

DTSC 701 Introduction to Big Data

Summer 2023

Big Data Project

Analyzing Formula 1 Data Using AWS Services

Prof. Liangwen Wu
Gupta, Abhimanyu
NYIT ID: 1322819
agupta52@nyit.edu

Introduction:

As a passionate Formula 1 fan, I have always been fascinated by the statistics and insights that the sport offers. I decided to analyze some formula 1 stats to find the driver that's the greatest of all time or the GOAT. The motivation behind choosing this dataset was my personal interest in Formula 1 and the desire to leverage cloud-based tools to gain insights into the sport's rich dataset.

The dataset chosen for this analysis was the "Formula 1 Race Data," obtained from Kaggle. This dataset contains a wealth of information about Formula 1 races, drivers, teams, and results. It includes details such as race results, lap times, weather conditions, and more. More specifically, I used the driver.csv and results.csv datasets. results.csv is a log of each race result for each driver and also has details like fastest lap, timing, starting and ending position in the race. So, if there are 20 drivers, for each race there are 20 results recorded. driver.csv is a log of all drivers and driver stats. It identifies drivers by a unique number that is same on all the csv files. This is used to identify drivers name to find the real GOAT. The dataset was obtained from the following source and was stored in an Amazon S3 bucket. source: <https://www.kaggle.com/datasets/rohanrao/formula-1-world-championship-1950-2020?select=drivers.csv>

The goal of this exercise is to use this Formula 1 data and find out the driver that has done the most overtakes. This should give us an idea of who the greatest Formula 1 driver could be.

Methodology:

To achieve this, the following steps were taken. We leveraged Amazon Web Services (AWS) for its powerful cloud-based tools, specifically Amazon EMR for data processing and storage using Amazon S3. Below is the detailed breakdown of our approach:

Creating an EMR Cluster

Amazon EMR was chosen as it provides a powerful platform for processing and analyzing large datasets using popular frameworks such as Apache Spark. I initiated the process by creating an EMR cluster with appropriate configurations, this cluster would serve as the computational backbone for our data analysis.

Storing and accessing the Dataset

To store and access the dataset stored, it was first stored in an S3 bucket. I configured permissions by creating an IAM role. This role was equipped with policies granting read and write access to the S3 bucket. This ensured that the EMR cluster could seamlessly retrieve the dataset.

Data Processing

With the EMR cluster up and running, I submitted a PySpark script that read the Formula 1 race data from the S3 bucket. The calculated statistics were then used to gain insights into the dataset. I utilized PySpark to identify top-performing driver based on positions gained during each race. After performing the necessary calculations, the results were stored in a designated folder within the same S3 bucket.

Python Script

Initially scripted and validated in Python, it was subsequently adapted for PySpark to leverage its distributed capabilities. The script primarily focused on driver-oriented metrics, extracting performance insights. Here's a concise breakdown of the script's flow:

CSV files containing race and driver data were loaded, followed by essential data cleaning and preprocessing steps to ensure data accuracy. Next the driver metrics are calculated. The core analysis encompassed driver performance during a race weekend. Number of positions gained are calculated by subtracting race position from the starting grid position. Data was then grouped by drivers, enabling the derivation of comprehensive aggregated statistics.


Once the python code was running and generating the desired output using numPy, the shift to PySpark was made. To unlock distributed processing, the script transitioned to PySpark, capitalizing on its DataFrame features and SQL querying prowess.

Results

The results offer a profound glimpse into Formula 1 driver performance. In this section we will explore the final results of the exercise.

The python script first generated the following output:

positions_gained...	
driverId	TotalPositionsGained
1	155.0
2	335.0
3	56.0
4	585.0
5	150.0
6	57.0
7	34.0
8	350.0
9	16.0
10	224.0
11	161.0
12	57.0
13	176.0
14	284.0
15	173.0
16	138.0
17	154.0
18	469.0
19	29.0
20	205.0
21	453.0
22	350.0
23	186.0
24	175.0
25	190.0
26	81.0
27	116.0
28	0.0
29	61.0
30	269.0
31	59.0
32	57.0
33	127.0
34	9.0



The screenshot shows a terminal window with a title bar containing three colored circles (red, yellow, green) and a file name 'goat.txt'. The terminal output consists of several lines of ASCII art that form the text 'FERNANDO ALONSO IS THE GOAT!!!'. The ASCII art is composed of various symbols like brackets, slashes, and backslashes, arranged to create the letters of the name and the phrase.

