Hybrid Analytics of Formula 1 Driver-Mechanic Radio Messages

A Data Integration and Event Integrity Approach

Efstratios Demertzoglou | TH20580

Introduction & Objective

F1 data is rich and diverse, combining structured race events with unstructured real time team communications

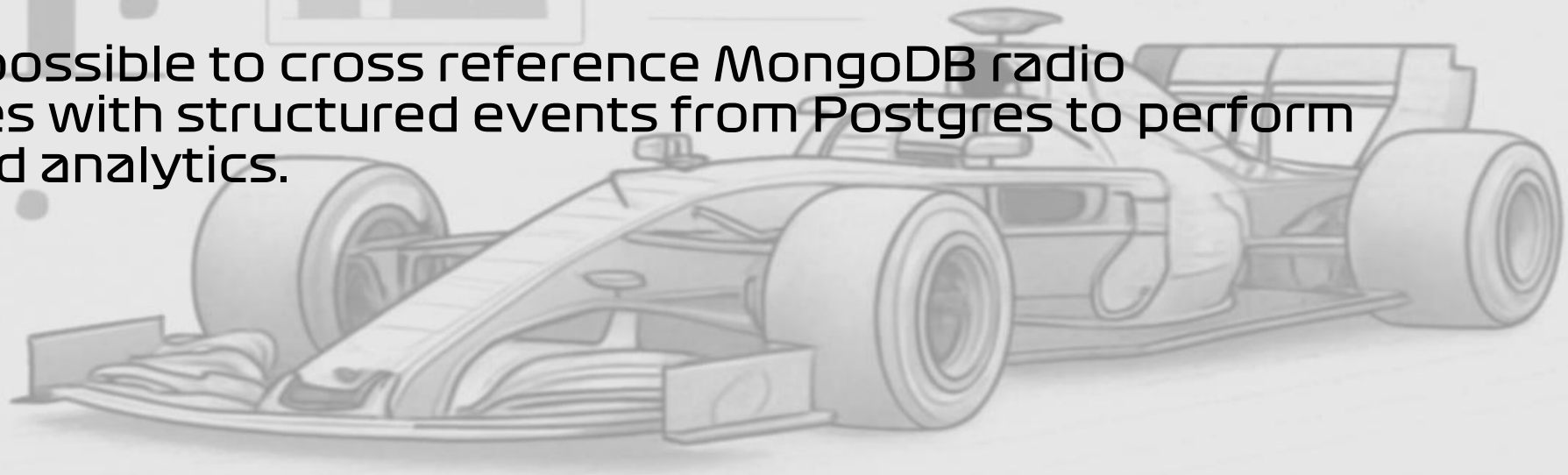
Originally, our F1 data was stored completely in relational databases like PostgreSQL since it was mostly focused on structured data (race results, pit stops, driver info etc)

Therefore this Project's objective is to to bridge the gap by Integrating structured data (PostgreSQL) with unstructured radio messages (MongoDB)

Why choose Mongo?

- Radio messages vary in length, structure and tagging which is very feasible to implement with document type data.

- It is still possible to cross reference MongoDB radio messages with structured events from Postgres to perform advanced analytics.



Data Sources

Structured Data (PostgreSQL):
Contains race events, results, pit stops, teams and circuits organized in relation tables.
Reliable, consistent and robust

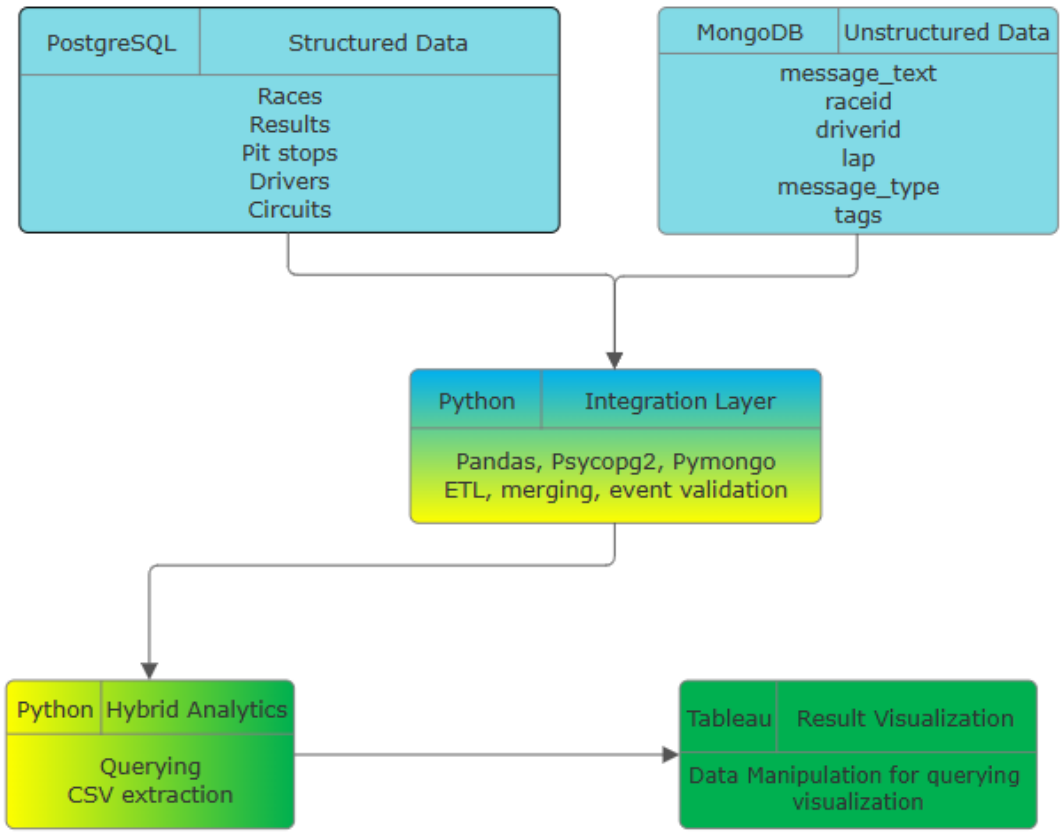
Unstructured Data (MongoDB):
Stores Radio message transcripts including message text , type ,tags and context fields.
Handles variability in message format , flexible tagging and future-proof design

Integration

Approach:

Relational keys like raceid and driverid, link messages to structured race events

For table merging the data was inserted in pandas dataframes and then merged on the respective common keys



Data Processing & Integration

Postgres Part:

