from.

- **Serving: n**ginx was installed on the instance to act as the web server, listening for incoming requests on both port 80 (HTTP) and port 443 (HTTPS) and mapping them to the correct files on disk. HTTPS was enabled by using Let's Encrypt via Certbot, which automatically obtained and installed an SSL certificate on the instance and configured nginx to use it.
- **Domain Access:** The domain https://uark-team6-f1-visual-dashboard.site/ was registered through Porkbun, a domain registrar, and pointed to the EC2 instance by configuring a DNS A-record with the instance's public IPv4 address as the target. This allows users to reach the dashboard through a readable domain name over a secure HTTPS connection, rather than directly through a raw IP address. Once a request arrives at the domain, it is routed through AWS's network to the EC2 instance, received by nginx, and the appropriate files are returned to the user's browser.

JavaScript:

- **Rendering UI components**: This includes the Driver Selection Header, Statistic Blocks, and Track Map Visuals. Each component is designed to update dynamically based on user input and reflect roster changes during the 2025 F1 season.
- **Managing user interactions**: The frontend handles clicks, selections, and track updates, ensuring that the page updates seamlessly in response to user actions.
- **Pulling Clustering Data**: When both a track and driver are selected, the frontend fetches the relevant pre-generated JSON files from the server: *driver_distribution.json* for per-driver cluster breakdowns, *cluster_overview.json* for race-wide cluster deviation summaries, and *track_summary.json* for information about lap ingestion for the race.
- **Interactive radar chart rendering**: Using Chart.js, the frontend renders a horizontal stacked bar chart showing each driver's lap count distribution across clusters, and a radar chart showing their percentage distribution across clusters. A separate diverging bar chart visualizes the telemetry deviations of the found clusters for that race from the average across eight different metrics: RPM, speed, throttle, brake, gear, and three axes of acceleration. All charts support hover tooltips displaying precise values.

HTML:

- **Structural Framework:** Defines the 3-panel composition of the website, organizing the static elements and providing injection points for the Javascript to implement visuals, all while updating dynamically when in use by a user.
- **Styling and theming**: Embedded CSS within the HTML file handles all visual styling of the dashboard. The design follows a dark theme built around the F1 color palette, using near-black backgrounds with red accents to match the aesthetic. CSS is also used to define the team-specific color schemes applied to each team block in the driver selection header, as well as responsive layout behavior such as horizontal scrolling for the driver statistics blocks and chart rows.

Data Flow Overview:

1. The user opens the website and the frontend loads the track selection dropdown and map panel at their default states, with the driver selection header showing the opening-round 2025 roster.
2. The user selects a track from the dropdown. The frontend fetches *Track_Summary.json* to display lap ingestion statistics, fetches *cluster_overview.json* to populate the cluster deviation chart, updates the Leaflet.js map to display the real-world circuit outline via OpenStreetMap, and then updates the driver selection header to reflect the correct roster for that race.
3. When a driver is selected, the frontend fetches *driver_distribution.json* for that race and locates the entry matching the driver's three-letter abbreviation. The cluster lap counts and percentages are extracted and rendered into the stacked bar chart and radar chart inside that driver's statistics block.
4. Additional driver selections load new statistics blocks in parallel without affecting existing ones. Switching to a different track clears all displayed data, re-fetches the three JSON files for the new race, and refreshes the charts of any already-selected drivers with the new race's data.

4.5 Frontend Implementation Adjustments

4.5.1 Web Server

To host the deliverable webpage, clustering algorithms, and database, a paid Amazon Web Services (AWS) account was created to run and deploy an Amazon EC2 instance. Depending on the type of instance created, the hourly rate charged changes drastically, but Amazon provides up to $200 in credits that will be used before the card attached to the account is charged. Utilizing this, we were able to develop and test without incurring operational costs, adjusting our instance’s processing power and memory usage as needed. When the database needed to be clustered, the instance would be scaled up temporarily and then scaled back down when only the website and storage needed to be hosted.

To prepare the instance to host the EC2 instance, a security group was created that allowed SSH, HTTP, and HTTPS connections from any source, which allowed all members of the team to open the website through an HTTP connection to its public IPv4 address. Opening the SSH port allowed members to remotely connect to the terminal of the EC2 instance and interact with its internals, which meant implementation of the frontend, as well as setting up the database, could begin. Rather than serving the site over a raw IPv4 address, a domain was registered for the final deliverable through Porkbun, allowing the site to be accessed securely over HTTPS at https://uark-team6-f1-visual-dashboard.site/.

