POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Systems and Methods for Big and Unstructured Data Project

Author(s): **Angelo Prete**
Antonio Riverso
Andrea Sanvito

Group Number: **32**

Academic Year: 2024-2025

Contents

Contents **i**

1 Introduction **1**
1.1 Letterbox Database 1
1.2 Formula 1 Database 1

2 Data Wrangling **3**
2.1 Letterbox dataset data wrangling 3
2.2 Formula 1 dataset data wrangling 4

3 Dataset **7**
3.1 Letterbox dataset 7
3.2 Formula 1 dataset 13

4 Queries **21**
4.1 Letterbox dataset queries 21
4.1.1 Average rating, length and number of movies over the years 21
4.1.2 Most consistent directors (all movies with high ratings) 23
4.1.3 Average rating by genre 24
4.1.4 Most Productive Studios by Year 25
4.1.5 Genre popularity through the years 26
4.1.6 Actors who acted in highly rated movies 29
4.1.7 Number of good movies by years and by countries they were filmed in 31
4.1.8 Frequent actor-genre pairings (since 2000) 33
4.1.9 Frequent actor-director pairings in movies with good ratings and produced in USA 34
4.1.10 Theatrical vs Digital releases 35
4.2 Formula 1 dataset queries 37
4.2.1 List of drivers sorted based on wins count 37

4.2.2 Most important nations in Formula 1 38
4.2.3 Most difficult tracks for overtaking 40
4.2.4 Wins from different starting positions 42
4.2.5 Evolution of Formula 1 lap times 44
4.2.6 Evolution of Formula 1 pit stops times 45
4.2.7 Drivers analysis 47
4.2.8 Shift of teams' dominance 49
4.2.9 Importance of car in Formula 1 51
4.2.10 Most Grand Prix starts for Ferrari 54

List of Figures **57**

List of Tables **59**

1 | Introduction

For the project, we decided to analyze two datasets, the Formula 1 dataset and the Letterboxd dataset. Both of them can be found at Kaggle.

1.1. Letterbox Database

Our first analysis is on the Letterboxd Movies Dataset, that is a comprehensive collection of movie-related data derived from the Letterboxd website. It contains information about movies, such as titles, release dates, genres, average ratings, directors, actors, studios, and production details.

The raw dataset was cleaned and processed using a Python script. This script replaced unsupported characters in the CSV files to ensure compatibility with the database. After cleaning, we imported the dataset into a Neo4j database, creating nodes for movies, actors, genres, and studios, and establishing relationships like actors starring in movies and genres linked to films.

We decided to use Neo4j since it makes the process of querying connected data like movies, actors and directors simple. The queries rely heavily on relationships—such as finding actor-director pairings or tracking genre trends over time—and Neo4j is designed to navigate these links efficiently. Since it is schema-free handling data like ours with a lot of missing values becomes easier. Additionally, importing and linking data from CSV files is very straightforward.

1.2. Formula 1 Database

Our second analysis focuses on a dataset of Formula 1 records. In particular, it includes the results of all Formula 1 sessions (races, qualifying rounds, sprint races, etc.) over the years, together with supplementary data such as pit stop durations and race lap times.

As with the previous case, the raw dataset was cleaned and processed using Python. However, this time it required additional effort. For example we chose to import it after

removing non-essential fields, such as links to Wikipedia pages related to different entities.

We decided to exploit MongoDB's functionalities to query the dataset, as the structure of races and championships aligns naturally with the document-oriented model of Documental databases. Additionally, the schema-less nature of MongoDB allowed us to eliminate all null values.

2 | Data Wrangling

2.1. Letterbox dataset data wrangling

In order to ensure clean and consistent CSV data for import into Neo4j, we performed a small data wrangling step in Python by using the following function. This step replaced any irregular escaped quotes (e.g. \¨) in our raw CSV files with standard quotes, preventing errors during loading.

Python Script for Cleaning CSV Files

```
import os

def replace_backslash_quotes_in_folder(input_folder, output_folder):
    try:
        # Ensure the output folder exists
        os.makedirs(output_folder, exist_ok=True)

        # Iterate through all files in the input folder
        for filename in os.listdir(input_folder):
            if filename.endswith('.csv'):
                input_file = os.path.join(input_folder, filename)
                output_file = os.path.join(output_folder, filename)

                # Read and process the file
                with open(input_file, 'r') as infile:
                    content = infile.read()

                # Replace occurrences of \" with  and of \"" with ""
                updated_content = content.replace('\\""', '""')
                updated_content = updated_content.replace('\\"', '"')
```

```
            # Write the updated content to the output file
            with open(output_file, 'w') as outfile:
                outfile.write(updated_content)

            print(f"Processed file: {input_file} -> {output_file}")

    print("All .csv files processed successfully.")
except Exception as e:
    print(f"An error occurred: {e}")
```

2.2. Formula 1 dataset data wrangling

As of the Formula 1 dataset, it was processed through another Python script.

The CSV files were loaded with the *csv* Python module, processed through dictionaries, and dumped into a JSON file using the *json* Python module.

Additionally, null values were addressed through the same Python script simply by not adding the related field.

As the script is fairly extensive (around 400 lines of code), we only reported examples of helper functions, *time_ to_ milliseconds*, load functions, *load_ qualifyings*, and the unifying "wrapper" function, *build_ championships*.

Python Script for Processing CSV Files

```
import csv
import json

input_folder: str = "csv/"
null_str: str = "\\N"

def time_to_milliseconds(time: str) -> int:
    """
    Parses time expressed as "minutes:seconds.milliseconds" to just milliseconds.
    """

    if time == null_str:
```

```
        return time

    minutes_seconds, milliseconds = time.split(".")

    if ":" in minutes_seconds:
        minutes, seconds = minutes_seconds.split(":")
        return (int(minutes) * 60 + int(seconds)) * 10**3 + int(milliseconds)

    return int(seconds := minutes_seconds) * 10**3 + int(milliseconds)


def load_qualifyings(drivers: dict, constructors: dict) -> dict:
    """
    Reads qualifying.csv and inserts data into a dictionary.
    """
    qualifyings = {}
    next(reader := csv.reader(open(input_folder + "qualifying.csv")))

    for row in reader:
        id: int = int(row[1])
        qualifying_result = {}

        qualifying_result["position"] = int(row[5])
        qualifying_result["car"] = {
            "number": int(row[4]),
            "driver": drivers[int(row[2])],
            "constructor": constructors[int(row[3])],
        }
        if not (
            row[6] == null_str and
            row[7] == null_str and
            row[8] == null_str
        ):
            qualifying_result["times"] = {}
            if row[6] != null_str:
                qualifying_result["times"]["q1"] = time_to_ms(row[6])
            if row[7] != null_str:
```

```
                qualifying_result["times"]["q2"] = time_to_ms(row[7])
            if row[8] != null_str:
                qualifying_result["times"]["q3"] = time_to_ms(row[8])

        if id not in qualifyings:
            qualifyings[id] = {"results": [qualifying_result]}
        else:
            qualifyings[id]["results"].append(qualifying_result)

    return qualifyings


def build_championships() -> dict:
    """
    Returns a dictionary file containing all data.
    """
    statuses: dict = load_statuses()
    circuits: dict = load_circuits()
    drivers: dict = load_drivers()
    constructors: dict = load_constructors()

    races: dict = load_races(statuses, drivers, constructors)
    qualifyings: dict = load_qualifyings(drivers, constructors)
    sprints: dict = load_sprints(statuses, drivers, constructors)

    standings: dict = load_standings(drivers, constructors)

    weekends: dict = load_weekends(circuits, qualifyings, races, sprints, standings)

    championships: list = []
    for year, ws in weekends.items():
        championships.append({
            "year": year,
            "weekends": ws
        })

    return sorted(championships, key=(lambda x : x["year"]))
```

3 | Dataset

3.1. Letterbox dataset

As stated in the dataset page on Kaggle, the dataset is provided through csv files with the following structure:

- **movies.csv** - Basic information about films:
  - *id* - Movie identifier (primary key);
  - *name* - The name of the film;
  - *date* - Year of release of the film;
  - *tagline* - The slogan of the film;
  - *description* - Description of the film;
  - *minute* - Movie duration (in minutes);
  - *rating* - Average rating of the film.
- **actors.csv** - Actors who took part in the filming of films:
  - *id* - Movie identifier (foreign key);
  - *name* - Name;
  - *role* - Role.
- **crew.csv** - Film crew:
  - *id* - Movie identifier (foreign key);
  - *role* - Role in the film crew (director, screenwriter, etc.);
  - *name* - Name.
- **languages.csv** - In what languages the films were shot:
  - *id* - Movie identifier (foreign key);

  - *type* - Type (primary, conversational, etc.);
  - *language* - Film language.
- **studios.csv** - Film studios:
  - *id* - Movie identifier (foreign key);
  - *studio* - Film studio.
- **countries.csv** - Countries:
  - *id* - Movie identifier (foreign key);
  - *country* - Country.
- **genres.csv** - Film genres:
  - *id* - Movie identifier (foreign key);
  - *genre* - Film genre.
- **themes.csv** - Themes in films:
  - *id* - Movie identifier (foreign key);
  - *theme* - The theme of the film.
- **releases.csv** - Movie releases:
  - *id* - Movie identifier (foreign key);
  - *country* - Release country;
  - *date* - Release date of the film;
  - *type* - Release type (theatrical, television, etc.) of the film;
  - *rating* - Age rating of the film.
- **posters.csv** - Movie posters:
  - *id* - Movie identifier (foreign key);
  - *country* - URL address.

Since we don't need the posters for our analysis, we didn't import it in our Neo4j instance.

