F1 project Documentation

Team: Acess

Members: **Denis Humeniuk, Sviatoslav Stehnii, Taras Lysun, Nazar Parnosov**

Motivation

Formula 1 is not only one of the most technologically advanced sports in the world — it is also an extraordinary real-time laboratory for data analysis. Every race weekend generates millions of data points: lap times, tyre degradation curves, pit stop metrics, weather conditions, driver inputs, power unit performance, and strategic decisions. With over 70 years of history, the sport offers a uniquely rich dataset that allows analysts to study human performance, engineering evolution, competitive strategies, and long-term team development.

Understanding what influences success in Formula 1 is far from straightforward. A driver's result is shaped by a complex interaction of mechanical reliability, race strategy, team resources, qualifying performance, traffic patterns, and pure driver skill. At the same time, narratives of dominance — Ferrari in the 2000s, Mercedes in the 2010s, Red Bull in the 2020s — raise deeper questions about how performance evolves across decades.

For students of data engineering and analytics, Formula 1 provides a perfect case study:

- The dataset is large, consistent, and historically rich.
- The problem naturally requires a **medallion architecture** (bronze → silver → gold) for cleaning, feature engineering, and analytics.
- The questions allow for **statistical analysis**, **visualization**, and **interactive dashboards**.
- The sport's competitive nature makes the results intuitive, explainable, and meaningful.

This project uses modern data engineering practices to answer three fundamental analytical questions about driver performance, overperformance, and long-term dominance in Formula 1.

Problem Statement

Problem 1

What factors most strongly influence a driver's race performance in Formula 1?

Problem 2

What determines whether a driver will outperform their starting grid position?

Problem 3

Which drivers and teams show consistent statistical dominance across decades?

Data Processing

1. Data Layers and How Data Changes (Bronze → Silver → Gold)

The pipeline follows a **Bronze → Silver → Gold** pattern.

Below we describe **what changes** at each step for the main entities.

1.1 Bronze Layer – Raw Structured Data

Goal:

Store the raw CSVs in Delta format with **schema enforcement** and **minimal transformation**.

Tables:

- `f1_bronze_races`
- `f1_bronze_drivers`
- `f1_bronze_constructors`
- `f1_bronze_results`
- `f1_bronze_sprint_results`
- `f1_bronze_lap_times`
- `f1_bronze_pit_stops`
- `f1_bronze_qualifying`
- `f1_bronze_driver_standings`
- `f1_bronze_constructor_standings`
- `f1_bronze_circuits`

- `f1_bronze_seasons`
- `f1_bronze_status`

Quarantine tables for bad records:

- `f1_invalid_results`
- `f1_invalid_sprint_results`
- `f1_invalid_lap_times`

Updates vs raw CSV:

- Enforced schemas (data types explicitly specified).
- Basic validation:
  - **Results & sprint_results**:
    - require non-null `raceId`, `driverId`, `constructorId`
    - only keep rows where `year` ∈ [1950, 2030] (via join with `races`)
    - invalid rows → moved to `f1_invalid_results` / `f1_invalid_sprint_results`
  - **Lap times**:
    - require non-null `raceId`, `driverId`, `lap`
    - allow `milliseconds` to be null or ≥ 0
    - invalid rows → moved to `f1_invalid_lap_times`
- Partitioning:
  - `f1_bronze_races` is partitioned by `year`.
  - Other bronze tables are typically **unpartitioned** to keep Bronze simple.

The idea: **Bronze is close to raw**, but with safe schemas and obvious garbage removed.

1.2 Silver Layer – Cleaned, Joined, Analytics-Ready

Goal:

Create **clean, normalized tables** that are ready for analytics and modeling.

