POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Formula One Results Analysis with Documental Database approaches - SMBUD Project

Author(s): **Paolo Riva - 10938975**

Group Number: **15**

Academic Year: 2024-2025

Contents

1 Introduction

Formula One is widely regarded as the pinnacle of motorsport, as the major open-wheel competition available to compete in for all top car manufacturer, motorsport entities and drivers. Formula One nowadays involves more than one-thousand people per team, focused on building and updating the fastest track vehicles in the world in order to race at the highest of levels among the competition. Everything, from car design and race strategy, to brand reputation and economic return, comes down to the two single race results each driver of a considered team score in each race of the current season, so analyzing on-track performance and overall results allows teams to better identify and choose their drivers as the elite ambassadors of the team's heritage in the Formula One environment. With a rich history spanning decades, F1 has accumulated vast datasets documenting race results, driver statistics, and track performances that provide a foundation for analyzing trends, uncovering insights, and developing a deeper understanding of the sport.

In this chapter, the state-of-the-art and the hypotheses are presented, in order to provide an overview of the underlying context the project addresses with respect to Formula One history data, considering its recording and managing, and the technologies best suiting the task of data gathering and analysis among

1.1. State of the Art

The application of data analytics in sports has evolved rapidly in the past decade, and Formula One is no exception. Public databases of race results have opened the doors to explore patterns, assess driver performances, and evaluate track-specific trends.

This project wants to analyze results and available data in order to provide an overview of F1 driver performance in history, i.e. answering questions like:

- What drivers can be considered the greatest to ever take part in F1?
- Which races were the most challenging among history?
- What drivers have shown the highest consistency in their career?

Much of these questions' answers depend on the availability of clean, structured datasets, requiring assumptions about how results are stored, managed, and processed. Therefore, choosing a Documental Database technology to perform queries on data allows to gather several analytic perspectives by performing tailored queries on available driver results, due to flexibility in handling hierarchical and semi-structured data, as well as powerful aggregation operations.

1.2. Project Specification

This project aims at building and querying on a documental database handling sportive results. Having chosen Formula One as the sport of interest, the goal is to design and implement queries capable of analyzing available data on races, drivers, and track performances. MongoDB has been chosen as the database management system, allowing the import of CSV-formatted data as Documental data and allowing various aggregation and filtering operations to be performed in order to gather and analyze the data of interest.

The main objectives of the project include:

- Importing a properly structured database containing data about races, drivers, circuits, and results.
- Designing efficient queries to extract meaningful insights, such as driver performance metrics, track-specific trends, and comparative analysis.

2 | Dataset Structure

To analyze Formula One data effectively, we rely on a main database "results" and 3 secondary databases, all available publicly on Kaggle. While the first one contains all results in races achieved by any driver ever, the others come as a support to associate IDs of tracks and drivers with their details.

2.1. Results Database (`results`)

The `results` database is the backbone of our analysis, recording the detailed outcomes of every race. Each row represents a single driver's performance in a specific race. The key fields include:

- **resultId**: A unique identifier for each result entry.
- **raceId**, **driverId**, **constructorId**: Link to their respective races, driver corresponding to the result, and constructor corresponding to the result.
- **grid** and **position**: Driver's starting grid position and final finishing place.
- **points**: Points awarded to the driver based on their finishing position.
- **fastestLap**, **fastestLapTime, fastestLapSpeed**: Metrics for the driver's fastest lap during the race.
- **statusId**: Indicates whether the driver finished the race or faced an issue (e.g., mechanical failure or crash).
- **time**: Total time to complete the race (if not lapped).

The field has been **preprocessed** to allow a more consistent comparison with respect to MongoDB types.

2.2. Drivers Database (`drivers`)

The `drivers` database is about current and past drivers to ever compete in F1, with fields:

- **driverId**: A unique identifier for each driver.
- **forename** and **surname**: Real name of the driver.
- **dob** and **nationality**: Date of birth and National Flag.
- **code**: 3-letter broadcast identifier (e.g. "HAM")
- **number**: Driver's race number.
- **url**: Links to detailed Wiki profiles about each driver.

2.3. Circuits Database (`circuits`)

The `drivers` database is about current and past tracks to ever host a race in F1, with fields:

- **circuitId**: A unique identifier for each circuit.
- **name**: The official name of the track.
- **location** and **country**: Where in the world you'll find the circuit.
- **lat**, **lng**, and **alt**: Geographic coordinates and altitude.
- **url**: Links to more information about each circuit.