We first created the following indexes to speed-up both imports and queries:

```
CREATE INDEX index_on_movie_id FOR (m:Movie) ON m.id;
CREATE INDEX index_on_actor FOR (a:Actor) ON a.name;
CREATE INDEX index_on_crew FOR (c:Crew) ON c.name;
CREATE INDEX crew_role_index FOR (c:Crew) ON c.role;
CREATE INDEX index_on_languages FOR (l:Language) ON l.name;
CREATE INDEX index_on_studio FOR (s:Studio) ON s.name;
CREATE INDEX index_on_countries FOR (c:Country) ON c.name;
CREATE INDEX index_on_genres FOR (g:Genre) ON g.name;
CREATE INDEX index_on_themes FOR (t:Theme) ON t.name;
```

CSV files were imported using the following commands (through peridic commits, given their sizes):

first of all we imported the movies

```
:auto
LOAD CSV WITH HEADERS FROM 'file:///movies.csv' AS row
CALL(row) {
CREATE (:Movie {
id: toInteger(row.id),
name: row.name,
release_date: toInteger(row.date),
tagline: row.tagline,
description: row.description,
duration: toInteger(row.minute),
rating: toFloat(row.rating)
})
} IN TRANSACTIONS
```

then, all the other tables (linked through the foreign key movie id, which is present in each row)

```
:auto
LOAD CSV WITH HEADERS FROM 'file:///actors.csv' AS row
CALL(row) {
MATCH (m:Movie {id: toInteger(row.id)})
```

```
MERGE (a:Actor {name: row.name})
CREATE (a)-[:ACTED_IN {role: row.role}]->(m)
} IN TRANSACTIONS;
```

```
:auto
LOAD CSV WITH HEADERS FROM 'file:///crew.csv' AS row
CALL(row) {
MATCH (m:Movie {id: toInteger(row.id)})
MERGE (c:Crew {name: row.name})
CREATE (c)-[:WORKED_IN {role: row.role}]->(m)
} IN TRANSACTIONS;
```

```
:auto
LOAD CSV WITH HEADERS FROM 'file:///languages.csv' AS row
CALL(row) {
MATCH (m:Movie {id: toInteger(row.id)})
MERGE (l:Language {name: row.language})
CREATE (l)<-[:SPOKEN_IN {type: row.type}]-(m)
} IN TRANSACTIONS;
```

```
:auto
LOAD CSV WITH HEADERS FROM 'file:///studios.csv' AS row
CALL(row) {
MATCH (m:Movie {id: toInteger(row.id)})
MERGE (s:Studio {name: row.studio})
CREATE (s)-[:HAS_PRODUCED]->(m)
} IN TRANSACTIONS;
```

```
:auto
LOAD CSV WITH HEADERS FROM 'file:///countries.csv' AS row
CALL(row) {
MATCH (m:Movie {id: toInteger(row.id)})
MERGE (c:Country {name: row.country})
CREATE (c)<-[:FILMED_IN]-(m)
```

```
} IN TRANSACTIONS;
```

```
:auto
LOAD CSV WITH HEADERS FROM 'file:///genres.csv' AS row
CALL(row) {
MATCH (m:Movie {id: toInteger(row.id)})
MERGE (g:Genre {name: row.genre})
CREATE (g)<-[:BELONGS_TO_GENRE]-(m)
} IN TRANSACTIONS;
```

```
:auto
LOAD CSV WITH HEADERS FROM 'file:///themes.csv' AS row
CALL(row) {
MATCH (m:Movie {id: toInteger(row.id)})
MERGE (t:Theme {name: row.theme})
CREATE (t)<-[:ABOUT]-(m)
} IN TRANSACTIONS;
```

for releases, we didn't create new nodes; instead, we used the existing Country nodes and added "RElEASED_IN" relationships between movies and countries

```
:auto
LOAD CSV WITH HEADERS FROM 'file:///releases.csv' AS row
CALL(row) {
MATCH (m:Movie {id: toInteger(row.id)}), (c:Country {name:
↪ row.country})
CREATE (c)<-[:RELEASED_IN {type: row.type, rating: row.rating,
↪ realeasedIn: date(row.date)}]-(m)
} IN TRANSACTIONS;
```

An example of the produced structure can be seen in figure 3.1, where we show some nodes (one of each type present in the database) related to the movie "The Wolf of Wall Street".

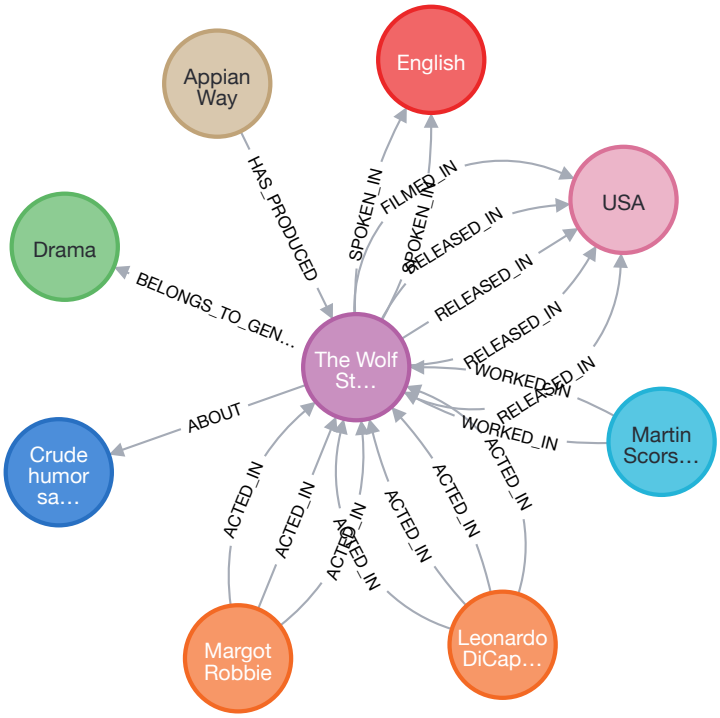

Figure 3.1: Truncated graph of "The Wolf of Wall Street" movie

3.2. Formula 1 dataset

As already mentioned, the dataset was taken from Kaggle. It can be found as a list of CSV files with the following structure:

- **circuits.csv** - Circuits in which at least one race has taken place:
  - *circuitId* - Circuit identifier (primary key);
  - *circuitRef* - Reference string for the circuit;
  - *name* - Name of the circuit;
  - *location* - City in which the circuit is located;
  - *country* - Country in which the circuit is located;
  - *lat* - Latitude coordinate;
  - *lng* - Longitude coordinate;
  - *alt* - Altitude coordinate;
  - *url* - Link to the Wikipedia page of the circuit;
- **constructor_results.csv** - Results and points for constructors after each race:
  - *constructorResultsId* - Result identifier (primary key);
  - *raceId* - Race identifier (foreign key);
  - *constructorId* - Constructor identifier (foreign key);
  - *points* - Count of points obtained by the race;
  - *status* - Status of the constructor after the race;
- **constructor_standings.csv** - Constructors standings after each race:
  - *constructorStandingsId* - Standings entry identifier (primary key);
  - *raceId* - Race identifier (foreign key);
  - *constructorId* - Constructor identifier (foreign key);
  - *points* - Count of points after the race;
  - *position* - Position in the standings;
  - *positionText* - Text version of the previous field;

  - *wins* - Count of wins so far;

- **constructors.csv** - Constructors general information:
  - *constructorId* - Constructor identifier (primary key);
  - *constructorRef* - Refernce string for the constructor;
  - *name* - Name of the constructor;
  - *nationality* - Name of the constructor;
  - *url* - Link to the Wikipedia page of the constructor;
- **driver_standings.csv** - Driver standings after each race:
  - *driverStandingsId* - Standings entry identifier (primary key);
  - *raceId* - Race identifier (foreign key);
  - *driverId* - Driver identifier (foreign key);
  - *points* - Count of points after the race;
  - *position* - Position in the standings;
  - *positionText* - Text version of the previous field;
  - *wins* - Count of wins so far;
- **drivers.csv** - Drivers general information:
  - *driverId* - Driver identifier (primary key);
  - *driverRef* - Reference string for the driver;
  - *number* - Number on the car of the driver;
  - *code* - Code used to display the driver during the races;
  - *forename* - Name of the driver;
  - *surname* - Surname of the driver;
  - *dob* - Date of birth of the driver;
  - *nationality* - Nationality of the driver;
  - *url* - Link to the Wikipedia page of the driver;
- **lap_times.csv** - Lap times registered at every session:

  - *raceId* - Race identifier (foreign key);
  - *driverId* - Driver identifier (foreign key);
  - *lap* - Lap of the race the time was registered in;
  - *position* - Position in which the driver registered the time;
  - *time* - Time of the lap (mm:ss.ms format);
  - *milliseconds* - Time of the lap (in milliseconds);
- **pit_stops.csv** - Information about pit stops taken in every race:
  - *raceId* - Race identifier (foreign key);
  - *driverId* - Driver identifier (foreign key);
  - *stop* - Counter of the stops;
  - *lap* - Lap of the race the pit stop was taken in;
  - *duration* - Duration of the pit stop (mm:ss.ms format);
  - *milliseconds* - Duration of the pit stop (in milliseconds);
- **qualifying.csv** - Results of qualifying sessions:
  - *qualifyId* - Qualifying entry identifier (primary key);
  - *raceId* - Race identifier (foreign key);
  - *driverId* - Driver identifier (foreign key);
  - *constructorId* - Constructor identifier (foreign key);
  - *number* - Number on the car of the driver;
  - *position* - Position in which the driver qualified;
  - *q1* - Lap time registered in the first qualifying session;
  - *q2* - Lap time registered in the second qualifying session;
  - *q3* - Lap time registered in the third qualifying session;
- **races.csv** - General information about race sessions:
  - *raceId* - Race identifier (primary key);
  - *year* - Year in which the race took place;