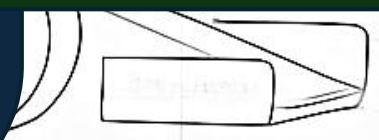
Connecting to Postegres server and creating a pandas dataframe to load the tables' data

```
engine = create_engine('postgresql://postgres:1234@localhost:5432/F1_Analysis')

# List of CSVs and table names
csv_files = {
    "drivers": "C:/F1_Project/data/postgres/drivers.csv",
    "races": "C:/F1_Project/data/postgres/races.csv",
    "results": "C:/F1_Project/data/postgres/results.csv",
    "constructors": "C:/F1_Project/data/postgres/constructors.csv",
    "pit_stops": "C:/F1_Project/data/postgres/pit_stops.csv",
    "lap_times": "C:/F1_Project/data/postgres/lap_times.csv",
    "status": "C:/F1_Project/data/postgres/status.csv",
    "circuits": "C:/F1_Project/data/postgres/circuits.csv"
    # Add more as needed
}

for table, path in csv_files.items():
    df = pd.read_csv(path)
    df.to_sql(table, engine, if_exists='replace', index=False)  # Creates table
if not exists
    print(f"Loaded {table} from {path}")
```

Mongo Part:

Connecting to Mongo server and inserting the data from the json file

```
with open('C:/F1_Project/data/mongo/data_mongo_driver_radio_messages.json', 'r') as f:
    messages = json.load(f)

client = MongoClient('mongodb://localhost:27017/')
db = client['F1_Message_Context']
collection = db['message_context']

# Insert all messages (optionally, clear collection first)
collection.delete_many({})
collection.insert_many(messages)
```



Json Example for Documenting Messages

```
[
...
  {
    "message_id": "msg001",
    "race_id": 333,
    "driver_id": 101,
    "mechanic_name": "Olivia Scott",
    "timestamp": "2025-06-07T13:00:00Z",
    "lap": 2,
    "message_text": "Rain expected in 10 minutes.",
    "message_type": "weather",
    "tags": [
      "weather",
      "rain"
    ]
  },
...
]
```

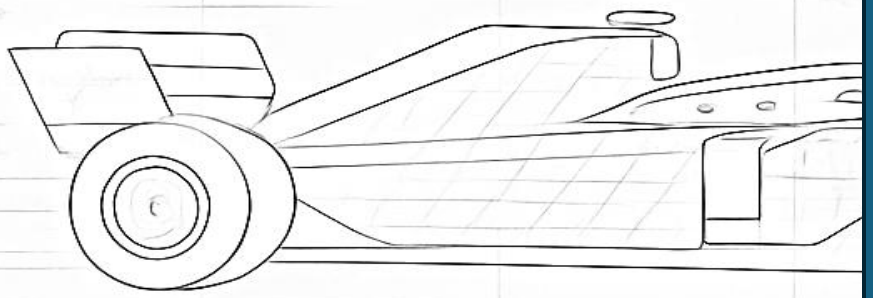
Hybrid Querying

For the following queries the referenced connections will be assumed for space management considerations

```
# Connect to Postgres
conn = psycopg2.connect(dbname='F1_Analysis', user='postgres', password='####', host='localhost')

# Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")
collection = client["F1_Message_Context"]["message_context"]
```

Query 1 | “Box” Messages Matching Real Pit Stops

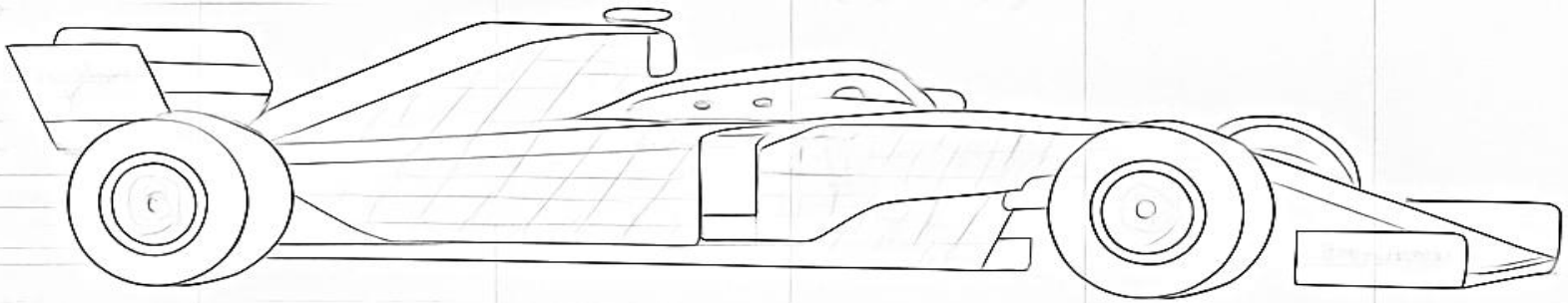
Find all radio messages about pit stops (“box”, “pit for”, etc.) that correspond to an actual pit stop event.



```
pit_stops = pd.read_sql('SELECT "raceId", "driverId", lap FROM pit_stops', conn)
pit_laps = set((row['raceId'], row['driverId'], row['lap']) for _, row in pit_stops.iterrows())

# Find all pitstop-related messages in MongoDB
box_msgs = collection.find({
    "$or": [
        {"message_text": {"$regex": "box", "$options": "i"}},
        {"message_text": {"$regex": "pit for", "$options": "i"}},
        {"tags": "pitstop"}
    ]
})

# Filter messages that match a real pit stop
real_box_msgs = []
for msg in box_msgs:
    key = (msg['race_id'], msg['driver_id'], msg['lap'])
    if key in pit_laps:
        real_box_msgs.append(msg)

print(f"Found {len(real_box_msgs)} messages that match real pit stops.")

# Export results to CSV
if real_box_msgs:
    df = pd.DataFrame(real_box_msgs)
    df.to_csv('hQ1.csv', index=False)
    print("Exported to hQ1.csv!")
else:
    print("No matching messages found.")
```