4.5.2 Frontend Website

Remote connecting to the EC2 instance, a GitHub pipeline was set up to document frontend updates and pull from the repository when needed. Once the Elastic IPv4 address was set up for the EC2 instance, and nginx was configured to work with it, a basic index HTML page was created and would be accessible as long as the instance was running. From here, the basic outline of the dashboard was created as described by the interface mockup in Figure 1.

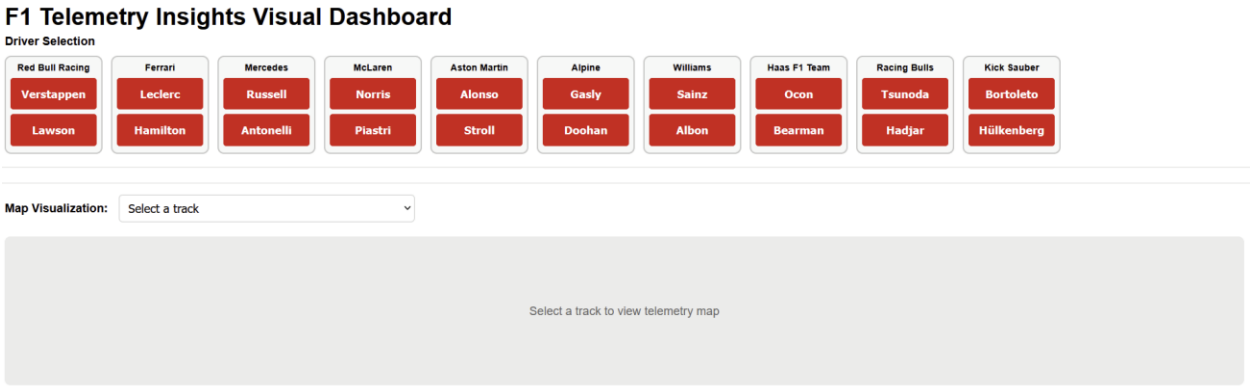


Fig 4. Pre-Developed Initial Visual Dashboard

As seen in Figure 4, the webpage is divided into 3 sections: a Driver Section with 10 teams containing 2 drivers each, an empty section that will be populated with a driver's statistics when they are selected, and then the map visualization where a track can be selected from a drop-down menu. Note that when a track is selected, the possible drivers to select will update to reflect who had driven the track during that race in the 2025 F1 season.

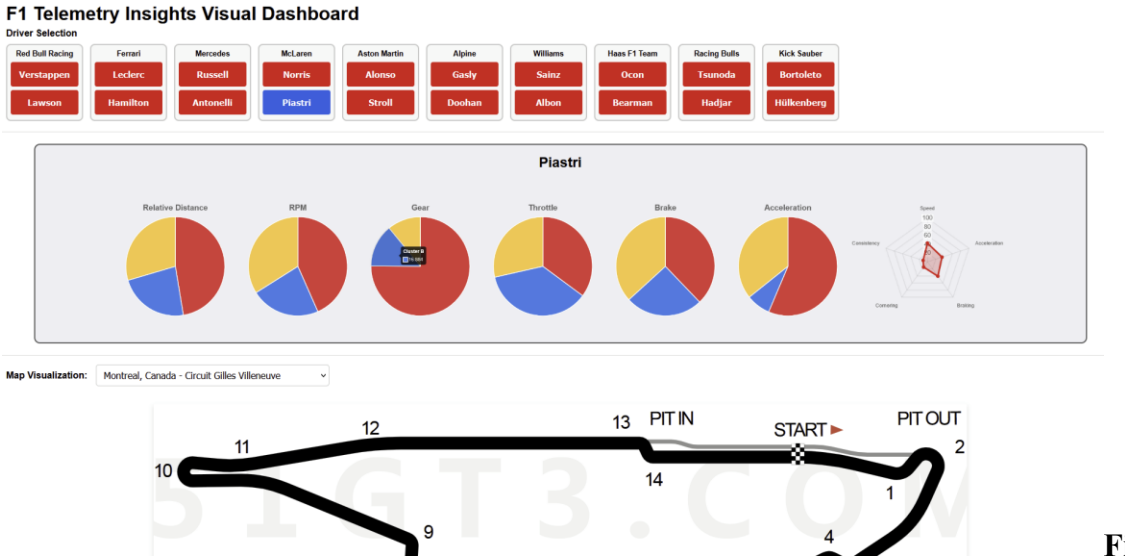


Fig 5. Selecting a Track and Driver

During the implementation of the Driver Statistics Element, some changes were made from how the mockup displayed the interface. Instead of statistic blocks populating to the side of other blocks, they now stack vertically to make use of the infinite scrolling capabilities of a typical webpage. As more information was gathered from the clustering sprints, it was realized that it would be possible to display the data gathered for each metric as its own pie chart, while keeping the radar chart for when multivariate clustering data was developed. As seen in Figure 5, these