2.4. Races Database (`races`)

The `races` database about the details of a single *race* event, specifying each season's specific details about the event hosted at a specific circuit, with fields:

- **raceId**: The unique identifier for each race.
- **year** and **round**: Which season and which round of the season the race occurred in.
- **circuitId**: A link to the circuit where the race was held.
- **name**, **date**, and **time**: The race's official name and schedule.

- **fp1_date**, **fp2_time**, etc.: Dates and times for practice sessions, qualifying, and sprint races (if applicable).

3 | Commands and Queries

This section presents how the MongoDB environment was set up, the preprocessing on existing data and following MongoDB queries on the imported databases.

3.1. Dataset Format

The "Formula 1 World Championship (1950 - 2024)" dataset was retrieved on Kaggle as a publicly available dataset, including results updated up to the last 6 months of races held. Four CSV files were obtained: *results.csv*, *drivers.csv*, *races.csv*, *circuits.csv*, matching the importing description of *Section 2*.

3.2. Preprocessing

Preprocessing was required on the CSV *results.csv* in order to perform some of the queries more efficiently. When a Formula 1 race ends, the timing data usually shows the winner's total race duration (like "1 hour, 30 minutes, and 45.123 seconds"), while all other drivers' times are shown as gaps to the winner (like "+45.123 seconds" or "+1 minute, 30.456 seconds" behind). This preprocessing Python script converts the winner's time into plain seconds, then for each other driver, it takes their gap time, converts it to seconds too, and adds it to the winner's time. This gives us every driver's actual finish time in seconds, making it much easier to compare and analyze race performances between drivers in the same race.

```
1  def transform_time(input_csv, output_csv):
2      def parse_time_to_seconds(time_str):
3          if "+" in time_str:  # Gap time, like "+minutes:seconds.
               milliseconds" or "+seconds.milliseconds"
4              time_str = time_str[1:]  # Remove the '+' sign
5          parts = time_str.split(":")
6          if len(parts) == 2:  # Format: "minutes:seconds.milliseconds"
7              minutes, seconds = map(float, parts)
```

```
            return round(minutes * 60 + seconds, 3)
        elif len(parts) == 1:  # Format: "seconds.milliseconds"
            return round(float(parts[0]), 3)
        else:
            raise ValueError(f"Unexpected time format: {time_str}")

    def winner_time_to_seconds(winner_time):
        parts = winner_time.split(":")
        if len(parts) == 3:  # Format: "hours:minutes:seconds.milliseconds"
            hours, minutes, seconds = map(float, parts)
            return round(hours * 3600 + minutes * 60 + seconds, 3)
        elif len(parts) == 2:  # Format: "minutes:seconds.milliseconds"
            minutes, seconds = map(float, parts)
            return round(minutes * 60 + seconds, 3)
        else:
            raise ValueError(f"Unexpected winner time format: {winner_time}")

    with open(input_csv, mode="r", newline="", encoding="utf-8") as infile, \
         open(output_csv, mode="w", newline="", encoding="utf-8") as outfile:
        reader = csv.DictReader(infile)
        fieldnames = reader.fieldnames
        writer = csv.DictWriter(outfile, fieldnames=fieldnames)
        writer.writeheader()

        for row in reader:
            if row["time"] == "\\N":  # Skip rows where the driver has retired or been lapped
                writer.writerow(row)
                continue

            if row["positionOrder"] == "1":  # For the winner
                winner_time = row["time"]
                winner_seconds = winner_time_to_seconds(winner_time)
                row["time"] = winner_seconds
            else:  # For other drivers
                gap_time = row["time"]
                try:
                    gap_seconds = parse_time_to_seconds(gap_time)
                    row["time"] = round(winner_seconds + gap_seconds, 3)  # Add the gap to the winner's total time
```

```
                except ValueError as e:
                    print(f"Skipping row with invalid time format: {row}")
                    continue

            writer.writerow(row)
```

An outputted *f1_results_transformed.csv* file is used as preprocessed replacement of the original *results.csv* file, allowing more efficient queries overall on the **time** field.