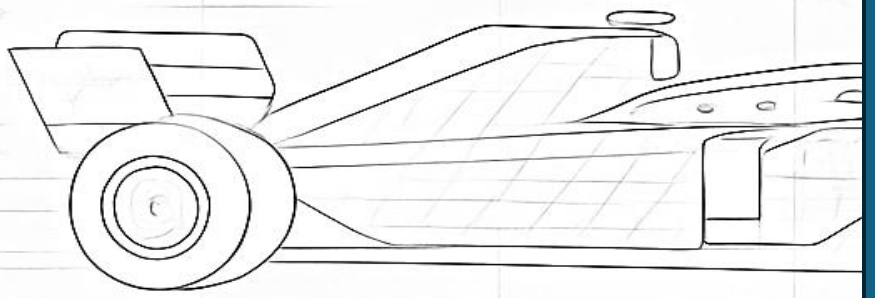
Query 1 | “Box” Messages Matching Real Pit Stops [12 rows (printing 10)]

```
_id,message_id,race_id,driver_id,mechanic_name,timestamp,lap,message_text,message_type,tags
68446db7621bbd729b514236,msg597,1111,857,Emily Adams,2025-06-08T08:52:00Z,63,"Switch to wets, rain intensifying.",weather,"['pitstop', 'wets', 'weather']"
68446db7621bbd729b514275,msg660,911,154,Ella Martin,2025-06-08T10:58:00Z,13,"Box for softs, push now.",strategy,"['pitstop', 'softs']"
68446db7621bbd729b514277,msg662,929,3,Olivia Scott,2025-06-08T11:02:00Z,34,Box now for mediums.,strategy,"['pitstop', 'mediums', 'first_stop']"
68446db7621bbd729b5146fb,msg1818,912,8,Daniel Lee,2025-06-10T01:34:00Z,20,"Box for softs, push now.",strategy,"['pitstop', 'softs']"
68446db7621bbd729b514964,msg2435,999,843,Mike Brown,2025-06-10T22:08:00Z,28,Box this lap for softs.,strategy,"['pitstop', 'softs', 'first_stop']"
68446db7621bbd729b514a87,msg2726,967,825,Sophie King,2025-06-11T07:50:00Z,41,Box for hards.,strategy,"['pitstop', 'hards', 'first_stop']"
68446db7621bbd729b514b98,msg2999,1074,807,John Smith,2025-06-11T16:56:00Z,37,"Pit for softs, final stint.",strategy,"['pitstop', 'softs', 'final_stint']"
68446db7621bbd729b514bf1,msg3088,910,16,John Smith,2025-06-11T19:54:00Z,8,Box now for mediums.,strategy,"['pitstop', 'mediums', 'first_stop']"
68446db7621bbd729b5151a7,msg4550,883,154,Benjamin Allen,2025-06-13T20:38:00Z,42,Box for hards.,strategy,"['pitstop', 'hards', 'first_stop']"
```



Query 2 | Messages for Engine-Failure Retirements

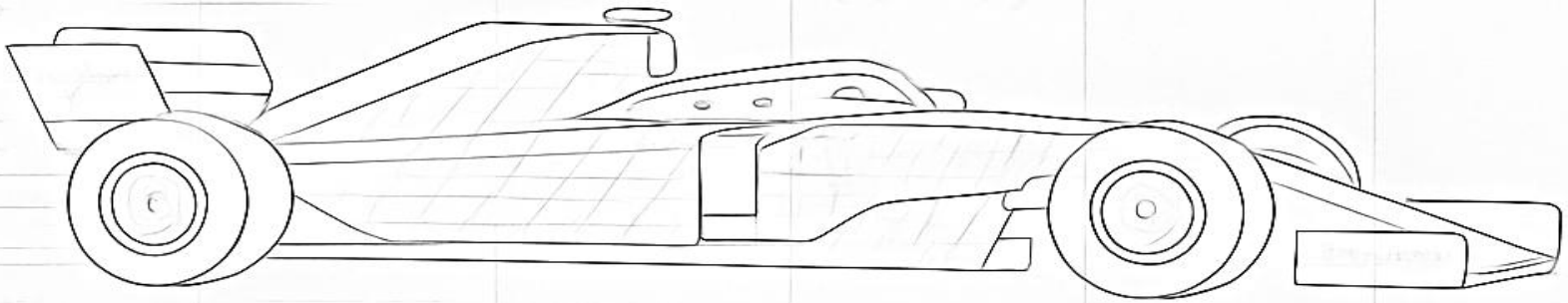
Get all radio messages for drivers who retired due to engine failure.



```
status = pd.read_sql('SELECT "statusId", LOWER(status) as status FROM status', pg_conn)
results = pd.read_sql('SELECT "raceId", "driverId", "statusId" FROM results', pg_conn)

# Identify engine failure statusIds
engine_status_ids =
status[status['status'].str.contains('engine')]['statusId'].astype(str).tolist()
engine_failures = set(
    tuple(x)
    for x in results[results['statusId'].astype(str).isin(engine_status_ids)][['raceId',
'driverId']].values
)

engine_failure_msgs = []
for (race_id, driver_id) in engine_failures:
    msgs = collection.find({"race_id": int(race_id), "driver_id": int(driver_id)})
    for msg in msgs:
        engine_failure_msgs.append(msg)

# Export results to CSV
if engine_failure_msgs:
    df = pd.DataFrame(engine_failure_msgs)
    df.to_csv('hQ2.csv', index=False)
    print("Exported to hQ2.csv!")
else:
    print("No matching messages found.")
```

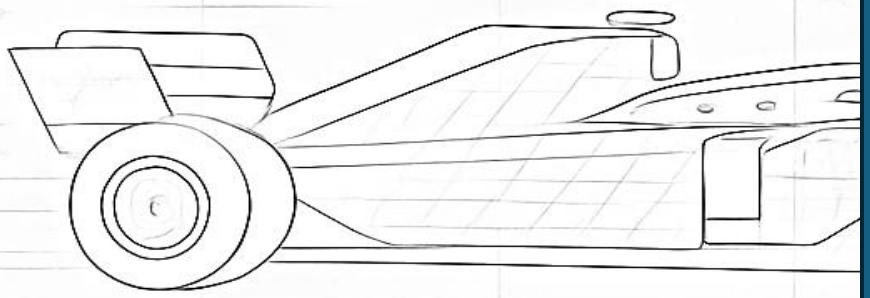
Query 2 | Messages for Engine-Failure Retirements [354 rows(printing 10)]

```
_id,message_id,race_id,driver_id,mechanic_name,timestamp,lap,message_text,message_type,tags
68446db7621bbd729b514f55,msg3956,647,346,Emily Adams,2025-06-13T00:50:00Z,32,Yellow flag in sector 2.,alert,"['yellow_flag', 'sector2']"
68446db7621bbd729b5143b6,msg981,540,219,Chloe Harris,2025-06-08T21:40:00Z,51,DRS enabled.,info,"['drs', 'info']"
68446db7621bbd729b514ccf,msg3310,540,219,Sophie King,2025-06-12T03:18:00Z,46,Rain expected in 10 minutes.,weather,"['weather', 'rain']"
68446db7621bbd729b514f66,msg3973,485,176,Lisa Turner,2025-06-13T01:24:00Z,7,"Front wing damage, box for repairs.",alert,"['damage', 'pitstop', 'repair']"
68446db7621bbd729b514dcc,msg3563,196,14,Benjamin Allen,2025-06-12T11:44:00Z,2,"Retire the car, loss of power.",retirement,"['retirement', 'engine']"
68446db7621bbd729b51517c,msg4507,196,14,Benjamin Allen,2025-06-13T19:12:00Z,12,"Box for softs, push now.",strategy,"['pitstop', 'softs']"
68446db7621bbd729b5146ab,msg1738,633,280,Benjamin Allen,2025-06-09T22:54:00Z,22,"Check tyre temps, tyres are cold.",info,"['tyre_management', 'cold']"
68446db7621bbd729b51470a,msg1833,633,280,Grace Clark,2025-06-10T02:04:00Z,21,"Box for softs, push now.",strategy,"['pitstop', 'softs']"
68446db7621bbd729b514973,msg2450,155,18,Olivia Scott,2025-06-10T22:38:00Z,2,Virtual safety car deployed.,alert,"['vsc', 'alert']"
```