Core Silver tables:

- `f1_silver_race_results`
- `f1_silver_sprint_results`
- `f1_silver_lap_times`

- f1_silver_pit_stops
- f1_silver_qualifying
- f1_silver_driver_standings
- f1_silver_constructor_standings
- f1_silver_seasons

1.2.1 Race Results (f1_silver_race_results)

Input:

- f1_bronze_results
- f1_bronze_races
- f1_bronze_drivers
- f1_bronze_constructors
- f1_bronze_circuits
- f1_bronze_status

Transformations / Updates:

- Deduplicate:
  - races , drivers , constructors , circuits , status → dropDuplicates on their IDs.
- Create readable driverName :
  - driverName = trim(forename + ' ' + surname)
- Create readable constructorName (team):
  - constructorName = trim(name) from constructors.
- Join all dimensions:
  - Join results with races to bring year (renamed to season), round , raceName .
  - Join circuits to add circuitName , location , country .
  - Join drivers to add driverName , driverNationality .
  - Join constructors to add teamName , teamNationality .
  - Join status to convert statusId → human-readable statusDescription .
- New columns:
  - season (from races.year)

    - `finishTimeStr`, `finishTimeMs` (renamed from raw)
- Partitioning:
    - Table is **partitioned by** `season` for faster year-based analysis.

Result: one **normalized row per driver–race** with all context (driver, team, track, status).

1.2.2 Sprint Results (`f1_silver_sprint_results`)

Same pattern as race results:

- Reads `f1_bronze_sprint_results` + dimensions.
- Adds `season`, `raceName`, `circuitName`, `driverName`, `teamName`, `statusDescription`.
- Partitioned by `season`.

1.2.3 Lap Times (`f1_silver_lap_times`)

Input:

- `f1_bronze_lap_times`
- `f1_bronze_races`
- `f1_bronze_drivers`

Transformations / Updates:

- Add `season` by joining with races on `raceId`.
- Add `driverName` by joining with drivers on `driverId`.
- Rename columns for clarity:
    - `position` → `lapPosition`
    - `time` → `lapTimeStr`
    - `milliseconds` → `lapTimeMs`
- Partitioning:
    - Partitioned by `season`, `raceId` (lap data can be large → good partitioning key).

1.2.4 Pit Stops (`f1_silver_pit_stops`)

Input:

- `f1_bronze_pit_stops`
- `f1_bronze_races`
- `f1_bronze_drivers`

Transformations / Updates:

- Add `season` via join with races.
- Add `driverName` via join with drivers.
- Rename:
  - `time` → `pitTimeStr`
  - `milliseconds` → `pitTimeMs`
- Partitioned by `season`, `raceId`.

1.2.5 Qualifying (`f1_silver_qualifying`)

Input:

- `f1_bronze_qualifying`
- `f1_bronze_races`
- `f1_bronze_drivers`
- `f1_bronze_constructors`

Transformations / Updates:

- Add `season`, `raceName`.
- Add `driverName`, `teamName`.
- Keep `q1`, `q2`, `q3` times.
- Partitioned by `season`.

1.2.6 Driver / Constructor Standings (`f1_silver_driver_standings` , `f1_silver_constructor_standings`)

Input:

- corresponding bronze tables + races + drivers/constructors.

Transformations / Updates:

- Add `season`, `raceName`, `round`.
- Add `driverName` or `teamName`.
- Partition by `season`.

1.2.7 Seasons (`f1_silver_seasons`)

Just cleaned dim from `f1_bronze_seasons` (deduped).

1.3 Gold Layer – Feature-Engineered Tables

Goal:

Provide **high-level aggregates and ML-ready features**.

Gold tables:

- `f1_gold_driver_season_stats`
- `f1_gold_constructor_season_stats`
- `f1_gold_race_driver_features`

1.3.1 Driver Season Stats (`f1_gold_driver_season_stats`)

Input: `f1_silver_race_results` , `f1_silver_driver_standings`

Transformations / Features:

From `race_results_silver` :

- Per (season, driverId, driverName, teamName):
  - `races_count` – total starts
  - `finished_races` – count where `statusDescription == 'Finished'`
  - `total_points`
  - `avg_grid_position` – average grid position (optionally excluding grid=0)
  - `avg_finish_position` – average finishing position (null DNFs ignored)
  - `wins` – count of `position == 1`
  - `podiums` – count of `position <= 3`
  - `dnf_count` – count where `statusDescription != 'Finished'`
  - `points_per_race = total_points / races_count`
  - `avg_grid_vs_finish = avg_grid_position - avg_finish_position`
  - (negative → tends to gain positions)

From `driver_standings_silver` :

- For the **final race** of a season:
  - `final_champ_points`
  - `final_champ_position`
  - `season_wins_recorded` (from standings table)

- Joined to driver stats based on `(season, driverId)`.