pie charts are made up of 3 different clusters, showcasing the distribution of clusters that a driver falls into for that metric. A user can determine which color is which cluster by hovering their mouse over a section of the pie chart, which will then display the name of that cluster through a tooltip. Note that this data depicted is not the actual data that has been generated by the clustering algorithms.

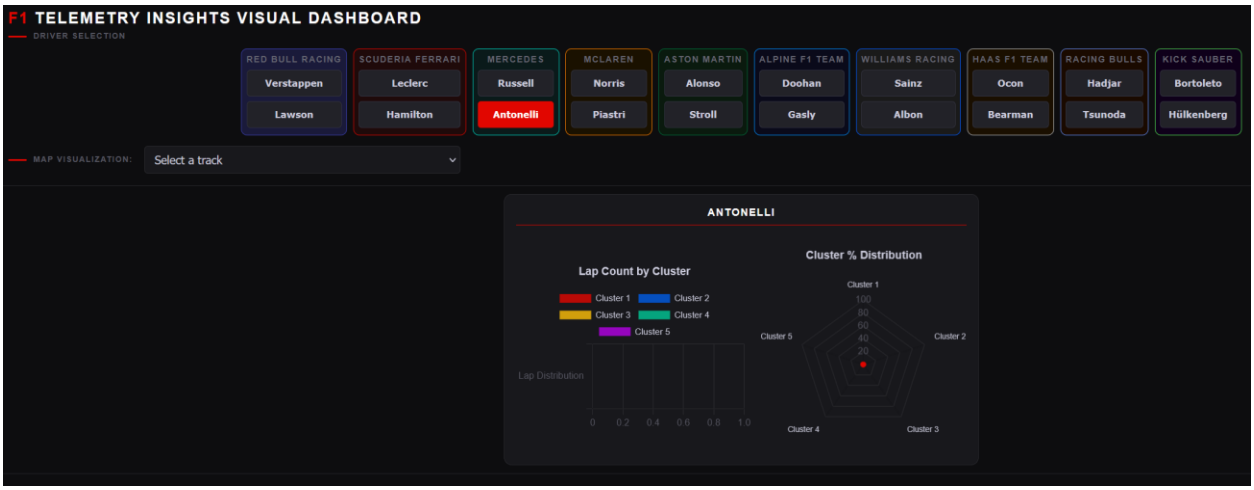


Fig 6. Final Frontend Visual Style

As the frontend was further developed, a consistent visual theme was established to create a more cohesive and final deliverable. The original mockup used a plain light interface with minimal styling, but a dark theme built around the F1 color palette would be utilized for the final implementation. Near-black backgrounds with surface panels in a slightly lighter shade create depth through layering, and F1 red was used as our primary accent color to draw attention without overwhelming the interface. Team blocks in the driver selection header were each given their own background tint and border color like their real-life team coloring. Finally, the Typography was kept tight and uppercase with wide letter spacing on labels and section titles so that everything was clear and legible to the user. Figure 6 also shows how the website was slightly rearranged from the original mockup, in that the track selection dropdown menu was brought up to be underneath the driver selection header. During implementation it became clear that this original ordering from the mockup was unintuitive, as a user arriving at the page would see driver buttons and empty statistic blocks before understanding that a track needed to be selected first. Also, as more drivers were selected, the bottom of the page would move further down, making it a nuisance to change tracks as the dropdown kept moving.