Query 3 - Messages After Pit Stops

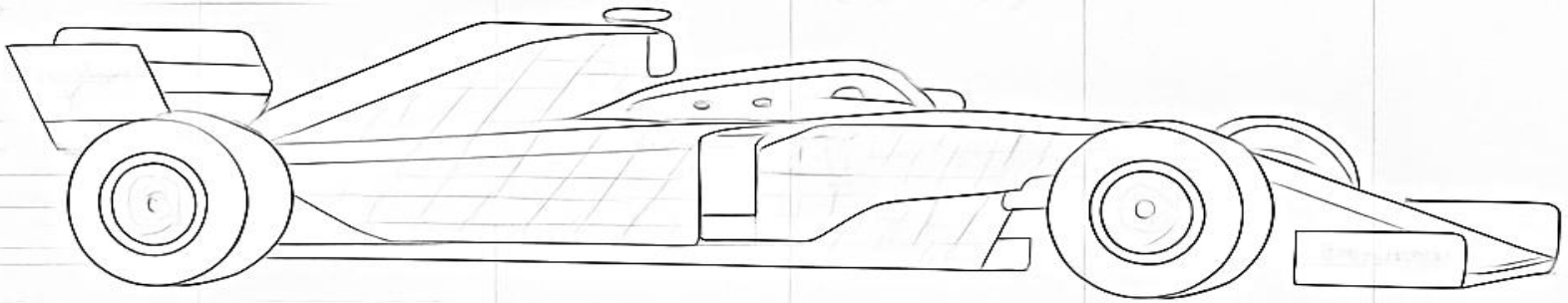
For each pit stop, get all radio messages for the next two laps for that driver/race.



```
pit_stops = pd.read_sql('SELECT "raceId", "driverId", lap FROM pit_stops',
pg_conn)


post_pit_msgs = []
for _, row in pit_stops.iterrows():
    race_id = int(row['raceId'])
    driver_id = int(row['driverId'])
    pit_lap = int(row['lap'])
    for lap in [pit_lap + 1, pit_lap + 2]:
        msgs = collection.find({"race_id": race_id, "driver_id": driver_id,
"lap": lap})
        for msg in msgs:
            post_pit_msgs.append(msg)

# Export results to CSV
if post_pit_msgs:
    df = pd.DataFrame(post_pit_msgs)
    df.to_csv('hQ3.csv', index=False)
    print("Exported to hQ3.csv!")
else:
    print("No matching messages found.")
```

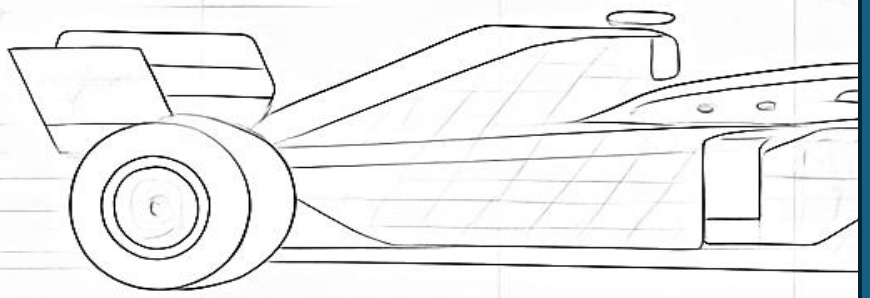
Query 3 | Messages after Pit Stops [74 rows(printing 10)]

```
_id,message_id,race_id,driver_id,mechanic_name,timestamp,lap,message_text,message_type,tags
68446db7621bbd729b514c5e,msg3197,842,67,Ella Martin,2025-06-11T23:32:00Z,33,"Good job, currently P3.",info,"['position', 'info']"
68446db7621bbd729b5140b2,msg209,843,155,John Smith,2025-06-07T19:56:00Z,15,Switch to engine mode 7.,engine,"['engine', 'mode_change']"
68446db7621bbd729b514aca,msg2793,845,13,Sophie King,2025-06-11T10:04:00Z,12,"Red flag, enter the pit lane.",alert,"['red_flag', 'alert']"
68446db7621bbd729b514c60,msg3199,848,67,Ryan Hall,2025-06-11T23:36:00Z,28,"Pit for softs, final stint.",strategy,"['pitstop', 'softs', 'final_stint']"
68446db7621bbd729b514a15,msg2612,851,16,Lisa Turner,2025-06-11T04:02:00Z,13,"Pit for softs, final stint.",strategy,"['pitstop', 'softs', 'final_stint']"
68446db7621bbd729b514a23,msg2626,857,13,Grace Clark,2025-06-11T04:30:00Z,31,"Harvest energy, charge battery.",engine,"['harvest', 'battery']"
68446db7621bbd729b514883,msg2210,858,1,Sophie King,2025-06-10T14:38:00Z,17,"Red flag, enter the pit lane.",alert,"['red_flag', 'alert']"
68446db7621bbd729b514c44,msg3171,864,4,Benjamin Allen,2025-06-11T22:40:00Z,12,"Hold position, conserve tyres.",strategy,"['tyre_management', 'conserve']"
68446db7621bbd729b515282,msg4769,865,155,Emma Young,2025-06-14T03:56:00Z,3,"Brake temps high, adjust pace.",engine,"['brakes', 'high_temp']"
```