Missing championship fields are filled with 0 to simplify downstream analysis.

1.3.2 Constructor Season Stats (`f1_gold_constructor_season_stats`)

Input: `f1_silver_race_results`, `f1_silver_constructor_standings`

Transformations / Features:

From `race_results_silver`:

- Per (season, constructorId, teamName):
  - `entries_count` – total car-race entries
  - `team_total_points`
  - `wins` – count of `position == 1`
  - `podiums` – count of `position <= 3`
  - `dnf_count` – count of non-finished entries

From `constructor_standings_silver` (final race):

- `final_cons_points`
- `final_cons_position`
- `season_cons_wins_recorded`

Null championship fields are filled with defaults (0 / 0.0).

1.3.3 Race-Driver Features (`f1_gold_race_driver_features`)

Input:

- `f1_silver_race_results`
- `f1_silver_lap_times`
- `f1_silver_pit_stops`
- `f1_silver_qualifying`
- `f1_silver_sprint_results`
- `f1_silver_driver_standings`

Transformations / Features:

From **lap times** (per season, raceId, driverId):

- `lap_count_recorded`

- `best_lap_ms`
- `avg_lap_ms`

From **pit stops**:

- `pit_stop_count`
- `avg_pit_stop_ms`
- `total_pit_stop_ms`

From **qualifying**:

- `quali_best_position`

From **sprint results**:

- `sprint_grid`
- `sprint_finish_position`
- `sprint_points`

From **driver standings** (using a window over (season, driverId) ordered by round):

- `champ_points_after_race`
- `champ_position_after_race`
- `prev_champ_points` – previous race championship points
- `prev_champ_position` – previous race championship position

From **race results**:

- context columns: `season`, `raceId`, `round`, `raceName`, `circuitName`, `country`
- driver/team info: `driverName`, `teamName`, nationalities
- race outcome:
  - `grid`
  - `race_finish_position`
  - `positionText`
  - `positionOrder`
  - `race_points`
  - `race_laps`
  - `statusDescription`

Handling missing values:

To simplify downstream EDA and modeling, numeric nulls are filled with 0, e.g.:

- no pit stops → `pit_stop_count = 0`, `avg_pit_stop_ms = 0`, etc.
- no sprint race → `sprint_points = 0`, etc.
- no standings row (e.g., early seasons) → championship fields = 0.

Partitioning:

- `f1_gold_driver_season_stats` → partitioned by `season`
- `f1_gold_constructor_season_stats` → partitioned by `season`
- `f1_gold_race_driver_features` → partitioned by `season`

---

2. Storage Architecture & Justification

Storage:

All data is stored in **Databricks File System (DBFS)** as **Delta tables** (no S3 needed).

Why Delta + Bronze/Silver/Gold?

- **Delta Lake**
  - ACID transactions → safe concurrent reads/writes.
  - Time travel → easier debugging and reproducibility.
  - Schema enforcement & evolution → protects against bad data.
- **Medallion architecture (Bronze / Silver / Gold)**
  - Bronze: raw/landing, keeps you close to source data.
  - Silver: cleaned, integrated layer → business logic is centralized.
  - Gold: optimized for specific analytics/ML tasks.

Partitioning strategy:

- Many queries filter by **season** (year) → tables partitioned by `season`.
- Lap and pit-stop data can be large and is naturally accessed by race → partitioned by `season`, `raceId`.
- This reduces the amount of data scanned and improves performance for typical analytics workloads.