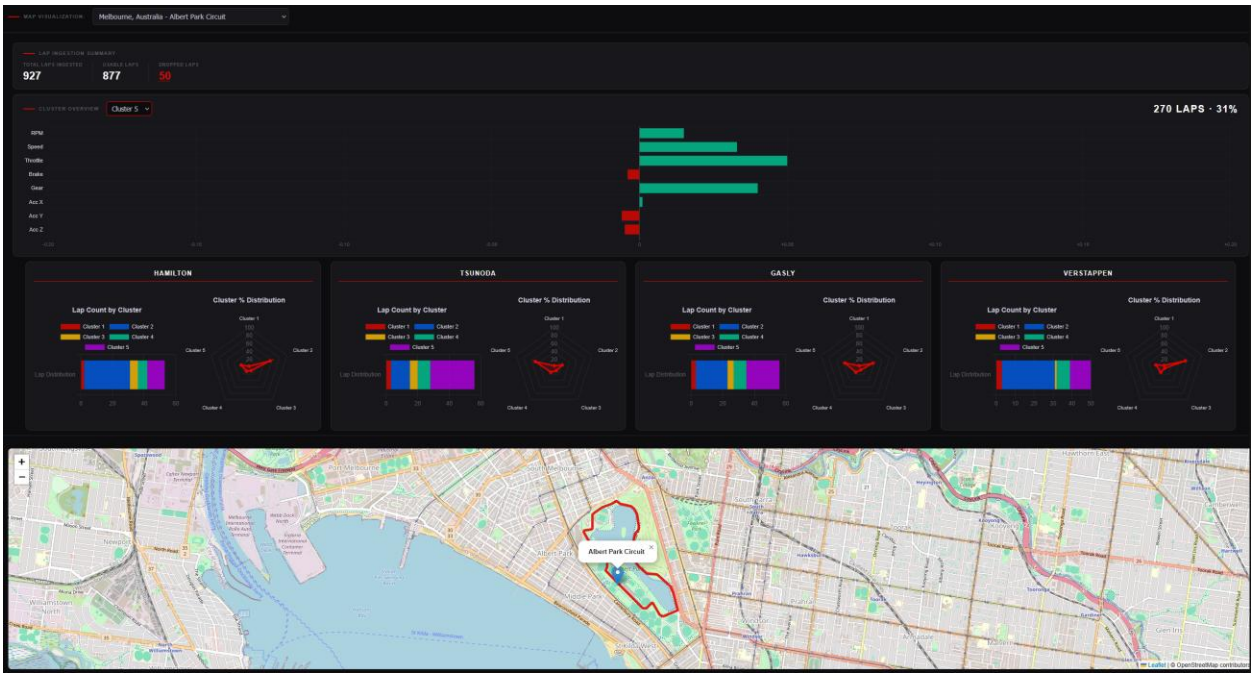


Fig 7. Final Frontend Statistical and Map Display

As the actual clustering output became available, the visualization strategy was revised significantly from the preliminary plan. The single variate pie charts that were shown in the earlier mockup were removed, as the final clustering approach produced 5 multi variate clusters that could be assigned per lap that was based on the metrics of a driver during that lap. In their place, two charts now populate each driver statistics block: a horizontal stacked bar chart showing the raw lap count each driver spent in each of the five clusters, and a radar chart showing the percentage distribution of those same clusters, giving both an absolute and relative view of each driver's driving style for that race. A summary of how many laps were ingested into our clustering algorithm for this race was also added, with the ability to see why certain laps were omitted. Additionally, a cluster overview panel for the given race was added, displaying a diverging horizontal bar chart that visualizes each cluster's average deviation from the mean baseline across the measured metrics of a driver. A dropdown allows the user to switch between clusters, making it possible to understand what each cluster actually represents in terms of driving behavior before they begin to make comparisons between drivers.

4.6 Data Management and Storage

All the data in this project is extracted from the Tracing Insights open-source GitHub repository for the 2025 season. This data is stored in nested, well-structured JSON files, and is incredibly easy to scrape using the Requests Python library. To leverage this data more efficiently, a SQL database has been created. Structuring the data in a database will allow us to conduct queries directly on the data, allowing for iterative runs of our clustering algorithms to determine the optimal method. The database schema that is being used is shown below.

Originally, our database consisted of two massive tables, one for telemetry and one for metadata, both totaling over 18 million records each. This division was done for the sake of faster queries, but, because of the number of records, we discovered quickly that queries were incredibly inefficient. It took us upwards of 10 minutes to get all 20 drivers' telemetry data for a single race. We decided to improve our database in a couple of ways. We started by splitting the data into individual tables for each race. This transformed our database into 48 tables instead of two but drastically reduced the record count in each table needed to sort through per query (roughly 750,000 per table instead of the original 18 million). We also introduced indexing by driver and lap in the race lap data table through a binary tree index. This caused the biggest performance improvement we had, dropping our per driver time from roughly 2 minutes all the way down to roughly 15-20 seconds, which makes sense given the logarithmic nature of binary tree lookup complexity.