  - *round* - Number representing what round in the championship the race is;
  - *circuitId* - Circuit identifier (foreign key);
  - *name* - Name of the race;
  - *date* - Date in which the race takes place;
  - *time* - Time at which the race takes place;
  - *url* - Link to the Wikipedia page of the race;
  - *fp1_date* - Date of the first practice session;
  - *fp1_time* - Time of the first practice session;
  - *fp2_date* - Date of the second practice session;
  - *fp2_time* - Time of the second practice session;
  - *fp3_date* - Date of the third practice session;
  - *fp3_time* - Time of the third practice session;
  - *quali_date* - Date of the qualifying session;
  - *quali_time* - Time of the qualifying session;
  - *sprint_date* - Date of the sprint session;
  - *sprint_time* - Time of the sprint session;
- **results.csv** - Results of the race sessions:
  - *resultId* - Identifier for the single entry (primary key);
  - *raceId* - Race identifier (foreign key);
  - *driverId* - Driver identifier (foreign key);
  - *constructorId* - Constructor identifier (foreign key);
  - *number* - Number on the car of the driver;
  - *grid* - Starting position for the driver;
  - *position* - Finishing position for the driver;
  - *positionText* - Text version of the previous field;
  - *positionOrder* - Position in the finishing order;

  - *points* - Points gained by the driver;
  - *laps* - Laps done by the driver;
  - *time* - Time it took the driver to finish the race (hh:mm::ss.ms format);
  - *milliseconds* - Time it took the driver to finish the race (in milliseconds);
  - *fastestLap* - Lap in which the driver recorded his best lap time;
  - *rank* - Rank of the fastest lap recorded;
  - *fastestLapTime* - Time of the fastest lap (mm:ss.ms format);
  - *fastestLapSpeed* - Average speed of the driver during the fastest lap;
  - *status* - Status of the driver at the end of the race;
- **seasons.csv** - General information of the F1 seasons:
  - *year* - Year of the season (primary key);
  - *url* - Link to the Wikipedia page of the season;
- **sprint_results.csv** - Results of the sprints sessions:
  - *resultId* - Identifier for the single entry (primary key);
  - *raceId* - Race identifier (foreign key);
  - *driverId* - Driver identifier (foreign key);
  - *constructorId* - Constructor identifier (foreign key);
  - *number* - Number on the car of the driver;
  - *grid* - Starting position for the driver;
  - *position* - Finishing position for the driver;
  - *positionText* - Text version of the previous field;
  - *positionOrder* - Position in the finishing order;
  - *points* - Points gained by the driver;
  - *laps* - Laps done by the driver;
  - *time* - Time it took the driver to finish the sprint (hh:mm::ss.ms format);
  - *milliseconds* - Time it took the driver to finish the sprint (in milliseconds);

  - *fastestLap* - Lap in which the driver recorded his best lap time;
  - *fastestLapTime* - Time of the fastest lap (mm:ss.ms format);
  - *status* - Status of the driver at the end of the sprint;

- **status.csv** - Possible driver statuses at the end of a session:
  - *statusId* - Status identifier (primary key);
  - *status* - Status of the driver;

The entire **lap_times** and **constructor_results** files were excluded as they were not relevant to our queries. All data was processed to align more closely with the Document-oriented approach.

Importing data into MongoDB

To store data into MongoDB Atlas, we created a database named **formula-1** and a collection named **championships**.

Excluding the functions for loading CSV files into Python (as outlined in Section 2.2), additional scripts were employed to convert the data to JSON format and import them into the database. The two principal functions are shown below.

```
import json
from data_wrangling import build_championships

def main():
    """
    Dumps the dataset in a local folder.
    """
    json.dump(build_championships(), open("json/dataset.json", 'w'))

main()
```

```
import json
from pymongo import MongoClient
from psw import connection_string

# connect
```

```
client = MongoClient(connection_string)
database = client["formula-1"]

collection = database["championships"]

# load from file
data = json.load(open("json/dataset.json"))

# upload to MongoDB
collection.insert_many(data)
```

4 | Queries

4.1. Letterbox dataset queries

4.1.1. Average rating, length and number of movies over the years

With this query we calculated the average rating, duration, and number of movies released each year for films over 60 minutes, excluding those from 2024 (since the dataset is incomplete for the year). It helped us identify trends in movie quality and length over time, as plotted in the graphs below.

```
MATCH (m:Movie)
WHERE m.duration >= 60 AND m.release_date < 2024
WITH m.release_date AS YEAR, avg(m.rating) AS AVG_RATING,
 ↪  avg(m.duration) AS AVG_DURATION, count(m) AS N_MOVIES
RETURN YEAR, AVG_RATING, AVG_DURATION, N_MOVIES
ORDER BY YEAR DESC;
```

The query returns the following (truncated) result:

year	avg_rating	avg_duration	n_movies
2023	3.18	121.77	15598
2022	3.15	125.01	14869
2021	3.17	122.92	13841
2020	3.15	120.36	12372
2019	3.16	114.06	14173
2018	3.14	112.90	13473
2017	3.14	111.82	12963

Table 4.1: Average Rating, Duration, and Number of Movies by Year

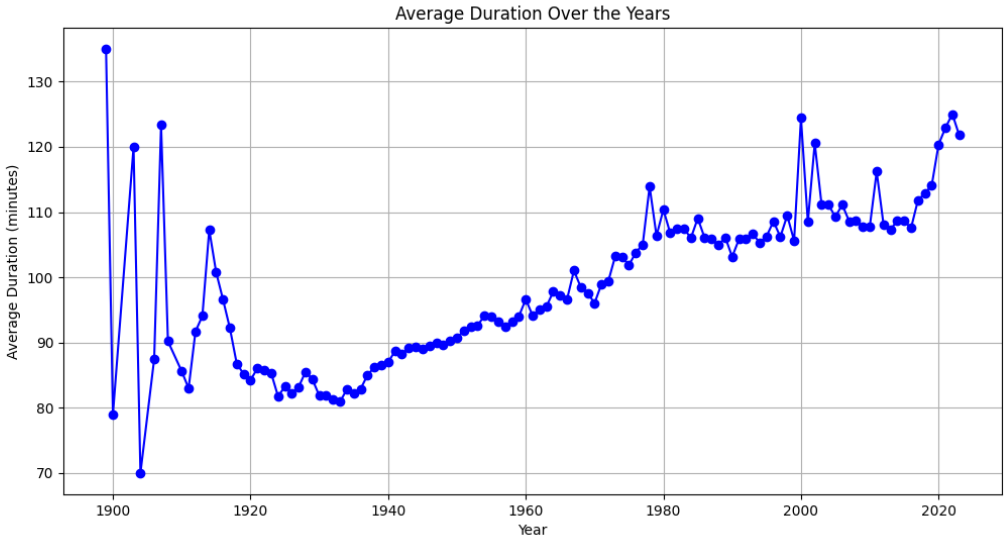

Figure 4.1: Average movie duration trend.

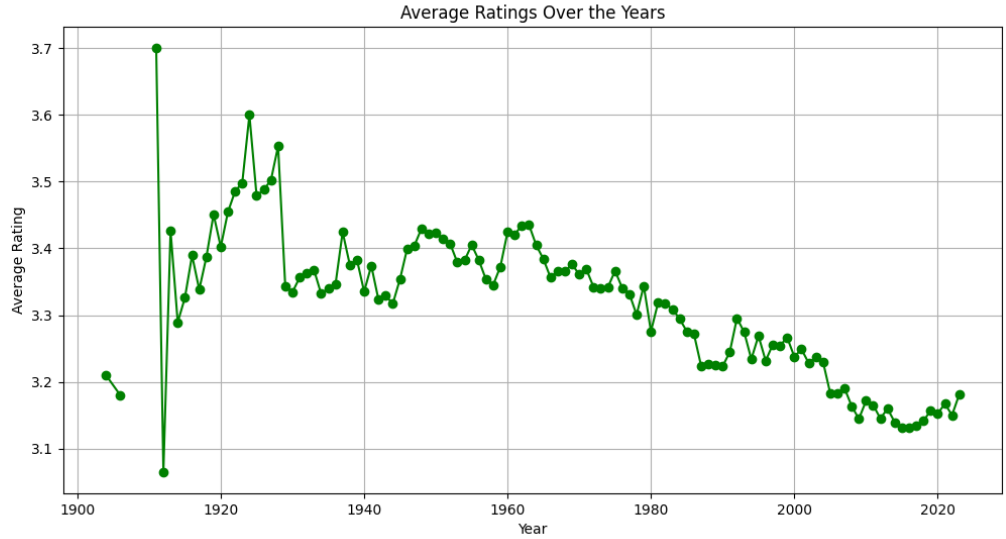

Figure 4.2: Average movie ratings trend.

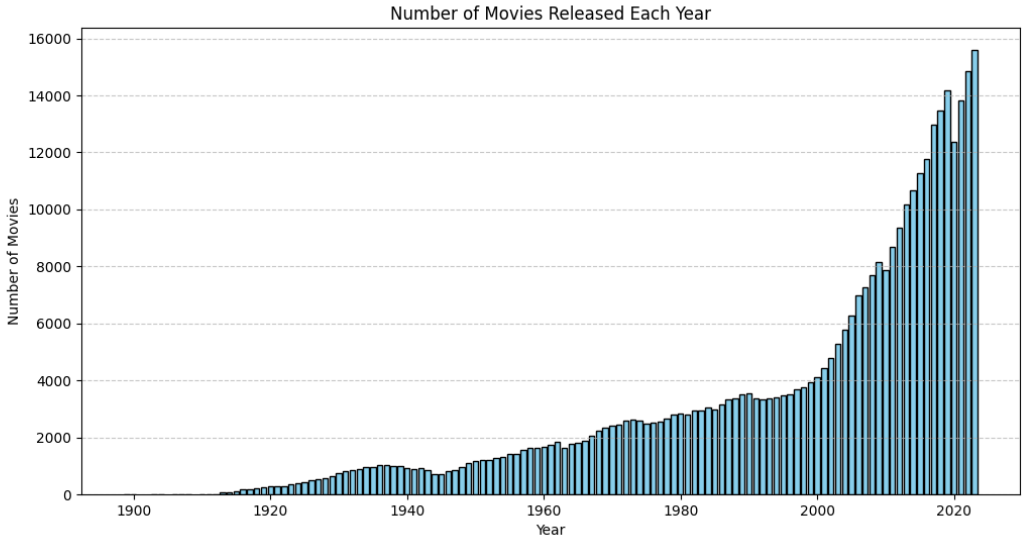

Figure 4.3: Movies count trend.