3.3. Interface and Import

MongoDB Compass was chosen as the official GUI tool for MongoDB, from the official MongoDB website. After installation on Windows 11, the local server was configured to run on the default port 27017. The data import process used MongoDB Compass's import functionality, which allows direct CSV-to-document conversion while maintaining data (possibly mixed) types integrity. Each CSV file was mapped to a corresponding collection within the database, with appropriate data type conversions and indexing strategies applied to optimize query performance.

3.4. Setup and Operators

3.4.1. Query syntax

MongoDB Compass allows to perform aggregation queries by simply specifing aggregation operations inside branches ("[]"). The following queries are, for this reason, in such form. To perform such queries in a regular MongoDB Shell, it's enough to enclosure the following queries as:

```
db.results.aggregate([QUERY]);
```

3.4.2. Import Data

To import data and create Documents, it's enough to create a Local MongoDB connection, and import the 4 specified databases using the MongoDB Compass import function from JSON/CSV as:

- results.csv as *results*
- drivers.csv as *drivers*

- circuits.csv as *circuits*
- races.csv as *races*

3.4.3. Specific operators

$lookup Operator

The `$lookup` operator performs a left outer join operation between collections in MongoDB, enabling the combination of documents from multiple collections within the same database. This operator requires four essential parameters:

- `from`: Specifies the target collection to join with
- `localField`: Defines the field from the input documents to match
- `foreignField`: Indicates the field to match against in the target collection
- `as`: Designates the name for the new array field that will contain the matching documents

When executed, `$lookup` adds a new array field to each input document. This array contains the matching documents from the foreign collection. If no matches are found, the array will be empty but still present in the result document. In our case, this allows to associate each result with the specific driver involved in it just from the specified ID, along with race and track details.

$concat Operator

The `$concat` operator is a string manipulation tool that combines multiple strings into a single string value. It accepts an array of string expressions as input and concatenates them in the specified order. This operator proves particularly useful when:

- Combining multiple fields into a single string field
- Adding constant strings between field values
- Formatting output data for presentation or export

In our case, this allows to join both name and surname of the drivers.

$switch Operator

The `$switch` operator provides conditional branching within MongoDB's aggregation pipeline, similar to switch-case statements in traditional programming languages. This

operator evaluates a series of cases and returns a value based on the first matching condition.

The operator requires two main components:

- `branches`: An array of possible cases, where each case contains:
  - `case`: The condition to evaluate
  - `then`: The value to return if the condition is true
- `default`: The value to return if no cases match (required to handle all possible scenarios)

The `$switch` operator evaluates each case in order until it finds a true condition, then returns the corresponding `then` value. In our case, it is used to evaluate the points achieved by the driver based on its finishing position.

$first Operator

The `$first` operator is an accumulator that returns the first document from a grouped result set in MongoDB's aggregation pipeline. It operates in two distinct contexts: group stages and array manipulation.

Within a `$group` stage, `$first` returns the first value it encounters in each group, making it particularly useful to retrieve initial documents in a list.

$cond Operator

The `$cond` operator provides if-then-else logical operations within MongoDB's aggregation pipeline. This operator can be expressed in two syntactical forms: a ternary expression or an object expression.

The operator requires three components:

- `if`: The boolean expression to evaluate
- `then`: The value to return when the condition is true
- `else`: The value to return when the condition is false

3.5. Queries

3.5.1. Query 1: Average Driver position

This query evaluates the average grid positions of all drivers with at least one result, returning the drivers in descending order.

```
[
  {
    // Grouping by driverId to calculate total wins and average grid
       position
    $group: {
      _id: "$driverId",
      avgGridPosition: { $avg: "$grid" }
    }
  },
  {
    $match: {
      avgGridPosition: { $ne: 0 }
    }
  },
  {
    // Join with the drivers_id collection to get driver details
    $lookup: {
      from: "drivers",
      localField: "_id",
      foreignField: "driverId",
      as: "driverDetails"
    }
  },
  {
    // Unwind the driverDetails array to flatten it
    $unwind: "$driverDetails"
  },
  {
    //Project final fields: driver name, total wins, and avg grid
       position
    $project: {
      _id: 0,
      driverName: {
        $concat: [
          "$driverDetails.forename",
          " ",
          "$driverDetails.surname"
        ]
      },
      totalWins: 1,
```

```
      avgGridPosition: {
        $round: ["$avgGridPosition", 2]
      } // Round to 2 decimals
    }
  },

  {
    // Sort by total wins in descending order
    $sort: { avgGridPosition: 1 }
  }
]
```