Table 1: Race Lap Data

This table organizes telemetry data by a specific driver, lap, and race. It is named as *metadata_***_2025*, where *** represents a three-letter abbreviation for the race name.

	Column	Data Type	Description
PK	lap_data_id	INT	Unique identifier for each record
	driver_id	CHAR(3)	Unique 3-character identifier for each driver
	lap	INT	Lap number
FK	tel_index	INT	Foreign key referencing the Telemetry table, which holds the actual sensor/time series data for a specific lap

Indexes:

*"metadata_***_2025_pkey" PRIMARY KEY, btree (lap_data_id)*

*"idx_metadata_***_2025_driver_lap" btree (driver_id, lap)*

Foreign-key constraints:

"fk_telemetry" FOREIGN KEY (tel_index) REFERENCES telemetry_***_2025(tel_index)

Table 2: Telemetry Data

This table stores the telemetry data extracted from Tracing Insights. It is named as *telemetry_***_2025*, where *** represents a three-letter abbreviation for the race name.

	Column	Data Type	Description
PK	tel_index	INT	Unique identifier for each record

	time	FLOAT	Time (in seconds) at which the data was recorded
	distance	FLOAT	Distance (in meters) from the starting line
	rel_distance	FLOAT	Normalized distance (in meters) from the starting line (0-1 scale)
	track_coordinate_x	FLOAT	x coordinate of the vehicle's position on the track
	track_coordinate_y	FLOAT	y coordinate of the vehicle's position on the track
	track_coordinate_z	FLOAT	z coordinate of the vehicle's position on the track
	rpm	FLOAT	Revolutions per minute
	gear	INT	Describes what gear the vehicle is in. Can be 1-8
	throttle	FLOAT	0-100%
	brake	BOOL	Whether the driver is applying the brake or not
	drs	BOOL	Drag reduction system. Can either be on or off
	speed	FLOAT	km/h
	acc_x	FLOAT	m/s^2
	acc_y	FLOAT	m/s^2
	acc_z	FLOAT	m/s^2

Indexes:

"telemetry_***_2025_pkey" PRIMARY KEY, btree (tel_index)

Check constraints:

"telemetry_***_2025_gear_check" CHECK (gear >= 0 AND gear <= 8)

"telemetry_***_2025_throttle_check" CHECK (throttle >= 0::double precision AND throttle <= 100::double precision)

5.0 Development Plan

5.1 Tasks

Database Tasks.

Completed:

- Acquire database (PostgreSQL + TimescaleDB), install database into server, and check accessibility
- Setup database for data ingestion
- Setup Python scripts for collecting Tracing Insights data from their main Github repository.
- Link up database to webserver
- Setup Python scripts for converting clustering data into database

Clustering Tasks.

Completed:

- Data ingestion scripts. Pull JSON data from the GitHub repository and store it in a PostgreSQL database.
- Statistical preprocessing
  - Distance-based alignment, so all laps share a consistent index.
  - Resampling to fixed length per zone.
  - Standardization (min-max normalization) applied per driver.
- Clustering Sprint 1
  - Infrastructure for clustering being set up. K-means model being experimented with.
    - These models will be implemented in a modular fashion to allow future re-testing with different data inputs. Evaluation measures will also be introduced in this sprint to assess the quality of clustering.
      - Silhouette score
      - Davies-Bouldin Index
- Clustering Sprint 2
  - This sprint will be dedicated to data engineering and preprocessing, and comparison of outputs from Sprint 1. This will require a solid evaluation pipeline already, which will be further developed in this sprint. Methods to tested include:
    - PCA
    - DTW
    - Lap filtering
  - As with Sprint 1, tasks are loosely defined to allow for trial-and-error testing and additional research into alternative methodologies as needed.

Unfinished:

- Clustering Sprint 3
  - Implementing the following models
    - Hierarchical clustering
    - Gaussian mixture model
    - DBSCAN
  - Experimenting with DTW preprocessing
- Analysis Stage of driver styles

- Once clusters are defined, we will conduct a final analysis stage to interpret and label the identified driving styles. Insights will be derived by comparing clusters against performance-related metrics, including:
  - Speed insights (i.e., fastest vs. slowest styles)
  - Tire wear insights (i.e., how tire wear affects driving style)
  - Acceleration insights (i.e., how acceleration differs across styles)
  - RPM insights (i.e., how RPM differs across styles)
- These insights will ultimately be integrated into the front-end for visualization and reporting.

Website Tasks.