4.1.2. Most consistent directors (all movies with high ratings)

This query returns the list of directors found to be more consistent w.r.t the rating of their movies. To do it we used a scoring function (inspired to the tf-idf one) that multiplies the average rating of all movies by the logarithm (applied twice in this case) of their amount.

```
MATCH (d:Crew)-[:WORKED_IN {role: 'Director'}]-(m:Movie)
WHERE m.rating IS NOT NULL
WITH d.name as DIRECTOR, avg(m.rating) as AVG_RATING, COUNT(m) as
 ↪  N_MOVIES
RETURN DIRECTOR, AVG_RATING, N_MOVIES
ORDER BY AVG_RATING * log(log(N_MOVIES+1)) DESC
```

The query returns the following (truncated) result:

Table 4.2: Most consistent directors

#	Director	Avg. Rating	Number of Movies
1	Chuck Jones	3.47	223
2	Friz Freleng	3.37	184
3	William Hanna	3.50	124
4	Joseph Barbera	3.50	123

#	Director	Avg. Rating	Number of Movies
5	Jean-Luc Godard	3.54	105
6	Tex Avery	3.43	108
7	Dave Fleischer	3.43	104
8	Georges Méliès	3.20	173
9	Stan Brakhage	3.50	88
10	Agnès Varda	3.75	54
11	Martin Scorsese	3.74	54
12	Ingmar Bergman	3.67	60
13	Werner Herzog	3.57	66
14	John Ford	3.50	71
15	Krzysztof Kieślowski	3.74	44

4.1.3. Average rating by genre

We wrote this query because we where interested in finding which genres had better reviewed movies by users on Letterboxd. Having an higher score doesn't necessarily mean that the movies of that category are better, it could alse be that the users of the platform like the genre more. As we expected, the horror genre is the one populated by low quality movies.

```
MATCH (m:Movie)-[:BELONGS_TO_GENRE]->(g:Genre)
WITH g.name AS genre, avg(m.rating) AS rating
RETURN genre, rating
ORDER BY rating DESC
```

The query returns the following result:

Table 4.3: Genre average ratings

#	Genre	Rating
1	Documentary	3.52
2	Music	3.44
3	History	3.40
4	War	3.38
5	Animation	3.36
6	Drama	3.33

#	Genre	Rating
7	Western	3.29
8	Crime	3.24
9	Romance	3.19
10	Comedy	3.18
11	Mystery	3.18
12	Fantasy	3.15
13	TV Movie	3.11
14	Family	3.11
15	Adventure	3.10
16	Action	3.05
17	Thriller	3.05
18	Science Fiction	3.03
19	Horror	2.92

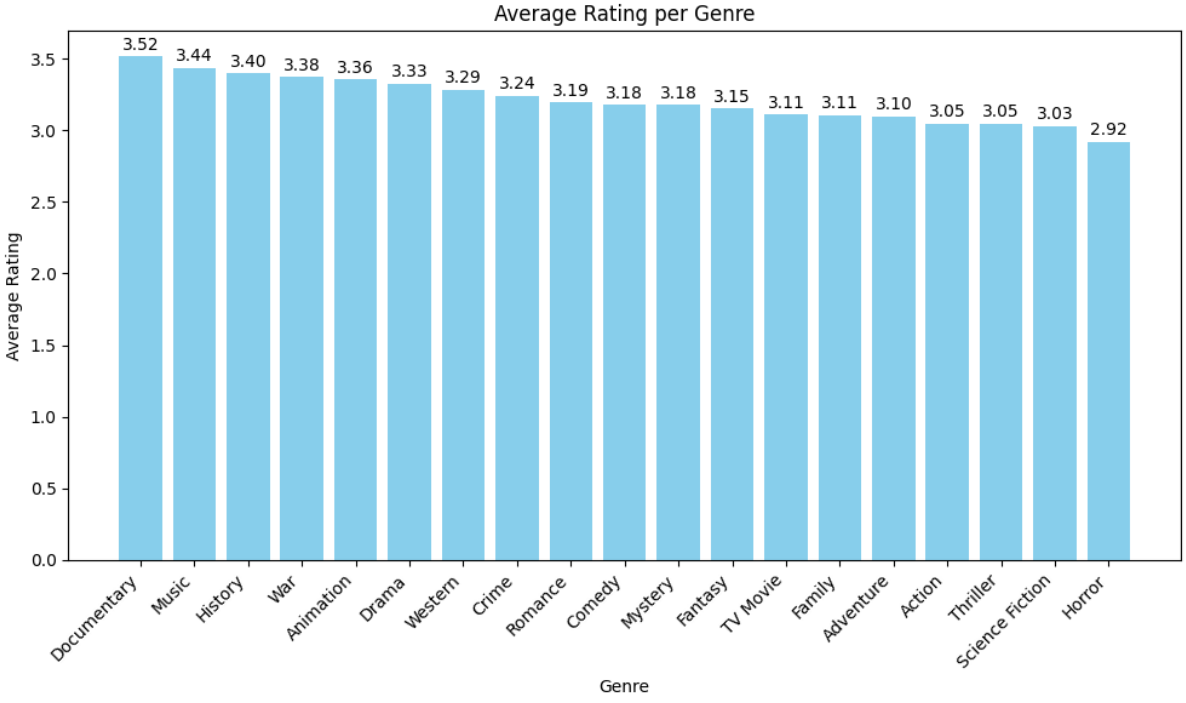

Figure 4.4: Average rating for genre.

4.1.4. Most Productive Studios by Year

With this query we found the most productive movie studios by year between 1900 and 2023, considering only movies with a rating of 3 or higher (i.e. at least 6/10). The

query calculates the number of qualifying movies produced by each studio for each year, returning the year, studio name, and count of movies. We decided to stop at 5 studios for each year.

```
MATCH (s:Studio)-[:HAS_PRODUCED]->(m:Movie)
WHERE m.release_date >= 1900 AND m.release_date <= 2023 AND m.rating
 ↪  >= 3
WITH s.name AS studio, m.release_date AS year, count(m) AS
 ↪  movies_count
ORDER BY year DESC, movies_count DESC
WITH year, COLLECT({studio: studio, movies_count: movies_count}) AS
 ↪  studios
UNWIND range(0, 4) AS idx
WITH year, studios[idx] AS studio_info
RETURN year, studio_info.studio AS studio, studio_info.movies_count AS
 ↪  movies_count
ORDER BY year DESC, movies_count DESC
```

The query returns the following (truncated to include only 2022 and 2034) result:

Table 4.4: Top Studios through years by movie count

Year	Studio	Movies Count
2023	France 2 Cinéma	32
2023	HBO Documentary Films	31
2023	Amazon Studios	29
2023	RAI Cinema	28
2023	France 3 Cinéma	27
2022	ARTE	38
2022	France 2 Cinéma	30
2022	Amazon Studios	24
2022	RAI Cinema	21
2022	ARTE France Cinéma	20

4.1.5. Genre popularity through the years

While this query might look simple, it highlights how genres alternated through the years in popularity. It shows how in recent years, while drama has always been a relevant genre,

comedy and documentaries have rose in popularity.

```
MATCH (m:Movie)-[:BELONGS_TO_GENRE]->(g:Genre)
WHERE m.release_date >= 1900 AND m.release_date <= 2023 AND m.rating
↪  >= 3
WITH m.release_date as year, g.name as genre, count(*) as cnt
RETURN year, genre, cnt
ORDER BY year DESC, cnt DESC
```

The output of this query has been truncated to show genre popularity only of 2022 and 2023:

Table 4.5: Genre popularity through the years

Year	Genre	Count
2023	Drama	1423
2023	Comedy	751
2023	Documentary	557
2023	Thriller	263
2023	Crime	256
2023	Animation	236
2023	Romance	233
2023	Action	174
2023	Mystery	155
2023	Horror	153
2023	Music	128
2023	Family	121
2023	Science Fiction	106
2023	Fantasy	103
2023	History	102
2023	Adventure	97
2023	TV Movie	64
2023	War	40
2023	Western	10
2022	Drama	1445
2022	Comedy	792
2022	Documentary	642
2022	Thriller	280

Year	Genre	Count
2022	Animation	266
2022	Romance	263
2022	Crime	241
2022	Horror	199
2022	Mystery	183
2022	Music	164
2022	Action	149
2022	Fantasy	135
2022	Science Fiction	120
2022	History	119
2022	Family	105
2022	Adventure	92
2022	TV Movie	78
2022	War	39
2022	Western	7