Returned:


```
[{
  "driverName": "Pierre-Henri Raphanel",
  "avgGridPosition": 1.06
},
{
  "driverName": "Juan Fangio",
  "avgGridPosition": 2.48
},
{
  "driverName": "Ayrton Senna",
  "avgGridPosition": 3.13
},
{
  "driverName": "Jack McGrath",
  "avgGridPosition": 3.17
},
{
  "driverName": "Nino Farina",
  "avgGridPosition": 3.19
},
{
  "driverName": "Neville Lederle",
  "avgGridPosition": 3.33
},
{
  "driverName": "Emilio de Villota",
  "avgGridPosition": 3.5
},
{
  "driverName": "Jim Clark",
  "avgGridPosition": 3.6
```

```
}
...
```

3.5.2. Query 2: Most Races in F1

This query computes the total races any driver has ever took part in, sorting the results in descending order.

```
[
  {
    // Group by driverId and count the total races
    $group: {
      _id: "$driverId", // Group by driverId
      totalRaces: { $sum: 1 } // Count the total number of races
    }
  },
  {
    // Lookup to join with the drivers_id collection
    $lookup: {
      from: "drivers", // The collection with driver details
      localField: "_id", // driverId from the grouped data
      foreignField: "driverId", // driverId field in drivers_id
      as: "driverDetails" // Alias for the joined data
    }
  },
  {
    // Unwind the driverDetails array to normalize
    $unwind: "$driverDetails"
  },
  {
    // Project the desired fields: name and total races
    $project: {
      _id: 0,
      driverName: {
        $concat: [
          "$driverDetails.forename",
          " ",
          "$driverDetails.surname"
        ]
      }, // Combine forename and surname
      totalRaces: 1 // Include the total races
    }
  },
  {
```

```
    // Sort the results by totalRaces in descending order
    $sort: { totalRaces: -1 }
  }
]
```

Returned:

```
[{
  "totalRaces": 392,
  "driverName": "Fernando Alonso"
},
{
  "totalRaces": 352,
  "driverName": "Kimi Raikkonen"
},
{
  "totalRaces": 344,
  "driverName": "Lewis Hamilton"
},
{
  "totalRaces": 326,
  "driverName": "Rubens Barrichello"
},
{
  "totalRaces": 309,
  "driverName": "Jenson Button"
},
{
  "totalRaces": 308,
  "driverName": "Michael Schumacher"
},
{
  "totalRaces": 300,
  "driverName": "Sebastian Vettel"
},
...
```

3.5.3. Query 3: Most F1 Wins

This query computes the total number of wins for each driver, and returns the most winning drivers in history in descending order.

```
[
  {
```

```
    // Filter race results to include only wins (position: 1)
    $match: { position: 1 }
  },
  {
    // Group by driverId and count the number of wins
    $group: {
      _id: "$driverId", // Group by driverId
      totalWins: { $sum: 1 } // Count total wins
    }
  },
  {
    // Lookup to join with the drivers_id collection
    $lookup: {
      from: "drivers", // The collection with driver details
      localField: "_id", // driverId from the grouped data
      foreignField: "driverId", // driverId field in drivers_id
      as: "driverDetails" // Alias for the joined data
    }
  },
  {
    // Unwind the driverDetails array to normalize
    $unwind: "$driverDetails"
  },
  {
    // Step 5: Project the desired fields: name and total wins
    $project: {
      _id: 0, // Hide the default _id
      driverName: {
        $concat: [
          "$driverDetails.forename",
          " ",
          "$driverDetails.surname"
        ]
      }, // Combine forename and surname
      totalWins: 1 // Include the total wins
    }
  },
  {
    // Step 6: Sort the results by totalWins in descending order
    $sort: { totalWins: -1 }
  }
]
```

Returning:

```
[{
  "totalWins": 104,
  "driverName": "Lewis Hamilton"
},
{
  "totalWins": 91,
  "driverName": "Michael Schumacher"
},
{
  "totalWins": 61,
  "driverName": "Max Verstappen"
},
{
  "totalWins": 53,
  "driverName": "Sebastian Vettel"
},
{
  "totalWins": 51,
  "driverName": "Alain Prost"
},
{
  "totalWins": 41,
  "driverName": "Ayrton Senna"
},
{
  "totalWins": 32,
  "driverName": "Fernando Alonso"
},
...
```