Data Analyzing

This section presents the analytical results built on top of the **Gold Layer** of our F1 Medallion Architecture. Using the engineered feature tables (driver season stats, constructor season stats, and race-driver features), we performed exploratory data analysis, statistical investigations, and advanced analytics to answer the three main research questions of the project.

3.1 Overview of analytical approach

Our analysis pipeline, implemented in Databricks notebooks, includes:

Exploratory data analysis (EDA)

- Distribution plots for grid position, finish position, points, and lap times
- Correlation heatmaps for driver performance features
- Trend charts (points per season, wins per decade, DNF trends)
- Comparative boxplots for teams and drivers

Advanced analytics

- Overperformance analysis (grid vs finish delta)
- Team/driver dominance scoring per decade
- Reliability analysis (DNFs across decades)

Dashboarding

We published three interactive dashboards in Databricks:

1. **Driver Performance Dashboard** (Problem 1)
2. **Race Overperformance Dashboard** (Problem 2)
3. **Team & Driver Dominance Dashboard** (Problem 3)

These dashboards were built using **Delta Live Views**, **SQL widgets**, and **Databricks Dashboard Editor**, based on curated SQL views such as:

- `f1_driver_performance_eda`
- `f1_overperformance_events`
- `f1_decade_team_driver_dominance`

3.2 Problem 1: What factors influence a driver's race performance?

To understand the strongest predictors of race performance, we analyzed the **f1_gold_race_driver_features** table, computing correlations, distributions, and regression trends.

Key Findings

1. Grid position is the strongest predictor of finish position

Correlation between **grid position** and **race finish position** was consistently high:

- Correlation ~ **0.72** → strong, positive
- Meaning: drivers starting near the front tend to finish near the front.

This aligns with race dynamics in F1, where overtaking difficulty, dirty air, and track layout amplify the importance of qualifying.

2. Championship momentum matters

Features derived from standings show strong relationships:

- **prev_champ_points → race_points** (positive correlation)
- **champ_position_after_race → race_finish_position** (moderate correlation)

Drivers already performing well in the season tend to continue performing strongly—indicating **form consistency**.

3. Pit stop metrics influence results

Significant negative correlations:

- **avg_pit_stop_ms** vs **race_points**
- **pit_stop_count** vs **finish position**

A single slow pit stop can drop a driver multiple places—confirmed by lap-time delta analysis.

4. Lap Time Consistency Predicts Points

- Lower **avg_lap_ms** correlates negatively with finish position.
- **best_lap_ms** has weaker correlation, meaning peak performance matters less than consistency.

5. DNFs Remain a Large Disruption Factor

Drivers with:

- High **DNF counts**
- Many races with `statusDescription != 'Finished'`

show much lower average seasonal points.

Visualizations used

- Heatmap of correlations (grid, lap times, pit stops, sprint data, champ data)
- Boxplots comparing pit stop speed distributions
- Scatter plots:
  - grid vs finish
  - avg_lap_ms vs race_points
  - total_pit_stop_ms vs race_points

3.3 Problem 2: What determines whether a driver outperforms their grid position?

We define an overperformance event as:

> positions_gained = grid − race_finish_position
>
> (positive → driver gained positions)

Key findings

1. Midfield drivers gain the most positions

Grid positions between **8 and 14** showed the highest average positions gained.

Reasons:

- They often benefit from front-runner battles.
- They can avoid congested backmarker traffic.
- Strategy variations are more impactful.

2. Championship leaders rarely overperform

Drivers starting in top 3 positions rarely gain places, because:

- They already start near optimal
- Risk minimization outweighs aggressive overtaking
- Race pace is often optimized to keep position rather than attack

3. High pit stop count → negative overperformance

Drivers with ≥2 pit stops usually **lose** positions unless safety cars intervene.

4. Sprint races slightly improve race overperformance

Correlation:

- **sprint_points** vs **positions_gained** → small positive

  Sprints help establish race pace and reduce surprises on Sunday.