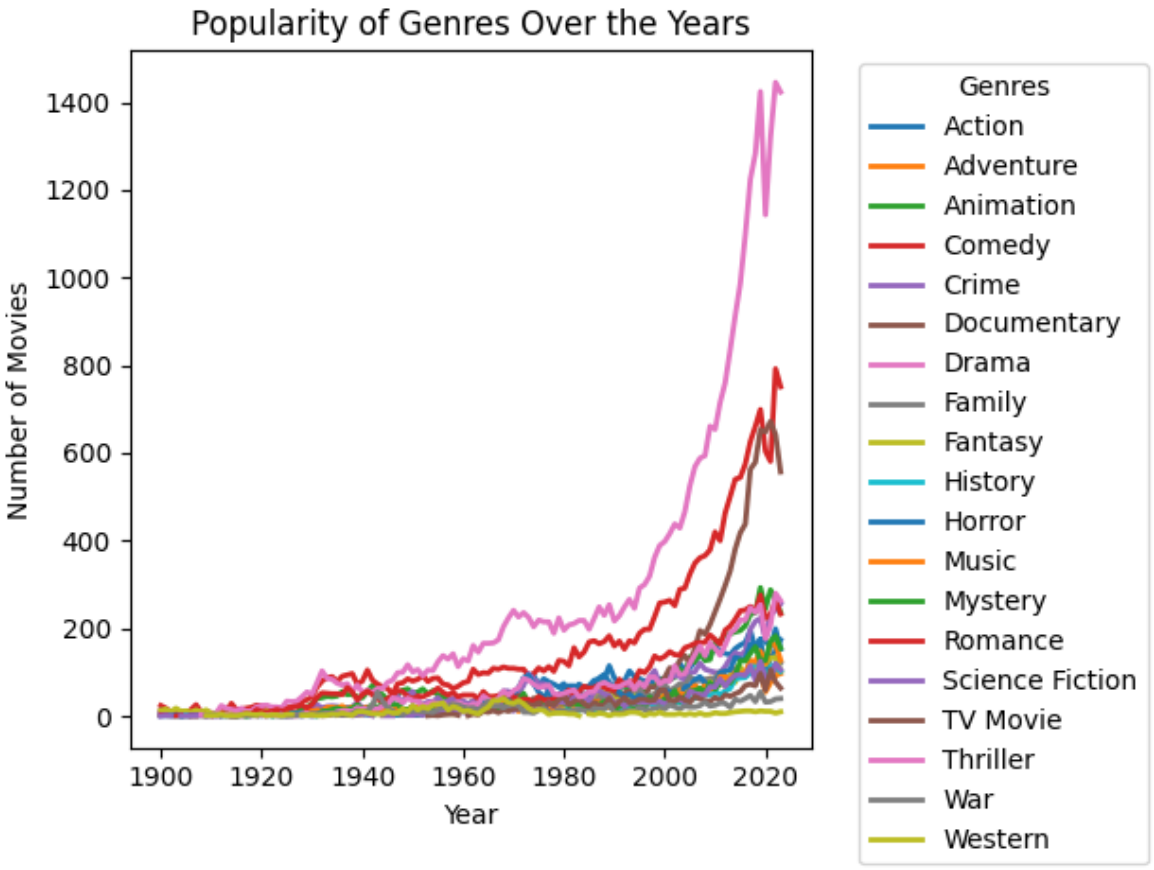

Figure 4.5: Genre popularity through the years.

4.1.6. Actors who acted in highly rated movies

Here, we wanted to see which actors (who acted in at least 20 movies) took part in movies that are highly rated. To speed up the query, we considered movies released since 2000 and filmed in USA.

```
MATCH (a:Actor)-[:ACTED_IN]->(m:Movie)-[:FILMED_IN]->(:Country {name:
↪  "USA"}), (m)-[:BELONGS_TO_GENRE]->(g:Genre)
WHERE m.release_date >= 2000 AND m.release_date <= 2024 AND m.rating
↪  is not null AND g.name <> "Documentary" AND g.name <> "Music"
WITH a.name as actor, avg(m.rating) as rating, COUNT(distinct m) as
↪  cnt
WHERE cnt >= 20
RETURN actor, rating, cnt
ORDER BY rating DESC
```

You can see the top 10 in the table below.

Actor	Rating	Count
Viggo Mortensen	3.64	21
Leonardo DiCaprio	3.64	24
Gregg Turkington	3.55	23
Dave Chappelle	3.53	23
Philip Seymour Hoffman	3.49	33
Tilda Swinton	3.49	42
Frank Wood	3.48	23
Ryan Gosling	3.47	30
Joaquin Phoenix	3.46	30
Carey Mulligan	3.46	20

Table 4.6: Actors and their ratings with count of ratings

4.1.7. Number of good movies by years and by countries they were filmed in

Here, we decided to find how many good movies were filmed across countries and years. To filter good movies, we decided to include only movies with a rating of at least 3.8/5.0.

```
MATCH (m:Movie)-[:FILMED_IN]->(c:Country)
WHERE m.release_date >= 1900 AND m.release_date <= 2023 AND m.rating
↪  >= 3.8
WITH m.release_date AS year, c.name AS country, count(m) AS
↪  movie_count
RETURN year, country, movie_count
ORDER BY year DESC, movie_count DESC;
```

Table 4.7: Count of good movies by country (truncated to include only some countries in 2023)

Year	Country	Movie Count
2023	USA	97
2023	UK	38
2023	France	28
2023	South Korea	27
2023	Japan	24
2023	Germany	11
2023	India	8
2023	Spain	8
2023	Brazil	7
2023	Italy	7
2023	Argentina	6
2023	Australia	4
2023	Belgium	4
2023	China	3
2023	Ireland	3
2023	Sweden	3
2023	Canada	3
2023	Poland	3
2023	Egypt	2
2023	Norway	2

Over the years, the number of movies with high ratings made by each country has gone up. But, when we look at the first query, it shows that the average ratings have gone down. This shows that even though more good movies are being made, the average review rating is dropping.

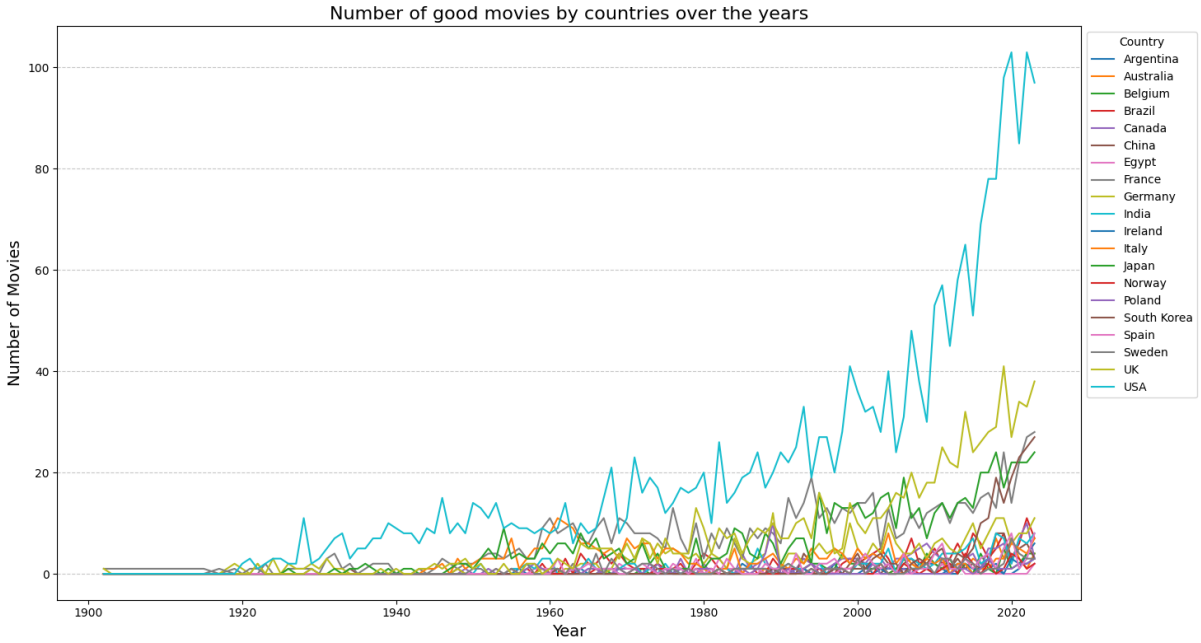

Figure 4.6: Count of good movies by country they were filmed in.

4.1.8. Frequent actor-genre pairings (since 2000)

Most actors often star in movies of the same genre, so we wanted to find the ones that are particularly "loyal" to a certain genre. To do it we used the following query:

```
MATCH (a:Actor)-[:ACTED_IN]->(m:Movie)-[:BELONGS_TO_GENRE]->(g:Genre),
↪  (m)-[:FILMED_IN]-(:Country {name: "USA"})
WHERE m.release_date >= 2000 AND m.release_date <= 2024 AND m.rating
↪  >= 3.5 AND g.name <> "Documentary" AND g.name <> "Music" AND
↪  g.name <> "Animation"
WITH a.name AS actor, g.name AS genre, COUNT(distinct m) AS
↪  appearances
RETURN actor, genre, appearances
ORDER BY appearances DESC
```

The truncated output with the first 20 pairings can be found in table 4.8.

Actor	Genre	Appearances
Tim Heidecker	Comedy	19
Tom Kenny	Comedy	18
Tilda Swinton	Drama	17
Jason Schwartzman	Comedy	16
Michael Shannon	Drama	16
Matt Damon	Drama	15
Bill Camp	Drama	15
Stan Lee	Action	15
Kathleen Barr	Family	15
Bill Murray	Comedy	15
Michael Stuhlbarg	Drama	15
Louis C.K.	Comedy	15
Cate Blanchett	Drama	15
Stan Lee	Adventure	14
Dee Bradley Baker	Comedy	14
Gregg Turkington	Comedy	14
Lauren Lopez	Comedy	14
Willem Dafoe	Drama	14
David Cross	Comedy	14
Jack Black	Comedy	13

Table 4.8: Actors, their genres, and number of appearances

4.1.9. Frequent actor-director pairings in movies with good ratings and produced in USA

The original idea behind this query was to find all pairings actor-director, but since the dataset was too big to do so (even with the help of indexes to speed up the query), we decided to restrict to movies filmed in USA and with a rating of at least 4/5.

```
MATCH (m:Movie)-[:BELONGS_TO_GENRE]->(g:Genre),
      (m)-[:FILMED_IN]-(:Country {name: "USA"})
WHERE m.release_date >= 1900 AND m.release_date <= 2024
  AND m.rating >= 4
WITH m, collect(g.name) AS genres
WHERE NONE(genre IN genres WHERE genre IN ["Documentary", "Music",
 ↪  "Animation"])
MATCH (m)<-[w:WORKED_IN {role: "Director"}]-(d:Crew)
MATCH (a:Actor)-[r:ACTED_IN]->(m)
WITH a.name AS actor, d.name AS director, count(m) AS cnt
```

```
RETURN actor, director, cnt
ORDER BY cnt DESC;
```

The result produced by the query is not surprising, as there are pairings like De Niro-Scorsese and DiCaprio-Tarantino.