Completed:

- Set up the initial website and connect to the server. We first set up a blank website to test server accessibility, then added initial UI elements to the main webpage of the website. This was an empty outline of our dashboard split into three windows for driver selection, driver statistics, and track display.
- Further development of UI:
  - Created a header for driver selection: horizontal scrolling allows the user to see all drivers paired with their teammates and select which drivers to populate the driving statistics panel. Drivers are highlighted upon user selection.
  - Created the UI element for driver statistics, which initially contained a blank radar chart for driving style analysis which populates with the selected driver data from the database.
- Updated the driver statistic blocks so that when other drivers are selected, statistic blocks appear above one another and allow for scrolling. Upon clicking a driver for a second time, their statistic block is removed from the panel, and the rest of the blocks are realigned.
- Added a track-portion dropdown selection to driver statistics blocks, which will change the output data to be based on specific turns in a track or the entire track.
- Included a visual of the portion of the track that is currently selected in the track visualization panel.
- Created simulated data for ingestion into the webpage, linked simulated data to the webpage, and tested how that data is displayed.
- Added Chart.js graphics to display driver metrics and information.
- Link actual generated JSON file clustering data to frontend, and check if it updates when that clustering information is updated.
- Finalize UI Elements for compatibility on both desktop and mobile devices and improve presentation of visuals. Ensure that driver selection and track-portion selection function as intended, and that data updates accurately based on the selected driver(s).
- Reach Task: Update map visualization to be imbedded google map of the track instead of static image of the track.

Unfinished:

- Reach Task: Update each driver statistics block so that it includes other drivers who were found to be in the same driving style grouping for their multivariate clustering.

General Tasks.

Completed:

- Amazon Web Services EC2 instance acquired to contain the database, clustering algorithms, and frontend website.
- Preliminary Report for deliverables.
- Final group testing/debugging. Self-explanatory, additional time to combine all facets of the project.
- Project Documentation. This includes adding documentation both inside and outside the code through a ReadMe file and finalizing the state of the GitHub repository in which the project will be stored.
- Finalize Project / Presentation. This is time dedicated to completing Capstone II class assignments, including the final presentation.

5.2 Schedule

Database / Clustering Tasks	Assigned Members	Est. Start	Est. End	Act. Start	Act. End
Acquire server & environment	All	12/31	1/12	1/1	1/1
Data ingestion scripts	Eli, Blake	1/12	1/18	1/15	1/15
Install PostgreSQL + TimescaleDB	Nick, Kolby	1/19	1/25	1/19	1/22
Create and configure DB	Nick, Kolby	1/19	1/25	1/19	1/22
Statistical preprocessing	Eli, Blake	1/19	2/1	2/5	2/11
Clustering Sprint 1	Eli, Nick, Blake, Kolby	2/1	2/15	2/12	2/20
Clustering Sprint 2	Eli, Nick, Blake, Kolby	2/16	3/1	2/23	3/1
Clustering Sprint 3	Eli, Nick, Blake, Kolby	3/2	3/15	3/2	3/15
Analysis Stage of driver styles	Eli, Nick, Blake, Kolby	3/16	3/30	3/16	4/2
Final group testing/debugging	All	3/30	4/12	4/14	4/18
Project Documentation	All	4/13	4/19	4/18	4/22
Finalize Project / Presentation	All	4/20	5/3	4/18	4/22

Website Tasks	Assigned Members	Est. Start	Est. End	Act. Start	Act. End
Acquire server & environment	All	12/31	1/12	1/1	1/1
Rough website prototype	Caden, Sam	1/12	1/21	1/14	1/20
Create a rough UI element for the driver selection header	Caden, Sam	1/21	1/23	1/24	1/26
Create a rough UI element for driver statistic blocks	Caden, Sam	1/26	1/26	1/24	1/26
Update driver statistic blocks to display side-by-side with horizontal scrolling when multiple drivers are selected	Caden, Sam	1/26	1/28	1/26	1/26
Add a track-area dropdown to the track visualization panel	Caden, Sam	1/28	1/30	1/30	1/30
Include visual of portion of track that is currently being ingested for data analysis underneath the driver statistics blocks	Caden, Sam	2/02	2/04	1/30	1/30
Create simulated data for ingestion into the webpage	Caden, Sam, Nick, Eli	2/04	2/11	2/5	2/17
Add Python script to generate radar chart from Pandas dataframe of a driver's driving style to the driver statistics block UI element	Caden, Sam	2/11	2/18	2/11	2/17
Link database to website (DELAYED)	All	2/18	2/25	4/07	4/16
Finalize UI Elements (DELAYED)	Caden, Sam	2/25	3/04	4/06	4/17
Final group testing/debugging	All	3/30	4/12	4/14	4/18
Project Documentation	All	4/13	4/19	4/18	4/22
Finalize Project / Presentation	All	4/20	5/3	4/18	4/22