Fig 2. Greatest of all time text output

Fig 1. Positions gained output file

Here we can see that driver named Fernando Alonso has done the most number of overtakes in his career and can be considered the “greatest of all time” or the “GOAT”

Figures 3, 4 and 5 show the conversion to spark code, running on AWS EMR using terminal and S3 Storage.

```
*goatfinder.py - /Users/ab/Downloads/goatfinder.py (3.11.2)*

# Calculate positions gained for each driver
data = data.withColumn("positions_gained", data["grid"] - data["position"])
positions_gained_by_driver = data.groupBy('driverId').sum('positions_gained')

# Save position gain data
positions_gained_by_driver.write.csv(output_csv, header=True)

def goatfinder(spark, positions_csv, drivers_csv):
    # Read Data using Spark
    positions_data = spark.read.option("header", "true").csv(positions_csv)
    drivers_data = spark.read.option("header", "true").csv(drivers_csv)

    # Merge positions gained and drivers names using driverId
    merged_data = positions_data.join(drivers_data, on='driverId')

    # Convert first and last names to uppercase
    merged_data = merged_data.withColumn("forename", merged_data["forename"].upper())
    merged_data = merged_data.withColumn("surname", merged_data["surname"].upper())

    # Save merged data
    merged_data.write.csv('goat.csv', header=True)

    # Find the driver with the most wins
    most_wins_driver = merged_data.orderBy("TotalPositionsGained", ascending=False).first()

    # Generate "goat" file
    with open('goat.txt', 'w') as txt_file:
        txt_file.write(f"    \n [---] \n []=/[\\=[] \n /:|\\ \n | /U\\ \n || _ || {most_wins_driver['forename']} {"

if __name__ == "__main__":
    spark = SparkSession.builder.appName("GoatFinder").getOrCreate()

    positions_gained(spark, 's3://goatf1/results.csv', 's3://goatf1/positions_gained')
    goatfinder(spark, 's3://goatf1/positions_gained/', 's3://goatf1/drivers.csv')

    spark.stop()
```

Fig 3. Converting code to use Spark

```
Downloads — hadoop@ip-172-31-31-0:~ — ssh -i EMRKey.pem hadoop@ec2-3-128-25-83.us-east-2.compute.amazonaws.com — 136x...

EEEEEEEEEEEEEEEEEEEE MMMMMMM          MMMMMMM RRRRRRRRRRRRRR
E:::EEEEEEEEEEEEEEEE M:::M:::M          M:::M:::M R:::R:::R:::R:::R
EE:::EEEEEEEEEEEEEEEE M:::M:::M          M:::M:::M R:::R:::R:::R:::R
E:::E          EEEEE M:::M:::M          M:::M:::M RR:::R          R:::R
E:::E          M:::M:::M M:::M:::M          M:::M:::M R:::R          R:::R
E:::EEEEEEEEEEEE M:::M M:::M M:::M M:::M          R:::RRRRRR:::R
E:::EEEEEEEEEEEE M:::M M:::M M:::M          R:::RRRRRR:::R
E:::E          M:::M M:::M M:::M          R:::R          R:::R
E:::E          EEEEE M:::M          MMM          M:::M R:::R          R:::R
EE:::EEEEEEEEEEEE M:::M          M:::M          R:::R          R:::R
E:::EEEEEEEEEEEE M:::M          M:::M RR:::R          R:::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM          MMMMMMM RRRRRRR          RRRRRR

[hadoop@ip-172-31-31-0 ~]$ vi goatfinder.py
[hadoop@ip-172-31-31-0 ~]$ ls
goatfinder.py
[hadoop@ip-172-31-31-0 ~]$ spark-submit --master yarn --deploy-mode client goatfinder.py

23/08/26 15:56:44 INFO SparkContext: Running Spark version 3.4.0-amzn-0
23/08/26 15:56:44 INFO ResourceUtils: =====
23/08/26 15:56:44 INFO ResourceUtils: No custom resources configured for spark.driver.
23/08/26 15:56:44 INFO ResourceUtils: =====
23/08/26 15:56:44 INFO SparkContext: Submitted application: GoatFinder
23/08/26 15:56:44 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 4, script
: , vendor: , memory -> name: memory, amount: 9486, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor: ), task r
esources: Map(cpus -> name: cpus, amount: 1.0)
23/08/26 15:56:44 INFO ResourceProfile: Limiting resource is cpus at 4 tasks per executor
23/08/26 15:56:44 INFO ResourceProfileManager: Added ResourceProfile id: 0
23/08/26 15:56:44 INFO SecurityManager: Changing view acls to: hadoop
23/08/26 15:56:44 INFO SecurityManager: Changing modify acls to: hadoop
23/08/26 15:56:44 INFO SecurityManager: Changing view acls groups to:
23/08/26 15:56:44 INFO SecurityManager: Changing modify acls groups to:
23/08/26 15:56:44 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: hadoop;
```