Query 4 - Final Lap Messages & Positions

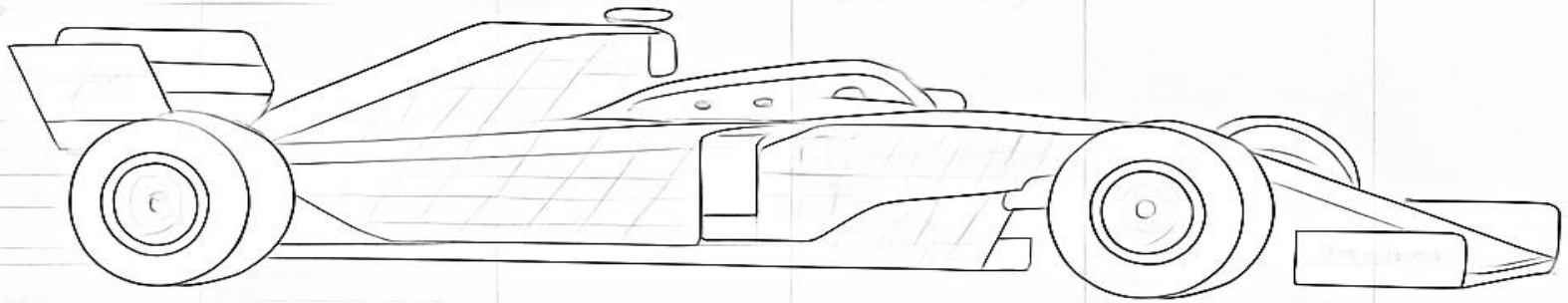
For every “final lap” radio message, get the driver’s position on that lap.



```
lap_times = pd.read_sql('SELECT "raceId", "driverId", lap, position FROM lap_times', pg_conn)
lap_times_lookup = {(row['raceId'], row['driverId'], row['lap']): row['position'] for _, row in lap_times.iterrows()}

final_lap_msgs = collection.find({
    "$or": [
        {"message_text": {"$regex": "final lap", "$options": "i"}},
        {"tags": "final_lap"}
    ]
})
results = []
for msg in final_lap_msgs:
    key = (msg['race_id'], msg['driver_id'], msg['lap'])
    position = lap_times_lookup.get(key, None)
    results.append({
        'race_id': msg['race_id'],
        'driver_id': msg['driver_id'],
        'lap': msg['lap'],
        'message_text': msg['message_text'],
        'position': position
    })

print(f"Found {len(results)} final lap messages with position info.")

if results:
    df = pd.DataFrame(results)
    df.to_csv('hQ4.csv', index=False)
    print("Exported to hQ4.csv!")
else:
    print("No matching messages found.")
```

Query 4 | Final Lap Messages and Positions [211 rows(printing 10)]

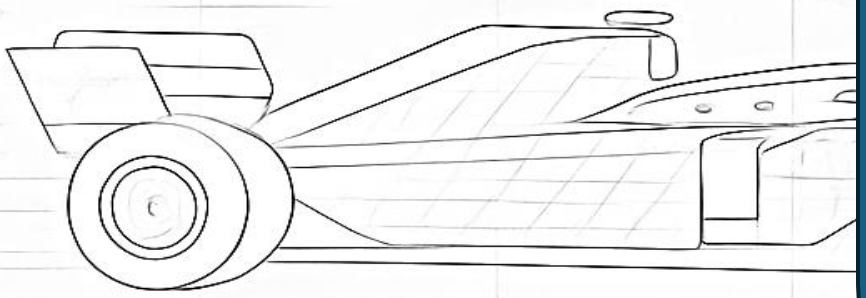
```
race_id,driver_id,lap,message_text,position
941,20,36,"Final lap, bring it home.",2.0
607,332,5,"Final lap, bring it home.",
623,314,19,"Final lap, bring it home.",
939,826,38,"Final lap, bring it home.",6.0
444,183,23,"Final lap, bring it home.",
747,341,46,"Final lap, bring it home.",
1105,822,17,"Final lap, bring it home.",17.0
980,815,4,"Final lap, bring it home.",10.0
902,821,14,"Final lap, bring it home.",16.0
```



Query 5

Most Frequent Message Types by Team

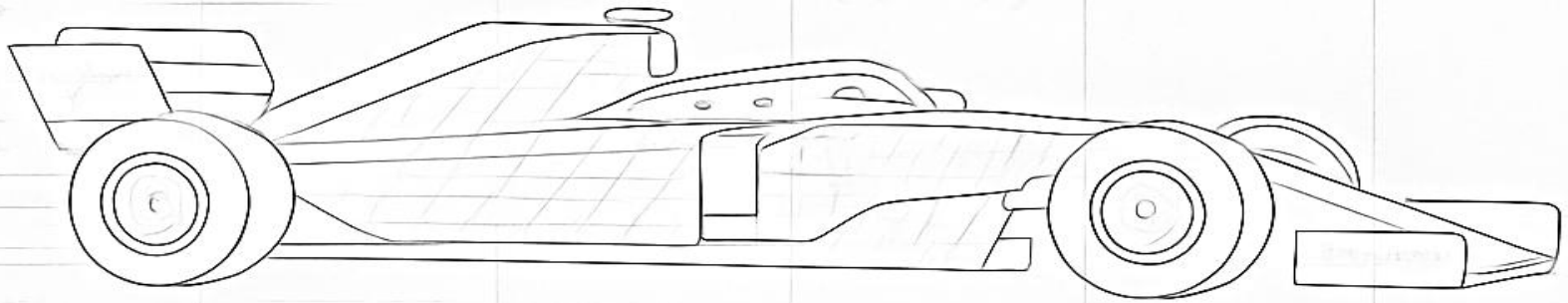
For each team, count how many messages of each type were sent.



```
pg_conn = psycopg2.connect(
    dbname='F1_Analysis', user='postgres', password='1234', host='localhost'
)
results = pd.read_sql('SELECT "raceId", "driverId", "constructorId" FROM results', pg_conn)
constructors = pd.read_sql('SELECT "constructorId", name FROM constructors', pg_conn)

driver_to_constructor = {(row['raceId'], row['driverId']): row['constructorId'] for _, row in results.iterrows()}
constructor_names = {row['constructorId']: row['name'] for _, row in constructors.iterrows()}

client = MongoClient("mongodb://localhost:27017/")
collection = client["F1_Message_Context"]["message_context"]

team_msg_types = defaultdict(lambda: defaultdict(int))

all_msgs = collection.find({})
for msg in all_msgs:
    key = (msg['race_id'], msg['driver_id'])
    constructor_id = driver_to_constructor.get(key)
    if constructor_id:
        team = constructor_names.get(constructor_id, f"constructor_{constructor_id}")
        team_msg_types[team][msg['message_type']] += 1

rows = []
for team, msg_types in team_msg_types.items():
    for msg_type, count in msg_types.items():
        rows.append({
            'team': team,
            'message_type': msg_type,
            'count': count
        })

df = pd.DataFrame(rows)
df.to_csv('hQ5.csv', index=False)
print("Exported to hQ5.csv!")
```

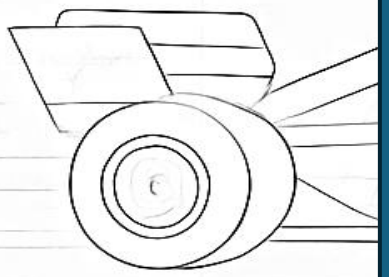
Query 5 | Most Frequent Message Type By Team [621 rows(printing 10)]

```
team,message_type,count
Brabham,weather,13
Brabham,penalty,5
Brabham,strategy,31
Brabham,alert,36
Brabham,engine,34
Brabham,info,20
Lotus-Ford,engine,3
Lotus-Ford,strategy,9
Lotus-Ford,alert,5
```



Query 7 - Messages & Top Team Per Circuit

For every circuit, count total messages and which team sent the most.