5.3 Deliverables

- Proposal Document: Contains a listing of each major software component
- Database scheme and initial data: A database following the schema given previously, along with data pulled from TracingInsights
- Source code (Python) for...
    - Collecting and normalizing data from TracingInsights

    - Clustering algorithms
    - Visualization of clustering algorithms
- JavaScript Object Notation (JSON) files containing clustering algorithm outputs
- Web site HTML file and link to the website
- Documentation for usage of the visual dashboard
- Final report

6.0 Key Personnel

Samuel Brooks: Brooks is a senior studying Computer Science and German at the University of Arkansas. Brooks has studied abroad in Germany, taking courses on algorithms and large data manipulation. Interning over the summer with Acosta, Brooks has gained experience with SQL and visualizing large datasets through software such as Power BI. Brooks primarily worked on the implementation of the front-end, and the connection of the clustering algorithm results to the front-end.

Nicholas Calhoun: Calhoun is a senior Computer Science major in the Electrical Engineering and Computer Science department at the University of Arkansas. Calhoun has experience working at the University's Information Technology department, where he helps ensure the University's software systems and websites meet federal accessibility standards. Calhoun also has coursework experience in full stack web development, including the creation of a web app for an SQL database. For this project, he assisted with back-end development and data clustering.

Blake Dismore: Dismore is a senior at the University of Arkansas studying Computer Science with a Mathematics minor. He has experience in software engineering, database design, and data analysis. Over the summer, he interned at USAA in San Antonio, where he worked on a project focused on consolidating data and aligning processes across multiple programs. For this project, he was responsible for backend and clustering algorithm development.

Eli Rose: Rose is a senior Computer Science major at the University of Arkansas with a Mathematics minor. He has worked on projects involving data pipelines, information retrieval systems, and database applications that process large datasets. He has gained experience with Python, SQL, and data analysis libraries, with a focus on data preprocessing and storage. For this project, he was responsible for the data science component, serving as the principal developer and designer of the project's database and its data preprocessing frameworks.

Kolby Stoll: Stoll is a senior Computer Science major at the University of Arkansas. He has experience in various backend systems such as SQL, game development, and integrated systems programming. He previously interned in the Quality Assurance department at Dassault Falcon Jet doing various projects such as building accessible websites for repair center workers, writing documentation in accordance with FAA, EASA, and company regulations, and building out a system for root cause analysis in production environments. For this project, he focused on assisting with back-end development.

Caden Williamson: Williamson is a senior Computer Science and German major at the University of Arkansas. He has work experience in front-end development with companies like

the University of Arkansas and JB Hunt, as well as educational experience in computer graphics, database management, music technology and mobile application development. For this project, he was responsible for assisting with front-end development.

Vanessa Black: As Data Science Manager for GM Motorsports, Vanessa Black leads advanced analytics initiatives to enhance F1 readiness. Her work spans image annotation for computer vision, audio transcription for race communications, and development of a strategy engine to optimize race decisions. She also drives performance engineering optimization through machine learning and predictive modeling to deliver competitive insights. She served as the main mentor and communicator between GM Motorsports and the Capstone team.

Patrick Bounds: Patrick Bounds is an AI/ML Scientist at GM Motorsports, responsible for developing and deploying advanced machine learning models to enhance the engineering of vehicle performance. The insights generated from these tools empower engineers to deeply understand their data, enabling them to design higher-performing cars while minimizing resource usage. He served as the resident expert on clustering for this project due to his vast experience in the subject.

Ryan Samo: Ryan Samo serves as Staff Data Architect for GM Motorsports, leading data platform architecture and infrastructure initiatives across five racing programs (NASCAR, IndyCar, IMSA, WEC, and Formula 1). His work on this project focused on consultation of streaming telemetry systems / real-time data analytics / cloud infrastructure, bringing expertise in distributed systems, event-driven architectures, and high-throughput data processing to support the team's technical objectives.