3.5.4. Query 4: Most laps ever completed by a driver

This query computes the maximum laps ever completed by a driver in F1 during his career.

```
[
  {
    $group: {
      _id: "$driverId", // Group by driverId
      totalLaps: { $sum: "$laps" }
    }
  },
  {
```

```
    $sort: { totalLaps: -1 } // Sort descending
  },
  {
    $limit: 1 // Show the highest result
  }
]
```

Returning:

```
[{
  "_id": 4,
  "totalLaps": 21173
}]
```

3.5.5. Query 5: Most Points in F1

This query shows the drivers who achieved the most points in F1. Although the *points* field exists, we want to compare all drivers in history equally, since the point system in F1 actually changed between 2009 and 2011 assigning much more points to the top 10 grid.

```
[
  {
    // Add points based on the modern scoring system
    $project: {
      driverId: 1,
      points: {
        $switch: {
          branches: [
            {
              case: { $eq: ["$position", 1] },
              then: 25
            },
            {
              case: { $eq: ["$position", 2] },
              then: 18
            },
            {
              case: { $eq: ["$position", 3] },
              then: 15
            },
            {
              case: { $eq: ["$position", 4] },
              then: 12
```

```
            },
            {
              case: { $eq: ["$position", 5] },
              then: 10
            },
            {
              case: { $eq: ["$position", 6] },
              then: 8
            },
            {
              case: { $eq: ["$position", 7] },
              then: 6
            },
            {
              case: { $eq: ["$position", 8] },
              then: 4
            },
            {
              case: { $eq: ["$position", 9] },
              then: 2
            },
            {
              case: { $eq: ["$position", 10] },
              then: 1
            }
          ],
          default: 0 // Positions outside top 10 get 0 points
        }
      }
    }
  },
  {
    // Group by driverId to calculate total points
    $group: {
      _id: "$driverId",
      totalPoints: { $sum: "$points" }
    }
  },
  {
    // Join with the drivers_id collection to get driver names
    $lookup: {
      from: "drivers",
      localField: "_id",
      foreignField: "driverId",
```

```
      as: "driverDetails"
    }
  },
  {
    // Unwind the driverDetails array
    $unwind: "$driverDetails"
  },
  {
    // Project the driver name and total points
    $project: {
      _id: 0,
      driverName: {
        $concat: [
          "$driverDetails.forename",
          " ",
          "$driverDetails.surname"
        ]
      },
      totalPoints: 1
    }
  },
  {
    // Sort by total points in descending order
    $sort: { totalPoints: -1 }
  }
]
```

Returning:

```
[{
  "totalPoints": 5048,
  "driverName": "Lewis Hamilton"
},
{
  "totalPoints": 3890,
  "driverName": "Michael Schumacher"
},
{
  "totalPoints": 3287,
  "driverName": "Sebastian Vettel"
},
{
  "totalPoints": 3143,
  "driverName": "Fernando Alonso"
},
```

```
{
  "totalPoints": 2791,
  "driverName": "Kimi Raikkonen"
},
{
  "totalPoints": 2730,
  "driverName": "Max Verstappen"
},
{
  "totalPoints": 2483,
  "driverName": "Alain Prost"
},
...
```

3.5.6. Query 6: Average Time of completion for each race

This query evaluates the average time of completion for each race ever held in history, to highlight the longest races ever completed by drivers in terms of total time.

```
[
  {
    $match: {
      time: { //Match only drivers with full laps, not DNF, not Lapped
        $exists: true,
        $ne: null
      }
    }
  },
  {
    $group: { // Evaluate Average time
      _id: "$raceId",
      averageTime: {
        $avg: "$time"
      }
    }
  },
  {
    $sort: {
      averageTime: 1
    }
  },
  {
    $project: {
      _id: 0,
```

```
      raceId: "$_id",
      averageTime: 1
    }
  }
]
```

Returning:

```
[{
  "averageTime": 226.32395000000002,
  "raceId": 1063
},
{
  "averageTime": 1527.7229999999997,
  "raceId": 320
},
{
  "averageTime": 2574.25,
  "raceId": 579
},
{
  "averageTime": 3534.8642857142854,
  "raceId": 587
},
{
  "averageTime": 3690.9816,
  "raceId": 441
},
{
  "averageTime": 4066.579090909091,
  "raceId": 540
},
{
  "averageTime": 4124.138,
  "raceId": 154
},
...
```