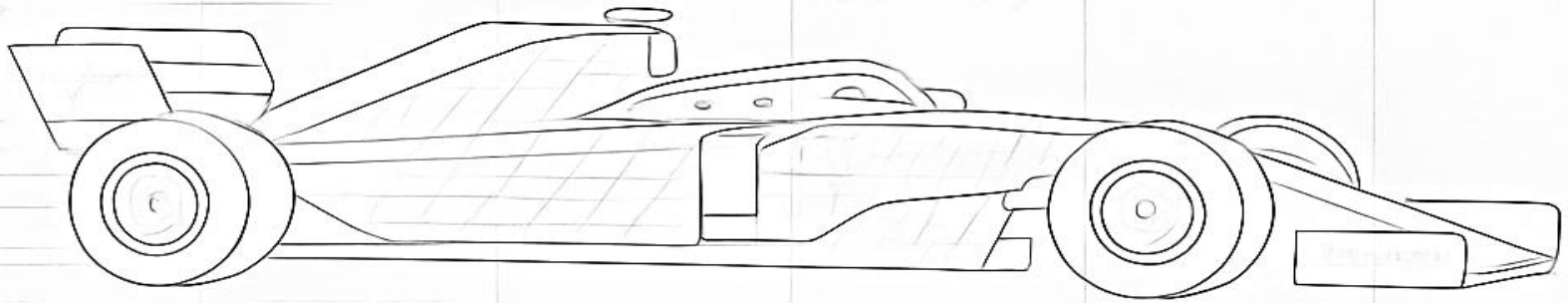


```
# Get raceId, circuitId for all races
races = pd.read_sql('SELECT "raceId", "circuitId" FROM races', pg_conn)

# Get driverId, constructorId, raceId for all results
results = pd.read_sql('SELECT "raceId", "driverId", "constructorId" FROM results',
pg_conn)

# Get constructorId, name mapping
constructors = pd.read_sql('SELECT "constructorId", "name" FROM constructors',
pg_conn)
constructor_names = {row['constructorId']: row['name'] for _, row in
constructors.iterrows()}

# Get circuitId, name mapping
circuits = pd.read_sql('SELECT "circuitId", "name" FROM circuits', pg_conn)
circuit_names = {row['circuitId']: row['name'] for _, row in circuits.iterrows()}

# Build a mapping: (raceId, driverId) -> team name
race_driver_to_team = {(row['raceId'], row['driverId']):
constructor_names.get(row['constructorId'], f"constructor_{row['constructorId']}")
for _, row in results.iterrows()}

# Map raceId -> circuitId
race_to_circuit = {row['raceId']: row['circuitId'] for _, row in races.iterrows()}

# Connect to MongoDB and get all messages
all_msgs = collection.find({})
```

```
# For each circuit, keep a list of all teams' message counts
circuit_total_msgs = defaultdict(int)
circuit_team_counts = defaultdict(lambda: Counter())

# Process each message
for msg in all_msgs:
    race_id = msg.get('race_id')
    driver_id = msg.get('driver_id')
    if race_id not in race_to_circuit:
        continue
    circuit_id = race_to_circuit[race_id]
    team = race_driver_to_team.get((race_id, driver_id), "Unknown")
    circuit_total_msgs[circuit_id] += 1
    circuit_team_counts[circuit_id][team] += 1

# Prepare results
rows = []
for circuit_id, total_msgs in circuit_total_msgs.items():
    team_counts = circuit_team_counts[circuit_id]
    if team_counts:
        top_team, top_count = team_counts.most_common(1)[0]
    else:
        top_team, top_count = "None", 0
    circuit_name = circuit_names.get(circuit_id, "Unknown")
    rows.append({
        "circuitId": circuit_id,
        "circuit_name": circuit_name,
        "total_msgs": total_msgs,
        "top_team": top_team,
        "top_team_msgs": top_count
    })

df = pd.DataFrame(rows)
df.to_csv("circuit_message_stats_with_name.csv", index=False)
print("Exported to circuit_message_stats_with_name.csv!")
```

Query 7 | Messages & Top Team per Circuit [77 rows(printing 10)]

```
circuitId,circuit_name,total_msgs,top_team,top_team_msgs
27,Autódromo do Estoril,73,Team Lotus,12
14,Autodromo Nazionale di Monza,359,Ferrari,44
20,Nürburgring,177,Ferrari,18
13,Circuit de Spa-Francorchamps,226,Ferrari,20
9,Silverstone Circuit,298,Ferrari,33
39,Circuit Park Zandvoort,152,Ferrari,23
69,Circuit of the Americas,35,Mercedes,5
46,Watkins Glen,82,Lotus-Climax,8
32,Autódromo Hermanos Rodríguez,107,Ferrari,11
```