Actor	Director	Count
Alfred Hitchcock	Alfred Hitchcock	9
Tim Heidecker	Eric Notarnicola	7
Gregg Turkington	Eric Notarnicola	7
Joe Estevez	Eric Notarnicola	6
Martin Scorsese	Martin Scorsese	6
Robert De Niro	Martin Scorsese	6
Charlie Chaplin	Charlie Chaplin	6
Henry Bergman	Charlie Chaplin	5
Samuel L. Jackson	Quentin Tarantino	5
Mark Proksch	Eric Notarnicola	5
Robert Duvall	Francis Ford Coppola	5
Bess Flowers	Alfred Hitchcock	5
Buster Keaton	Buster Keaton	5
Michael Caine	Christopher Nolan	4
Russ Fega	Christopher Nolan	4
Uma Thurman	Quentin Tarantino	4
Quentin Tarantino	Quentin Tarantino	4
Leonardo DiCaprio	Martin Scorsese	4
Kyle MacLachlan	David Lynch	4
Jack Nance	David Lynch	4

Table 4.9: Actor-Director pairings

4.1.10. Theatrical vs Digital releases

For the last query, we wanted to see how movie releases changed since the introduction of online renting and streaming.

```
MATCH (m:Movie)-[r:RELEASED_IN]->(c:Country)
WHERE r.type IN ["Theatrical", "Digital"] AND m.release_date =
↪  date(r.realeasedIn).year AND date(r.realeasedIn).year <= 2023
WITH date(r.realeasedIn).year AS releaseYear, r.type AS releaseType,
↪  COUNT(DISTINCT m) AS totalMovies
RETURN releaseYear, releaseType, totalMovies
```

```
ORDER BY releaseYear DESC, totalMovies DESC
```

Release Year	Release Type	Total Movies
2023	Theatrical	12161
2023	Digital	11671
2022	Digital	11033
2022	Theatrical	10984
2021	Digital	12381
2021	Theatrical	10072
2020	Digital	12645
2020	Theatrical	9466
2019	Theatrical	12323
2019	Digital	7174
2018	Theatrical	12629
2018	Digital	5766

Table 4.10: Release Year, Release Type, and Total Movies

By looking at the graph we generated from the query, we can notice how during covid digital releases overtook theatrical ones (as expected) and, more in general, how digital releases have been on the rise since early 2000s.

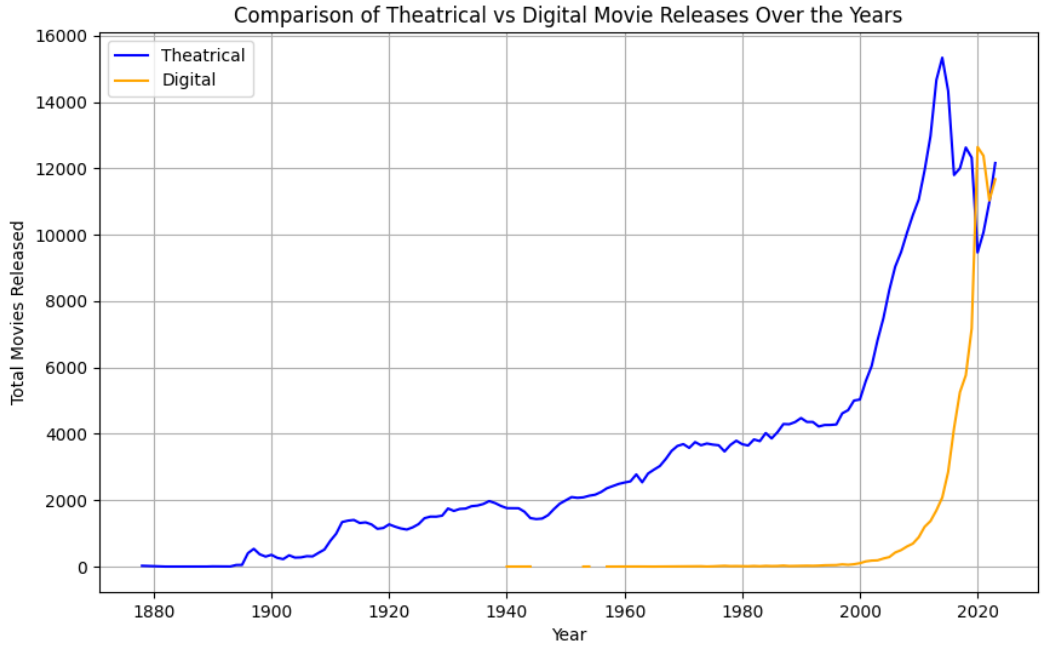

Figure 4.7: Digital vs Theatrical releases.

4.2. Formula 1 dataset queries

4.2.1. List of drivers sorted based on wins count

Drivers in Formula 1 dream since being children about winning their first race in the most important category of the motorsport world. Through this query we aim to seek who the most winning drivers ever are, and we list the top 10.

```
query = collection.aggregate([
    { "$unwind": { "path": "$weekends" } },
    { "$unwind": { "path": "$weekends.race.results" } },
    { "$match": { "weekends.race.results.position": 1 } },
    {
        "$group": {
            "_id": {"driver": "$weekends.race.results.car.driver"},
            "wins": {"$sum": 1}
        }
    },
    { "$sort": {"wins": -1} },
    { "$limit": 10 },
    {
        "$project": {
            "_id": 0,
            "Name": "$_id.driver.name",
            "Surname": "$_id.driver.surname",
            "Wins": "$wins",
        }
    }
])
```

Name	Surname	Wins
Lewis	Hamilton	104
Michael	Schumacher	91
Max	Verstappen	61
Sebastian	Vettel	53
Alain	Prost	51
Ayrton	Senna	41
Fernando	Alonso	32
Nigel	Mansell	31
Jackie	Stewart	27
Niki	Lauda	25

Table 4.11: Top 10 drivers in terms of race wins

4.2.2. Most important nations in Formula 1

For the next query, divided into two, our objective is to identify the most relevant countries in Formula 1. The sport is always perceived as being Europe-centric, and we will try to confirm or disprove the theory.

```
drivers = collection.aggregate([
    { "$unwind": {"path": "$weekends"} },
    { "$unwind": {"path": "$weekends.race.results"} },
    {
        "$group": {
            "_id": {"driver": "$weekends.race.results.car.driver"}
        }
    },
    {
        "$group": {
            "_id": {"nationality": "$_id.driver.nationality"},
            "count": {"$sum": 1}
        }
    },
    { "$sort": {"count": -1} },
    { "$limit": 10 },
    {
        "$project": {
            "_id": 0,
```

```
            "Country": "$_id.nationality",
            "Drivers Count": "$count",
        }
    }
])

constructors = collection.aggregate([
    { "$unwind": {"path": "$weekends"} },
    { "$unwind": {"path": "$weekends.race.results"} },
    {
        "$group": {
            "_id": {"constructor":
            ↪  "$weekends.race.results.car.constructor"}
        }
    },
    {
        "$group": {
            "_id": {"nationality": "$_id.constructor.nationality"},
            "count": {"$sum": 1}
        }
    },
    { "$sort": {"count": -1} },
    { "$limit": 10 },
    {
        "$project": {
            "_id": 0,
            "Country": "$_id.nationality",
            "Constructors Count": "$count",
        }
    }
])
```

Country	Drivers Count
British	166
American	158
Italian	99
French	73
German	50
Brazilian	32
Swiss	23
Argentine	23
Belgian	23
South African	23

Table 4.12: Count of drivers by country

Country	Constructors Count
British	86
American	38
Italian	30
French	13
German	10
Brazilian	5
Swiss	5
Argentine	3
Belgian	3
South African	2

Table 4.13: Count of constructors by country

The results show a big percentage of European countries, thus confirming what is perceived of the sport.

4.2.3. Most difficult tracks for overtaking

Different circuits have different characteristics, including track width. This implies overtaking another driver also depends on track features. Let us try to infer on which circuits it is more difficult to fight another driver, counting the races that were won by starting from pole position.

```
query = collection.aggregate([
    { "$unwind": {"path": "$weekends"} },
    { "$unwind": {"path": "$weekends.race.results"} },
    { "$match": {"weekends.race.results.position": 1} },
```

```
    {
        "$group": {
            "_id": {"circuit": "$weekends.circuit"},
            "wins_from_pole": {
                "$sum": {
                    "$cond": { "if": { "$eq":
                    ↪  ["$weekends.race.results.grid", 1] }, "then":
                    ↪  1, "else": 0 }
                }
            },
            "number_of_races": {"$sum": 1}
        }
    },
    { "$match": { "number_of_races": {"$gte": 5} } },
    {
        "$addFields": {
            "wins_percentage": { "$divide": ["$wins_from_pole",
            ↪  "$number_of_races"] }
        }
    },
    { "$sort": {"wins_percentage": -1} },
    { "$limit": 10 },
    {
        "$project": {
            "_id": 0,
            "Circuit": "$_id.circuit.location.city",
            "Wins From Pole": "$wins_from_poles",
            "Races Count": "$number_of_races",
            "Win Percentage": "$wins_percentage"
        }
    }
])
```

Circuit	Wins from Pole	Races Count	Wins Percentage
Montmelò	24	34	0.71
Istanbul	6	9	0.67
Abu Dhabi	10	15	0.67
Marina Bay	9	14	0.64
Le Castellet	11	18	0.61
Valencia	3	5	0.6
Shanghai	10	17	0.59
Suzuka	18	34	0.53
New York State	10	20	0.5
Liverpool	3	6	0.5

Table 4.14: Ratio of races won from pole position

Tracks related to a high percentage should be the ones having less overtaking possibilities. As seen from the results, that is true for the majority of them, but not for all: the track found in Istanbul presents a lot of overtaking spots. Additionally, some tracks that offer very few overtaking spots in reality, such as Monaco, are unexpectedly missing from the list.

4.2.4. Wins from different starting positions

We now know the list of circuits on which it is most difficult to overtake, discovered by exploring how many races were won from pole position. One might ask about the other starting positions and how many races where won from them. That is what we will try to understand with the next query.

```
query = collection.aggregate([
    { "$unwind": { "path": "$weekends" } },
    { "$unwind": { "path": "$weekends.race.results" } },
    { "$match": { "weekends.race.results.position": 1 } },
    {
        "$group": {
            "_id": {"grid": "$weekends.race.results.grid"},
            "wins": {"$sum": 1}
        }
    },
    { "$sort": {"_id.grid": 1} },
    {
```

```
        "$project": {
            "_id": 0,
            "Grid": "$_id.grid",
            "Wins Count": "$wins"
        }
    }
])
```

Grid	Wins Count
1	476
2	265
3	136
4	66
5	49
6	40
7	23
8	17
9	5
10	12
11	5
12	3
13	4
14	7
15	1
16	2
17	2
18	1
19	1
22	1

Table 4.15: Count of wins based on the starting position

As expected, the likelihood of winning decreases progressively with the starting positions further from pole.