Fig 4. Running AWS EMR

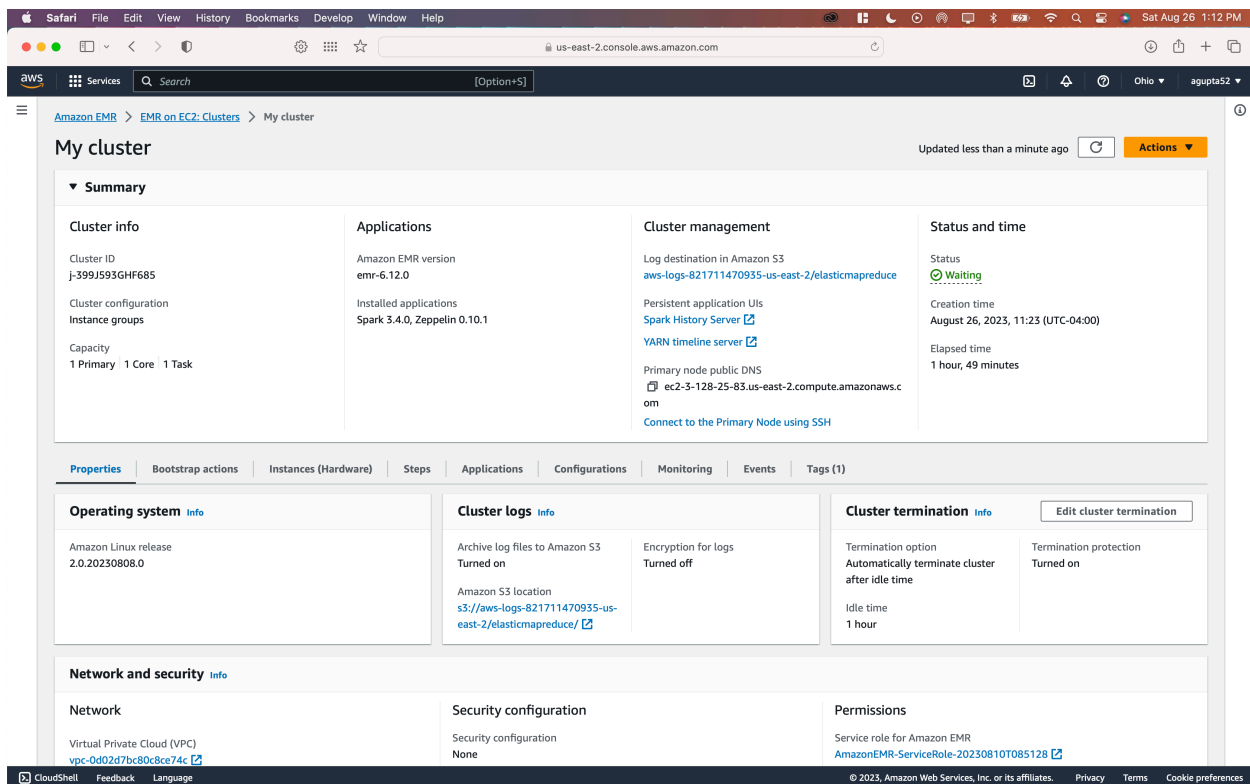


Fig 5. AWS EMR Cluster

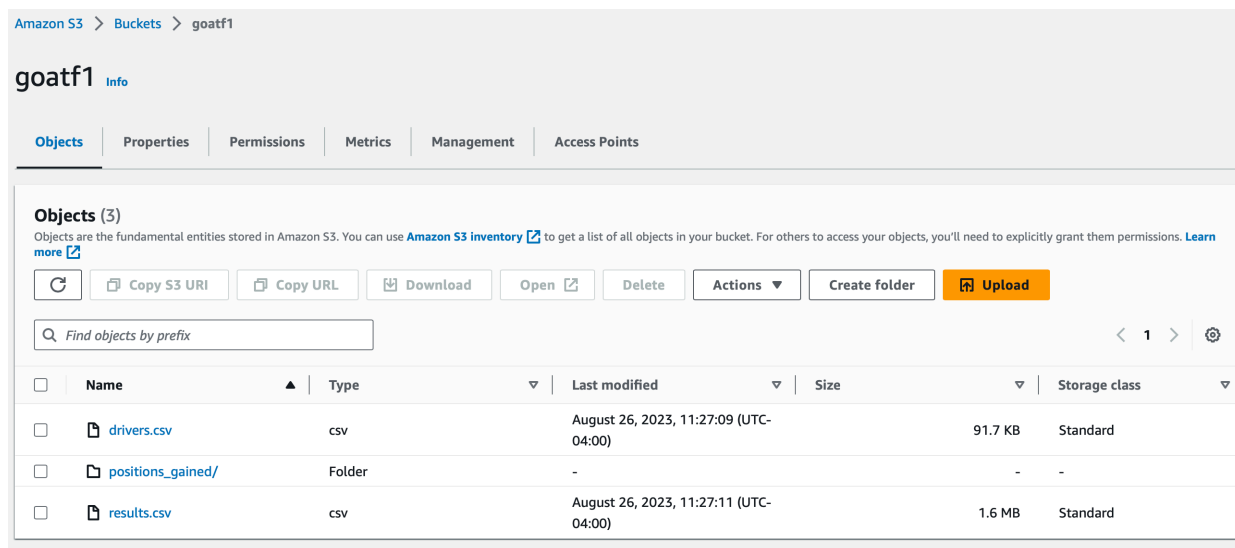



Fig 6. S3 bucket and output generated



3.4.0-amzn-0

History Server

Event log directory: hdfs:///var/log/spark/apps

Last updated: 2023-08-26 13:09:52

Client local time zone: America/New_York

Search:

Version	App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
3.4.0-amzn-0	application_1693063580806_0003	GoatFinder	2023-08-26 13:00:49	2023-08-26 13:01:18	29 s	hadoop	2023-08-26 13:01:18	Download
3.4.0-amzn-0	application_1693063580806_0002	GoatFinder	2023-08-26 12:22:18	2023-08-26 12:22:34	16 s	hadoop	2023-08-26 12:22:34	Download
3.4.0-amzn-0	application_1693063580806_0001	GoatFinder	2023-08-26 11:56:44	2023-08-26 11:57:23	39 s	hadoop	2023-08-26 11:57:23	Download

Showing 1 to 3 of 3 entries

[Show incomplete applications](#)

Figure 7: Server log

Figures 6 shows the S3 bucket along with the output folder generated after running the python script on EMR cluster. Figure 7 shows the history log.

Conclusion

In conclusion, this data analysis project enabled me to dive deep into Formula 1 race data using AWS services. The motivation behind choosing this dataset was my passion for Formula 1 and the desire to explore its statistical aspects. The chosen dataset provided comprehensive information about races, drivers, and teams, allowing me to calculate key metrics and gain valuable insights.

In the future, this project could be expanded by incorporating data visualization tools to create graphs and charts that visually represent the calculated statistics. Additionally, better calculations can be made and more metrics can be involved in finding the “greatest of all time”. Fernando Alonso has done the most number of overtakes in his career and can be considered the “GOAT”

This project not only satisfied my curiosity as a Formula 1 fan but also showcased the potential of cloud-based data analysis using AWS services. It highlighted how AWS EMR and S3 can be seamlessly integrated to process, analyze, and derive insights from large datasets.