3.5.7. Query 7: Most Wins per Circuit

This query joins both circuit and race details to the result documents in order to compute the drivers with the highest number of wins on each track, showing for each circuit the driver that achieved the highest number of wins in such track.

```
[
  {
    // Filter for race winners
    $match: { position: 1 }
  },
  {
    // Lookup the race details to get the circuitId
    $lookup: {
      from: "races",
      localField: "raceId",
      foreignField: "raceId",
      as: "raceDetails"
    }
  },
  {
    //Unwind the raceDetails array to access circuitId
    $unwind: "$raceDetails"
  },
  {
    // Lookup the circuit details to get circuit data
    $lookup: {
      from: "circuits",
      localField: "raceDetails.circuitId",
      foreignField: "circuitId",
      as: "circuitDetails"
    }
  },
  {
    // Unwind the circuitDetails array
    $unwind: "$circuitDetails"
  },
  {
    // Group by circuit and driver, and count wins
    $group: {
      _id: {
        circuitId: "$circuitDetails.circuitId",
        driverId: "$driverId"
      },
      circuitName: {
        $first: "$circuitDetails.name"
      }, // Get circuit name
      driverId: { $first: "$driverId" }, // Get driverId
      wins: { $sum: 1 } // Count wins
    }
```

```
  },
  {
    // Group by circuit to find the driver with the most wins
    $group: {
      _id: "$_id.circuitId", // Group by circuitId
      circuitName: { $first: "$circuitName" }, // Include circuit name
      driverId: { $first: "$driverId" }, // Driver with the most wins
      wins: { $max: "$wins" } // Maximum wins
    }
  },
  {
    // Lookup driver details to get the driver's name
    $lookup: {
      from: "drivers",
      localField: "driverId",
      foreignField: "driverId",
      as: "driverDetails"
    }
  },
  {
    // Unwind the driverDetails array
    $unwind: "$driverDetails"
  },
  {
    // Project the final output
    $project: {
      _id: 0,
      circuitName: 1,
      driverName: {
        $concat: [
          "$driverDetails.forename",
          " ",
          "$driverDetails.surname"
        ]
      },
      wins: 1
    }
  },
  {
    // Sort by circuit name (alphabetic order)
    $sort: { circuitName: 1 }
  }
]
```

Returning:


```
[{
  "circuitName": "Aintree",
  "wins": 2,
  "driverName": "Stirling Moss"
},
{
  "circuitName": "Albert Park Grand Prix Circuit",
  "wins": 4,
  "driverName": "David Coulthard"
},
{
  "circuitName": "Autodromo Enzo e Dino Ferrari",
  "wins": 7,
  "driverName": "Damon Hill"
},
{
  "circuitName": "Autodromo Internazionale del Mugello",
  "wins": 1,
  "driverName": "Lewis Hamilton"
},
{
  "circuitName": "Autodromo Nazionale di Monza",
  "wins": 5,
  "driverName": "Graham Hill"
},
```


3.5.8. Query 8: Highest average speed in a race

This query averages the drivers' fastest lap on each race event to average the highest average speed ever observed in a race. This query aims at showing the event where there raced the fastest generation of cars on a single track in history.

```
[
  {
    // Filter records where fastestLapSpeed exists
    $match: { fastestLapSpeed: { $ne: null } }
  },
  {
    // Group by raceId and calculate the average fastestLapSpeed
    $group: {
      _id: "$raceId",
      avgFastestLapSpeed: {
```

```
        $avg: "$fastestLapSpeed"
      }
    }
  },
  {
    // Sort by the average fastestLapSpeed in descending order
    $sort: { avgFastestLapSpeed: -1 }
  },
  {
    // Limit to the race with the highest average fastestLapSpeed
    $limit: 1
  },
  {
    // Lookup race details for the corresponding raceId
    $lookup: {
      from: "races",
      localField: "_id",
      foreignField: "raceId",
      as: "raceDetails"
    }
  },
  {
    // Unwind the race details
    $unwind: "$raceDetails"
  },
  {
    // Project the final result with race details and average speed
    $project: {
      _id: 0,
      raceName: "$raceDetails.name",
      raceDate: "$raceDetails.date",
      circuitId: "$raceDetails.circuitId",
      avgFastestLapSpeed: 1
    }
  }
]
```