7.0 Facilities and Equipment

7.1 Webserver

7.1.1 AWS (Amazon Web Services)

- AWS served as the hosting platform for our web application, providing the infrastructure necessary to deploy and maintain the Formula 1 Telemetry Insights Visual Dashboard. AWS offers scalable cloud computing resources that can accommodate varying levels of user traffic without compromising performance. The platform's reliability and uptime guarantees ensure that the dashboard remains accessible to both Cadillac engineers and F1 enthusiasts. Additionally, AWS provides integration capabilities with our PostgreSQL database and frontend application, streamlining the deployment process. The use of an EC2 instance allows for straightforward application deployment, while AWS's pay-as-you-go pricing model ensures cost-effectiveness for the scope of our project. AWS also offers robust security features to protect our telemetry data and clustering results during transmission and storage.

7.1.2 Nginx

- Nginx [12] was used as the web server on the EC2 instance, acting as the layer between incoming browser requests and the files stored on the server. Beyond basic file serving, nginx is responsible for enabling HTTPS on the EC2 instance. Using Let's Encrypt via Certbot, an SSL certificate can be obtained, and nginx will then handle all encryption and decryption of traffic between the server and the user's browser. Nginx requires no application-level backend to be written, being sufficient to handle all server-side responsibilities alone.

7.1.2 JavaScript

- JavaScript is a versatile, high-level programming language that runs directly in web browsers, enabling dynamic and interactive web experiences. Its functionality allows users to interact with webpage elements in real time without requiring a full page reload, making it ideal for creating responsive dashboards. JavaScript provides the tools necessary to handle user interactions and update content such as driver statistic blocks, radar charts, and track visualizations dynamically. Additionally, JavaScript integrates seamlessly with external libraries such as Chart.js [13] for interactive charting, and Leaflet.js [14] for rendering interactive maps directly in the browser.

7.2 Database

7.2.1 PostgreSQL

- PostgreSQL provides SQL with relational database support and is an industry standard. TimescaleDB is an extension of PostgreSQL thatwas considered but ultimately remained unused. The database is structured according to the schema outlined in Section 4.5, with each race having separate tables for race lap data, and telemetry data. This relational structure facilitated rapid queries for populating driver statistics blocks and generating radar chart visualizations. PostgreSQL's compatibility with AWS Relational Database Service (RDS) enabled seamless integration between our database and web hosting infrastructure.

7.3 Data Source

7.3.1 TracingInsights

- TracingInsights is an open-source repository providing Python scripts for gleaning and normalizing F1 data from the FastF1 Python library [3]. This tool automatically corrects fuel usage over time, ensuring that acceleration metrics are more accurate and independent of the weight of the car. TracingInsights served as the primary source for raw telemetry data, which was normalized and processed for input into our clustering algorithms.

8.0 References

[1] “Data Driven at 200 MPH: How Analytics Transforms Formula One Racing,” https://www.forbes.com/sites/joelshapiro/2023/01/26/data-driven-at-200-mph-how-analytics-transforms-formula-one-racing/

[2] “The Ferrari F1 Car Traits That Can Unlock Lewis Hamilton Again,” https://www.the-race.com/formula-1/ferrari-f1-car-traits-lewis-hamilton-driving-style/

[3] “Tracing Insights | Formula 1 Analytics,” https://tracinginsights.com/

[4] “GitHub - hynesconnor/formula1-telemetry-tool: A fast, GUI based application, providing insight into Formula 1 telemetry data,” https://github.com/hynesconnor/formula1-telemetry-tool

[5] “F1NSIGHT,” https://www.f1nsight.com/

[6] “F1 - The Official Home of Formula 1 Racing,” https://www.formula1.com/en/drivers

[7] “PostgreSQL + TimescaleDB: 1,000x Faster Queries, 90% Data Compression and Much More,” https://www.tigerdata.com/blog/postgresql-timescaledb-1000x-faster-queries-90-data-compression-and-much-more

[8] “K means Clustering – Introduction,” https://www.geeksforgeeks.org/machine-learning/k-means-clustering-introduction/

[9] “Hierarchical Clustering in Machine Learning,” https://www.geeksforgeeks.org/machine-learning/hierarchical-clustering/

[10] “Gaussian Mixture Model,” https://www.geeksforgeeks.org/machine-learning/gaussian-mixture-model/

[11] “Fuzzy Clustering ML,” https://www.geeksforgeeks.org/machine-learning/ml-fuzzy-clustering/

[12] “nginx Documentation,” https://nginx.org/en/docs/

[13] “Chart.js Documentation,” https://www.chartjs.org/docs/latest/

[14] “Leaflet.js Documentation,” https://leafletjs.com/reference.html

[15] “Porkbun Domain Registrar,” https://porkbun.com/