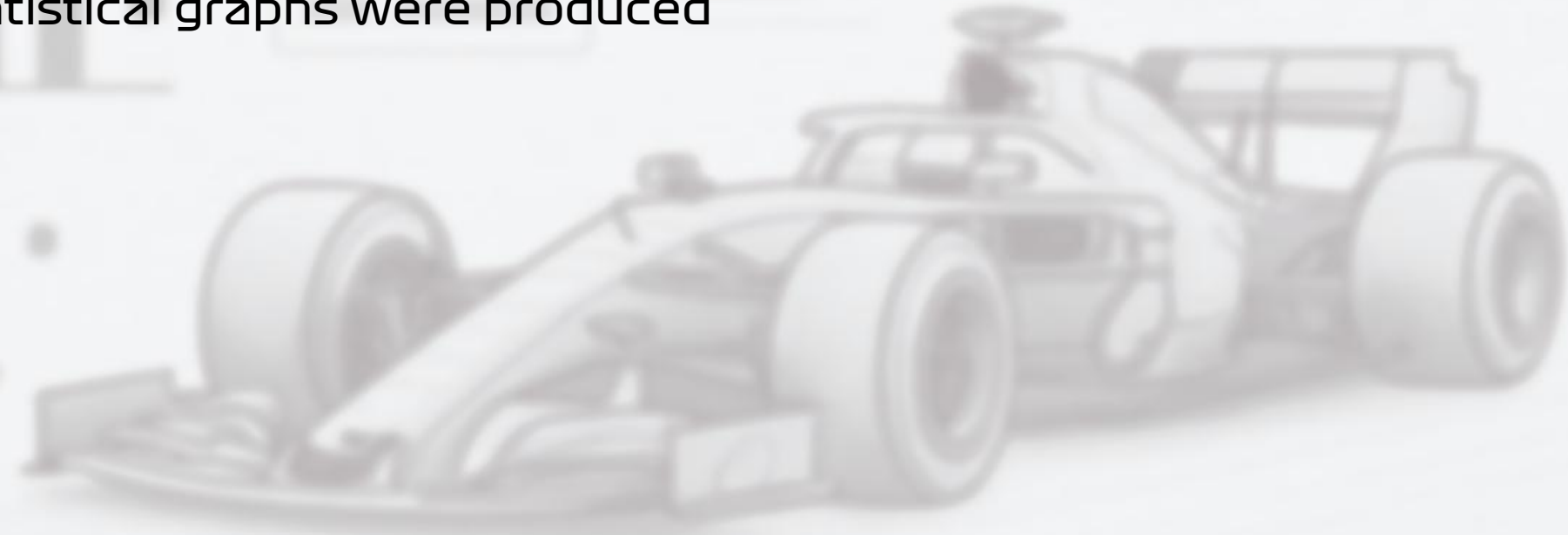
Summarized Querying resutls

Query	Analytical Focus	Key Result / Statistic
Q1	Pit stop message validation	X% message-event match
Q2	Engine failure retirements	Engine fail messages pre-retirement
Q3	Post-pit stop communication	Y avg. messages post-pit stop
Q4	Final lap & position linkage	Message content varies by position
Q5	Message types per team	High comm. teams = high performance
Q6/7	Circuit/team communication trends	Monza: most messages, Mercedes top

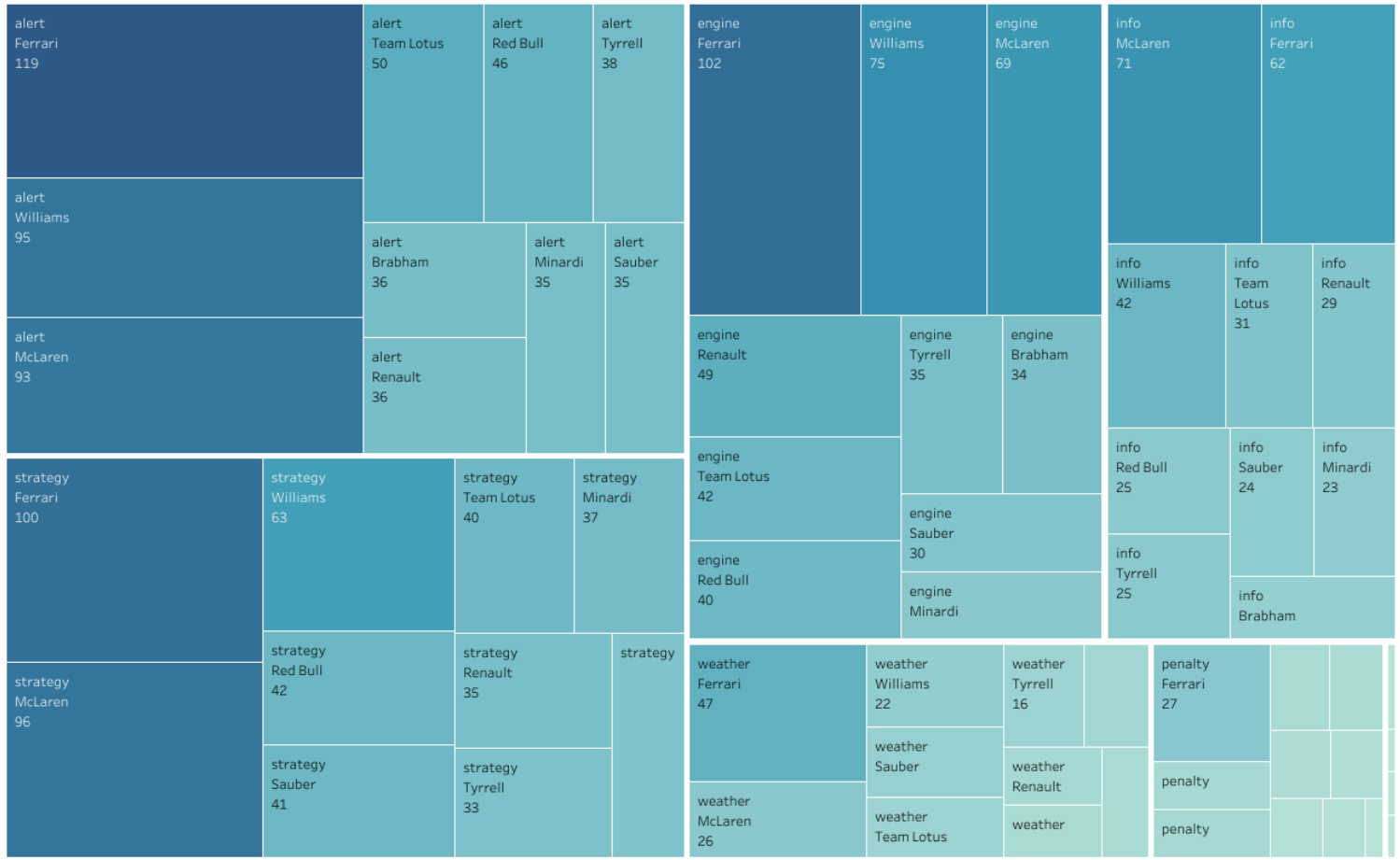
Analytics with Tableau

The result from the 5th and 7th hybrid Query were exported into csv files and inserted into Tableau for visualization purposes.