4.2.5. Evolution of Formula 1 lap times

Being the pinnacle of motorsport, Formula 1 is in continuous evolution: lap times are the first element in the Formula 1 world which is affected. It might be very interesting to know the entity of this change throughout the years.

To do so, results from qualifying on one of the most famous and long lasting tracks, Silverstone, are used.

```
query = collection.aggregate([
    { "$unwind": {"path": "$weekends"} },
    { "$match": {"weekends.circuit.location.city": "Silverstone"} },
    {
        "$project": {
            "year": 1,
            "weekends.qualifying": 1
        }
    },
    { "$unwind": {"path": "$weekends.qualifying.results"} },
    { "$sort": {"weekends.qualifying.results.times.q1": 1} },
    { "$match": {"weekends.qualifying.results.times.q1": {"$exists":
    ↪  True}} },
    {
        "$group": {
            "_id": {"year": "$year"},
            "fastest_time": {"$first":
            ↪  "$weekends.qualifying.results.times.q1"}
        }
    },
    { "$sort": {"_id.year": 1} },
    {
        "$project": {
            "_id": 0,
            "Year": "$_id.year",
            "Fastest Time": "$fastest_time"
        }
    }
```

```
])
```

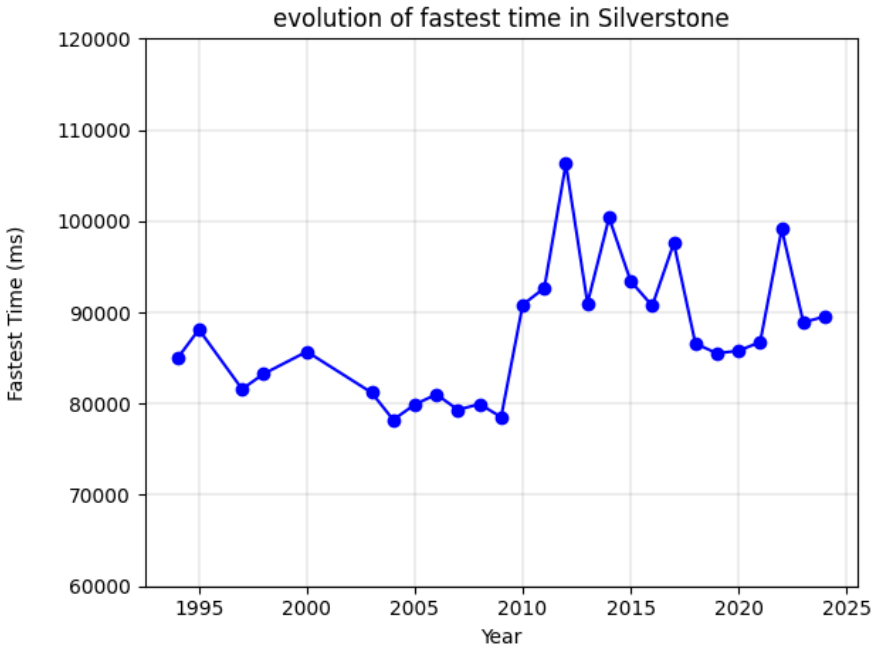

Figure 4.8: Evolution of lap times in Silverstone

Unfortunately, qualifying data is only available from year 1994 onwards. The curve generally shows a descending trend, with the exception around 2014, when new car regulations were introduced. Random peaks in the diagram are related to years where qualifying sessions were carried out in wet conditions, that is, while it was raining.

4.2.6. Evolution of Formula 1 pit stops times

Formula 1 is about finding perfection in every aspect of racing, including pit stops. The graph shows the evolution of pit stop times in the Monza circuit. Unfortunately, pit stop data is only available from 2011 onwards, for all circuits.

```
query = collection.aggregate([
    { "$unwind": { "path": "$weekends" } },
    { "$match": { "weekends.race.pit_stops": { "$exists": True } } },
    { "$unwind": { "path": "$weekends.race.pit_stops" } },
    {
        "$group": {
            "_id": {
```

```
                "year": "$year",
                "circuit": "$weekends.circuit"
            },
            "avg_time": { "$avg": "$weekends.race.pit_stops.duration"
            ↪ }
        }
    },
    { "$match": {"_id.circuit.location.city": "Monza"} },
    {
        "$project": {
            "_id": 0,
            "Year": "$_id.year",
            "Average Time": "$avg_time"
        }
    },
    { "$sort": {"Year": 1} }
])
```

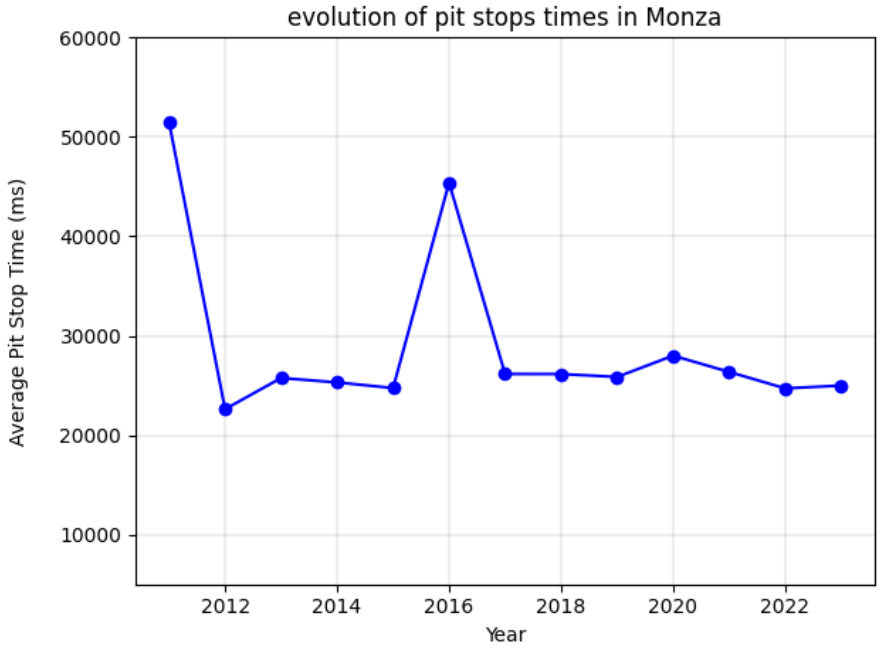

Figure 4.9: Evolution of pit stop times in Monza

Over the past decade, pit stop times in Formula 1 have remained unchanged. This stagnation can be attributed to two main factors. First, teams have reached the physical

lower limit of what is achievable. Second, the FIA, the governing body that establishes Formula 1 regulations, periodically introduces new rules for pit stops, preventing teams from fully optimizing their processes.

4.2.7. Drivers analysis

In *Section 4.2.1*, a query was performed to identify the drivers with most wins during their career. However, in the world of motorsport, victories often depend on having a competitive car, making it impossible to determine the best drivers solely from this list.

To uncover the top talents ever, a more reliable metric can be obtained by comparing drivers to their teammates, who drive identical cars: the higher the percentage of races in which a driver beats their teammate, the greater their talent is likely to be.

For the analysis, only drivers completing at least 100 races (around 5-6 years of career) were considered.

```
query = collection.aggregate([
    { "$unwind": { "path": "$weekends" } },
    { "$addFields": { "first_result": "$weekends.race.results" } },
    {
        "$project": {
            "first_result": 1,
            "second_result": "$weekends.race.results"
        }
    },
    { "$unwind": { "path": "$first_result" } },
    { "$unwind": { "path": "$second_result" } },
    {
        "$match": {
            "$expr": {
                "$and": [
                    { "$ne": ["$first_result.car.driver",
                    ↪  "$second_result.car.driver"] },
                    { "$eq": ["$first_result.car.constructor",
                    ↪  "$second_result.car.constructor"] }
                ]
            }
```

```
        }
    },
    {
        "$group": {
            "_id": {"driver": "$first_result.car.driver"},
            "wins": {
                "$sum": {
                    "$cond": { "if": { "$lt": ["$first_result.order",
                    ↪  "$second_result.order"] }, "then": 1, "else":
                    ↪  0 }
                }
            },
            "duels": {"$sum": 1}
        }
    },
    {
        "$project": {
            "driver": "$_id.driver",
            "wins": "$wins",
            "duels": "$duels",
            "wins_percentage": { "$divide": ["$wins", "$duels"] }
        }
    },
    { "$sort": { "wins_percentage": -1 } },
    { "$match": {"duels": { "$gt": 100 }} },
    { "$limit": 10 },
    {
        "$project": {
            "_id": 0,
            "Name": "$_id.driver.name",
            "Surname": "$_id.driver.surname",
            "Wins": "$wins",
            "Duels Count": "$duels",
            "Win Percentage": "$wins_percentage",
        }
    }
```

```
])
```

Name	Surname	Wins	Duels	Win Percentage
Duane	Carter	93	117	0.79
Juan	Fangio	189	246	0.77
Bill	Vukovich	80	107	0.75
Jim	Clark	161	218	0.74
Art	Cross	78	107	0.73
Max	Verstappen	142	197	0.72
Mika	Salo	78	111	0.70
Bruce	McLaren	174	251	0.69
Jack	McGrath	94	137	0.69
Fernando	Alonso	265	392	0.68

Table 4.16: Percentage of duels with teammate won

4.2.8. Shift of teams' dominance

Now, our focus is on teams rather than drivers. How did the most important constructors, i.e. the ones that won the most championships, behave over the course of the years?