Returning:

```
[{
  "avgFastestLapSpeed": 250.63199999999998,
  "raceName": "Italian Grand Prix",
  "raceDate": {
    "$date": "2004-09-12T00:00:00.000Z"
  },
```

```
  "circuitId": 14
}]
```

3.5.9. Query 9: Most Podiums in F1

This query aims at showing the 10 drivers with the most number of podiums in F1.

```
[
  {
    $match: { position: { $lte: 3 } } // Positions 1, 2, 3
  },
  {
    $group: {
      _id: "$driverId", // Group by driverId
      podiums: { $sum: 1 }
    }
  },

  {
    // Join with the drivers_id collection to get driver details
    $lookup: {
      from: "drivers",
      localField: "_id",
      foreignField: "driverId",
      as: "driverDetails"
    }
  },
  {
    // Unwind the driverDetails array to flatten it
    $unwind: "$driverDetails"
  },
  {
    // Project final fields: driver name, total wins, and avg grid
       position
    $project: {
      _id: 0,
      driverName: {
        $concat: [
          "$driverDetails.forename",
          " ",
          "$driverDetails.surname"
        ]
      },
      podiums: 1
```

```
    }
  },

  {
    $sort: { podiums: -1 }
  },
  { $limit: 10 }
]
```

Returning:

```
[{
  "podiums": 199,
  "driverName": "Lewis Hamilton"
},
{
  "podiums": 155,
  "driverName": "Michael Schumacher"
},
{
  "podiums": 122,
  "driverName": "Sebastian Vettel"
},
{
  "podiums": 107,
  "driverName": "Max Verstappen"
},
{
  "podiums": 106,
  "driverName": "Alain Prost"
},
{
  "podiums": 106,
  "driverName": "Fernando Alonso"
},
{
  "podiums": 103,
  "driverName": "Kimi Raikkonen"
},
{
  "podiums": 80,
  "driverName": "Ayrton Senna"
},
{
  "podiums": 68,
```

```
  "driverName": "Rubens Barrichello"
},
{
  "podiums": 67,
  "driverName": "Valtteri Bottas"
}]
```

3.5.10. Query 10: Races finished disastrously

This query returns the number of races with the highest number of crashes, technical DNFs and mistakes in F1 history, specifically counting all the races where the number of drivers that finished the race (statusId == 1) was lower than the ones that didn't.

```
[
  {
    // Group results by raceId and calculate counts for statusId == 1
       and statusId > 1
    $group: {
      _id: "$raceId",
      totalFinishedRegularly: {
        $sum: {
          $cond: [{ $eq: ["$statusId", 1] }, 1, 0]
        }
      },
      totalNotFinishedRegularly: {
        $sum: {
          $cond: [{ $gt: ["$statusId", 1] }, 1, 0]
        }
      }
    }
  },
  {
    // Filter only races where not finished > finished regularly
    $match: {
      $expr: {
        $gt: [
          "$totalNotFinishedRegularly",
          "$totalFinishedRegularly"
        ]
      }
    }
  },
  {
    //Count the number of races
```

```
    $count: "racesWithMoreNonFinishers"
  }
]
```

Returning:

```
[{
  "racesWithMoreNonFinishers": 893
}]
```

4 | Conclusion

This project aimed at showing different analytics regarding Formula One over the history of the sport demonstrating the use of MongoDB in analyzing several data entries. By leveraging advanced aggregation techniques, such as grouping, filtering, and lookups, we were able to meaningful insights into drivers' performances, race trends, and circuit statistics. The flexibility of MongoDB allowed for efficient handling of complex relationships between datasets, such as driver details, race results, and circuit information. Although an agglomerated grouping of the four databases involved could have also be considered by utilizing subdocument structures inside Results document, which was not chosen to avoid excessive redundancy, this analysis highlighted the potential of NoSQL databases for exploring and uncovering patterns in large, interconnected datasets, providing valuable perspectives on the evolution of Formula One and its entities.

A | External Links

- Formula 1 World Championship (1950 - 2024) Dataset: https://www.kaggle.com/datasets/rohanrao/formula-1-world-championship-1950-2020
- MongoDB Compass: https://www.mongodb.com/products/tools/compass