Visualizations used

- Histogram of positions gained
- Overperformance by driver
- Overperformance by team
- Scatter: pit_stop_count vs positions gained
- Maps: overperformance by track/country
- Trend: average positions gained per season

3.4 Problem 3: Which drivers and teams dominate across decades?

Using aggregated Gold tables:

- `f1_gold_driver_season_stats`
- `f1_gold_constructor_season_stats`

We built decade-level dominance scores based on:

- wins
- podiums
- points_per_race
- avg_finish_position
- championship positions

Team dominance by decade

1950s → Alfa Romeo, Maserati

Dominated early racing due to engineering advantages.

1960s → Lotus, Brabham

Lightweight chassis revolution.

1970s → Ferrari, McLaren

Era of Niki Lauda and the emergence of aerodynamic cars.

1980s → McLaren, Williams

Turbo era, Senna + Prost era.

1990s → Williams, Ferrari (rise)

Williams technology dominance; Ferrari catching up.

2000s → Ferrari

Michael Schumacher era of absolute dominance.

2010s → Mercedes, Red Bull

Hybrid era and aerodynamic optimization.

2020s → Red Bull

Verstappen era begins.

Driver dominance by decade

Top drivers (wins, points, average finish):

- **1960s: Clark, Stewart**
- **1970s: Lauda, Fittipaldi**
- **1980s: Senna, Prost**
- **1990s: Schumacher, Häkkinen**
- **2000s: Schumacher, Alonso**
- **2010s: Hamilton, Vettel**
- **2020s: Verstappen**

Visualizations used

- Decade-level heatmaps
- Points per season line charts
- Wins per decade bar charts
- Podium scatter plots
- Team reliability trends (DNFs per decade)

3.5 Summary of insights

Most important factors for race performance

- Grid position
- Lap consistency
- Pit stop execution
- Driver form (championship momentum)
- Avoiding DNFs
- Car/team performance baseline

Key predictors of overperformance

- Starting in midfield positions
- Tracks that allow overtaking
- Clean pit strategy
- Race pace significantly stronger than qualifying pace

Long-term dominance

Teams and drivers show clear decade-based clusters of dominance, aligning with:

- Regulation changes
- Technology breakthroughs
- Engine eras (turbo, hybrid)
- Driver talent generations

3.6 Analysis conclusion

Through structured EDA, statistical modeling, and interactive dashboarding, we answered all main project questions using a robust medallion architecture. The project demonstrates how clean data engineering combined with analytical workflows in Databricks can reveal deep insights into Formula 1 performance patterns.

Data product guide

4.1 Predictive Modelling of Race Outcomes

As data product for our project, we implemented a supervised machine learning model to **predict race outcomes for each driver** based on the engineered features in the **Gold layer**. This model directly supports **Problem 1** (factors influencing race performance) and **Problem 2** (conditions under which drivers overperform their grid position).

4.1.1 Problem framing and target

We model race outcome at the **driver–race** level using the all gold layer tables.

Instead of predicting the raw finishing position directly, we predict the **position delta** relative to the starting grid:

$$target\_delta = race\_finish\_position - grid$$

- `target_delta < 0` → driver gained positions
- `target_delta = 0` → finished where they started
- `target_delta > 0` → driver lost positions

The model predicts `target_deltâ`. The **predicted finishing position** is then reconstructed as:

$$finish\_position = grid + targt\_delta$$

This formulation makes the task more stable and directly aligned with Problem 2.

4.1.2 Features used

The model is trained on a combination of **race-level**, **driver-season**, and **constructor-season** features from Gold tables:

Race-level features (`f1_gold_race_driver_features`)

- Race context: `season`, `round`, `driverId`
- Grid and race outcome:
  - `grid`, `positionOrder`, `race_laps`, `prev_champ_points`, `prev_champ_position`,
- Lap time metrics:
  - `lap_count_recorded`, `best_lap_ms`, `avg_lap_ms`
- Pit stop metrics:
  - `pit_stop_count`, `avg_pit_stop_ms`, `total_pit_stop_ms`
- Qualifying and sprint:

- `quali_best_position`
- `sprint_grid`, `sprint_finish_position`

Driver-season features (`f1_gold_driver_season_stats`)

Joined by `(season, driverId)`:

- `points_per_race`
- `avg_grid_position`, `avg_finish_position`, `avg_grid_vs_finish`
- `wins`, `podiums`, `dnf_count`

Constructor-season features (`f1_gold_constructor_season_stats`)

Joined by `(season, teamName)` or constructor identifier:

- `team_total_points`
- `team_wins`, `team_wins`, `team_wins`

Numeric missing values (e.g., no sprint race, no pit stops, early seasons without standings) are filled with zeros;

4.1.3 Modelling approach and pipeline

We use a **gradient-boosted decision tree regressor** (Spark ML `GBTRegressor`) to predict `target_delta`. The model is wrapped in a Spark ML pipeline and trained in Databricks.

The pipeline consists of:

1. **Categorical encoding**
   - `StringIndexer` for categorical columns:
     - `circuitName`, `country`, `teamName`, `driverName`, `driverNationality`, `teamNationality`
   - `OneHotEncoder` to convert indexed categories into sparse binary vectors
2. **Feature assembly**
   - `VectorAssembler` concatenates:
     - all numeric feature columns (grid, lap times, pit stats, season stats, etc.)
     - all one-hot-encoded categorical vectors
   - Output column: `features`
3. **Regressor**
   - `GBTRegressor` with:
     - `labelCol = "target_delta"`

    - `featuresCol = "features"`
    - tuned depth and number of trees (e.g., moderate depth and ~100 boosting iterations to balance performance and training time)

To avoid information leakage, the model is evaluated on **held-out races**, not reused during training. In a realistic setting, this is implemented as **a season-based split** (train on earlier seasons, test on later seasons), which better reflects real-world generalization to future championships.

4.1.4 Evaluation and baselines

We evaluate the predictive model on a hold-out test set at the **driver–race** level using:

- **Root Mean Squared Error (RMSE)** on predicted finishing position
- **Mean Absolute Error (MAE)** on predicted finishing position
- Share of predictions within ±1 or ±2 positions of the true finish

As a simple baseline, we use:

- **Grid-only baseline**: predicted finish position = starting grid (i.e., `target_deltâ = 0` for all drivers)

The gradient-boosted model systematically outperforms this baseline, especially for **midfield and backmarker drivers**, where strategy, pace, and reliability play a larger role than pure starting position. The model captures patterns such as:

- drivers who consistently gain places relative to their grid
- teams whose race pace is significantly stronger/weaker than qualifying pace
- races and circuits with historically higher position volatility

4.2 Intelligent Agent & User Interface

On top of the Silver/Gold tables and ML model we built an **LLM-powered F1 Agent** – a chat interface that lets users explore the data in natural language instead of writing SQL.

The agent can:

- translate questions into SQL over Silver/Gold Delta tables,
- call the f1_race_position_model to predict finishing positions in specific races,
- return tables, summaries and simple charts using the Databricks visualization API.

In the UI this appears as:

- **Agent Chat** – free-form questions (“Show Verstappen’s win rate by track”, “Predict Leclerc’s finish in Monza 2023”);
- **Data Explorer & Prediction Playground** – guided views to browse drivers/teams and test race scenarios.

From an agent’s point of view, this library is a **single entry point** to:

- read any curated F1 data it needs;
- run analytics and predictions on top of that data;
- return structured results (tables, JSON, aggregates) back to the end user, regardless of which LLM or UI is used (Claude, ChatGPT, custom front-end, etc.).

References and data sources

1. https://www.kaggle.com/datasets/rohanrao/formula-1-world-championship-1950-2020
2. https://www.fia.com/regulation/category/110
3. https://www.formula1.com/en/results/2025/races
4. https://docs.databricks.com/aws/en
5. https://spark.apache.org/docs/latest/api/sql/
6. https://docs.databricks.com/aws/en/sql/user/dashboards