It is possible to study it by analyzing their final positions in the constructors championships through time.

```
query = collection.aggregate([
    { "$unwind": { "path": "$weekends" } },
    { "$sort": { "weekends.round": -1 } },
    {
        "$group": {
            "_id": {"year": "$year"},
            "weekend": { "$first": "$weekends" }
        },
    },
    {
        "$project": {
            "year": "$_id.year",
            "weekends": "$weekend"
        }
```

```
    },
    { "$unwind": {"path": "$weekends.standings.constructors"} },
    {
        "$match": {
            "weekends.standings.constructors.constructor.name": {
                "$in": ["Ferrari", "Mercedes", "Red Bull", "McLaren",
                ↪  "Williams", "Renault", "Team Lotus", "Brabham",
                ↪  "Cooper"]
            }
        }
    },
    {
        "$project": {
            "constructor":
            ↪  "$weekends.standings.constructors.constructor.name",
            "year": "$_id.year",
            "position": "$weekends.standings.constructors.position"
        }
    },
    {
        "$sort": {
            "constructor": 1,
            "year": 1
        }
    },
    {
        "$project": {
            "_id": 0,
            "Year": "$_id.year",
            "Constructor": "$constructor",
            "Position": "$position"
        }
    }
])
```

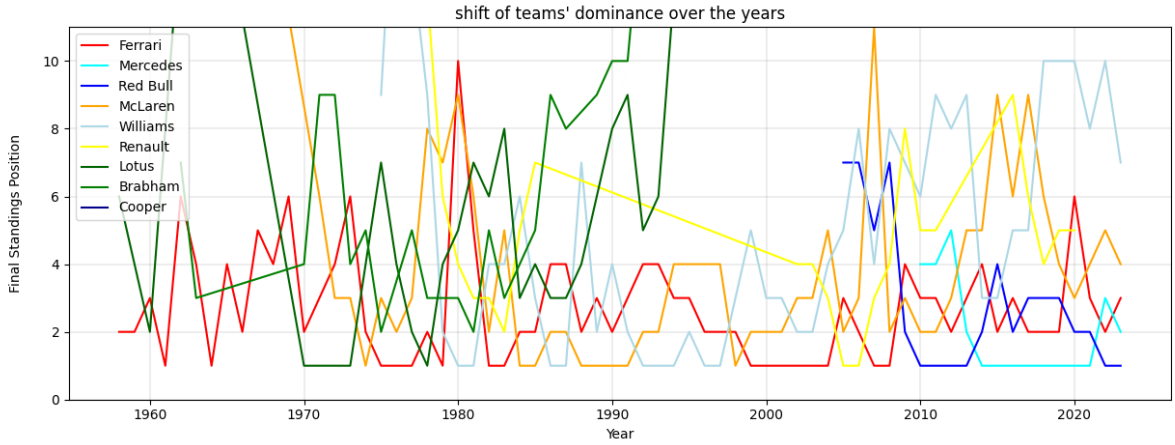

Figure 4.10: Shift of teams' dominance over the years

Both Mercedes and Red Bull teams managed to finish the year in a good position since they entered the world of Formula 1.

Williams and McLaren teams, instead, fluctuated more in terms of performances, but also started their career a very long time ago, making them experienced and reliable.

However, there is one team, Ferrari, who managed to keep a good position during their stint even though they never missed a race since 1950, making them the most influential team to ever exist in Formula 1.

4.2.9. Importance of car in Formula 1

In Formula 1, it is only possible to win if given a car capable of winning. With the following query we count the number of world champion drivers who won with a car that also won the world constructors championship, trying to infer how important a car is for a Grand Prix win.

Note: due to the limitations regarding the sorting functionality for the MongoDB Atlas free-tier, we were obliged to restrict the number of entries. In particular, we only considered championships from 1975 onwards.

```
query = collection.aggregate([
    { "$unwind": { "path": "$weekends" } },
    { "$sort": { "weekends.round": -1 } },
    {
        "$group": {
```

```
            "_id": {"year": "$year"},
            "year": {"$first": "$year"},
            "weekend": {"$first": "$weekends"}
        }
    },
    {
        "$project": {
            "_id": 0,
            "year": 1,
            "weekend": 1
        }
    },
    { "$match": { "year": { "$gt": 1975 } } },
    { "$unwind": { "path": "$weekend.standings.drivers" } },
    { "$sort": { "weekend.standings.drivers.position": 1 } },
    {
        "$group": {
            "_id": {"year": "$year"},
            "year": {"$first": "$year"},
            "weekend": {"$first": "$weekend"}
        }
    },
    { "$unwind": { "path": "$weekend.standings.constructors" } },
    { "$sort": { "weekend.standings.constructors.position": 1 } },
    {
        "$group": {
            "_id": {"year": "$year"},
            "year": {"$first": "$year"},
            "weekend": {"$first": "$weekend"}
        }
    },
    {
        "$project": {
            "_id": 0,
            "year": 1,
            "weekend.race": 1,
            "weekend.standings": 1
```

```
        }
    },
    { "$unwind": { "path": "$weekend.race.results" } },
    {
        "$match": {
            "$and": [
                { "$expr": { "$eq":
                ↪  ["$weekend.race.results.car.driver.surname",
                ↪  "$weekend.standings.drivers.driver.surname"] } },
            ]
        }
    },
    {
        "$project": {
            "_id": 0,
            "year": 1,
            "weekend.race.results": 1,
            "weekend.standings": 1
        }
    },
    {
        "$match": {
            "$and": [
                { "$expr": { "$eq":
                ↪  ["$weekend.race.results.car.constructor.name",
                ↪  "$weekend.standings.constructors.constructor.name"]
                ↪  } },
            ]
        }
    },
    # {
    #     "$project": {
    #         "Year": "$year",
    #         "Name": "$weekend.race.results.car.driver.name",
    #         "Surname": "$weekend.race.results.car.driver.surname",
    #         "Constructor":
    ↪  "$weekend.race.results.car.constructor.name"
```

```
    #     }
    # },
    # { "$sort": {"Year": -1} },
    # { "$limit": 10 },
    { "$count": "Both Championships Winners" }
])
```

Both Championships Winners	38

Table 4.17: Times both championships were won by the same team

Only seasons since 1975 were considered: this implies that 78% of driver championships were won with the fastest car on the grid.

We also wanted to showcase the code for displaying the last ten occasions in which both championships were won by a team. If we were to run the commented code instead of the "$count" function, the table below would be generated.

Year	Name	Surname	Constructor
2023	Max	Verstappen	Red Bull
2022	Max	Verstappen	Red Bull
2020	Lewis	Hamilton	Mercedes
2019	Lewis	Hamilton	Mercedes
2018	Lewis	Hamilton	Mercedes
2017	Lewis	Hamilton	Mercedes
2016	Nico	Rosberg	Mercedes
2015	Lewis	Hamilton	Mercedes
2014	Lewis	Hamilton	Mercedes
2013	Sebastian	Vettel	Red Bull

Table 4.18: Times both championships were won by the same team

In the last 12 years, a championship was won with a non championship-winning car only one time.

4.2.10. Most Grand Prix starts for Ferrari

As seen from *Section 4.2.8*, Ferrari is the greatest constructor in Formula 1, ever.

To conclude our dataset analysis, we can identify the top 15 drivers in terms of races completed for the most influential team in Formula 1, and how many of these were eventually

won.

```
query = collection.aggregate([
    { "$unwind": { "path": "$weekends" } },
    { "$unwind": { "path": "$weekends.race.results" } },
    {
        "$match": {
            "weekends.race.results.car.constructor.name": "Ferrari"
        }
    },
    {
        "$group": {
            "_id": {"driver": "$weekends.race.results.car.driver"},
            "wins": {
                "$sum": {
                    "$cond": { "if": { "$eq":
                    ↪  ["$weekends.race.results.position", 1] },
                    ↪  "then": 1, "else": 0 }
                }
            },
            "races": {"$sum": 1}
        }
    },
    {
        "$project": {
            "_id": 0,
            "Name": "$_id.driver.name",
            "Surname": "$_id.driver.surname",
            "Wins": "$wins",
            "Races": "$races",
        }
    },
    { "$sort": {"Races": -1} },
    { "$limit": 15 }
])
```

Name	Surname	Wins	Races
Michael	Schumacher	72	181
Kimi	Raikkonen	10	152
Felipe	Massa	11	140
Sebastian	Vettel	14	119
Charles	Leclerc	6	116
Rubens	Barrichello	9	104
Fernando	Alonso	11	96
Gerharg	Berger	5	96
Michele	Alboreto	3	80
Jean	Alesi	1	79
Carlos	Sainz	3	77
Clay	Regazzoni	4	73
Gilles	Villeneuve	6	67
Eddie	Irvine	4	65
Niki	Lauda	15	58

Table 4.19: Most important drivers for Ferrari

List of Figures

3.1 Truncated graph of "The Wolf of Wall Street" movie 12

4.1 Average movie duration trend. 22
4.2 Average movie ratings trend. 22
4.3 Movies count trend. 23
4.4 Average rating for genre. 25
4.5 Genre popularity through the years. 29
4.6 Count of good movies by country they were filmed in. 33
4.7 Digital vs Theatrical releases. 36
4.8 Evolution of lap times in Silverstone 45
4.9 Evolution of pit stop times in Monza 46
4.10 Shift of teams' dominance over the years 51

List of Tables

4.1 Average Rating, Duration, and Number of Movies by Year 21
4.2 Most consistent directors 23
4.3 Genre average ratings 24
4.4 Top Studios through years by movie count 26
4.5 Genre popularity through the years 27
4.6 Actors and their ratings with count of ratings 30
4.7 Count of good movies by country (truncated to include only some countries in 2023) 32
4.8 Actors, their genres, and number of appearances 34
4.9 Actor-Director pairings 35
4.10 Release Year, Release Type, and Total Movies 36
4.11 Top 10 drivers in terms of race wins 38
4.12 Count of drivers by country 40
4.13 Count of constructors by country 40
4.14 Ratio of races won from pole position 42
4.15 Count of wins based on the starting position 43
4.16 Percentage of duels with teammate won 49
4.17 Times both championships were won by the same team 54
4.18 Times both championships were won by the same team 54
4.19 Most important drivers for Ferrari 56