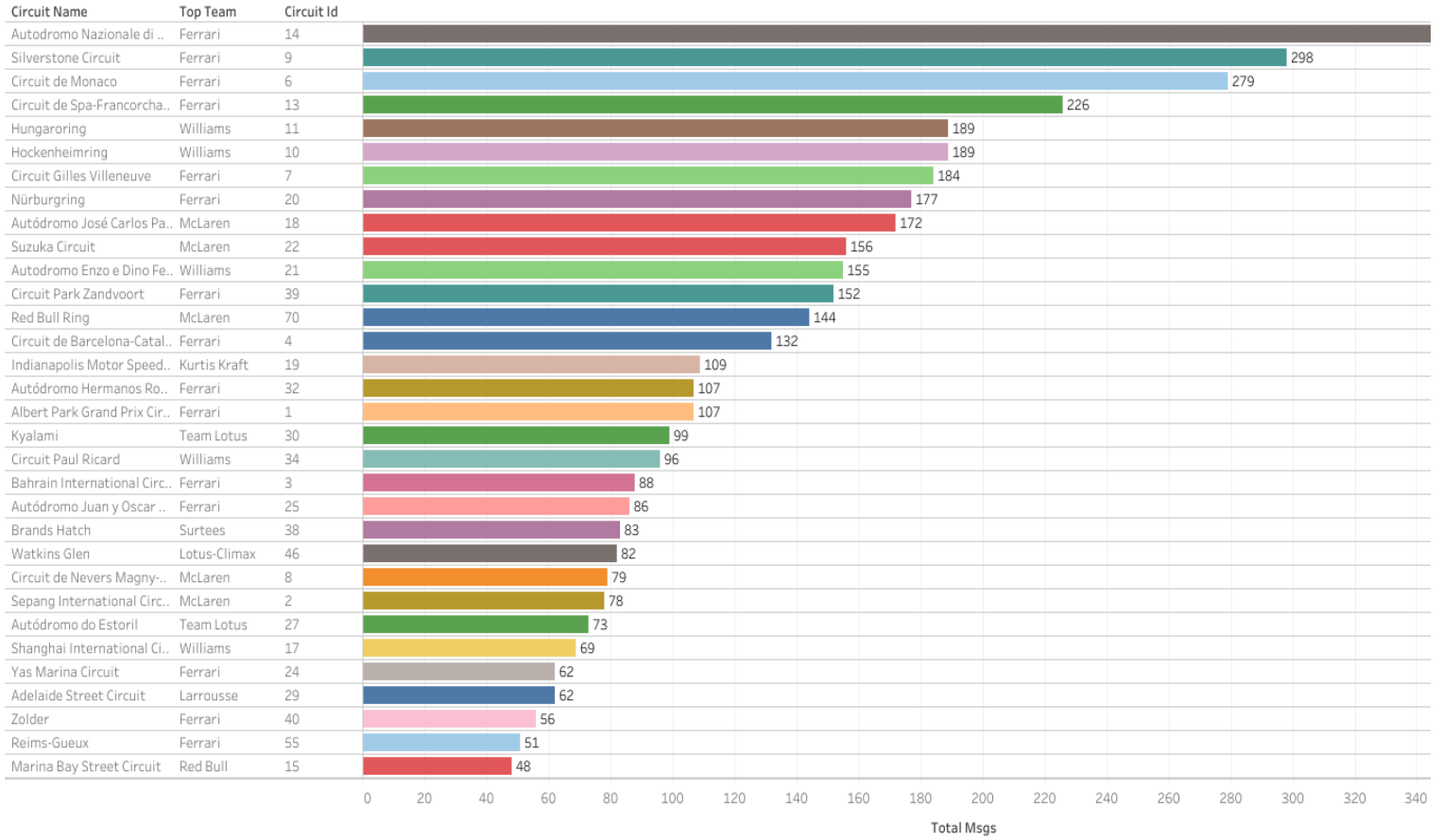
Different Attributes were distributed among rows and columns in a diagram the following statistical graphs were produced



Query 5 - Visualization



Query 7 - Visualization



Discussion & Interpretation

The showcased work demonstrated seam integration of both structured and unstructured forms of data to enable hybrid analytics.

Every radio Message analyzed was randomly generated and validated against actual race events according to the Kaggle F1 dataset so the results can be as technically accurate as possible.

It has to be noted that 5000 radio message samples were generated for academic purposes and in reality the working samples would have been millions.

None the less, a remarkable effort was made to ensure data validity and robust querying in an academic and educational context.

Data generation Algorithm for documentation purposes

```
results =
pd.read_csv('C:/F1_Project/data/postgres/results.csv')
lap_times =
pd.read_csv('C:/F1_Project/data/postgres/lap_times.csv')
pit_stops =
pd.read_csv('C:/F1_Project/data/postgres/pit_stops.csv')
status =
pd.read_csv('C:/F1_Project/data/postgres/status.csv')

# Build valid (raceId, driverId) pairs
valid_pairs = results[['raceId',
'driverId']].drop_duplicates().values.tolist()

# Map each (raceId, driverId) to valid laps
lap_map = lap_times.groupby(['raceId',
'driverId'])['lap'].unique().to_dict()

# Build pit stop lookup: (raceId, driverId) -> set of pit
stop laps
pit_lap_map = pit_stops.groupby(['raceId',
'driverId'])['lap'].unique().to_dict()

# Build engine failure lookup: (raceId, driverId) where
status is 'Engine'
engine_status_ids =
status[status['status'].str.lower().str.contains('engine')][
'statusId'].astype(str).tolist()
engine_failures = set(
    tuple(x) for x in
results[results['statusId'].astype(str).isin(engine_status_i
ds)][['raceId', 'driverId']].values
)
```

```
start_time = datetime(2025, 6, 7, 13, 0, 0)
messages = []
message_count = 5000  # or whatever you want

for i in range(message_count):
    race_id, driver_id = random.choice(valid_pairs)
    laps = lap_map.get((race_id, driver_id), list(range(1, 56)))
    lap = int(random.choice(laps))
    mechanic = random.choice(mechanic_names)

    # Find which templates are allowed on this lap
    allowed_templates = []
    for tpl in message_templates:
        text, typ, tags, condition = tpl
        pit_laps = set(pit_lap_map.get((race_id, driver_id), []))
        if condition == "pit_now":
            if lap in pit_laps:
                allowed_templates.append(tpl)
        elif condition == "pit_next":
            if (lap+1) in pit_laps:
                allowed_templates.append(tpl)
        elif condition == "engine_fail":
            if (race_id, driver_id) in engine_failures:
                allowed_templates.append(tpl)
        elif condition is None:
            allowed_templates.append(tpl)
```

```
    if not allowed_templates:
        continue  # skip this message if nothing valid to say

    template = random.choice(allowed_templates)
    timestamp = start_time + timedelta(minutes=2*i)
    messages.append({
        "message_id": f"msg{str(len(messages)+1).zfill(3)}",
        "race_id": int(race_id),
        "driver_id": int(driver_id),
        "mechanic_name": mechanic,
        "timestamp": timestamp.isoformat() + "Z",
        "lap": lap,
        "message_text": template[0],
        "message_type": template[1],
        "tags": template[2]
    })

with
open("C:/F1_Project/data/mongo/data_mongo_driver_radio_messag
es.json", "w") as f:
    json.dump(messages, f, indent=2)
```

References

Formula 1 Data:
Kaggle Formula 1 World Championship Data
Databases:
PostgreSQL
Documentation. https://www.postgresql.org/docs/
MongoDB Documentation. https://docs.mongodb.com/
Python Libraries:
pandas Documentation. https://pandas.pydata.org/docs/
psycopg2 Documentation. https://www.psycopg.org/docs/
PyMongo Documentation. https://pymongo.readthedocs.io/
SQLAlchemy Documentation. https://docs.sqlalchemy.org/
Visualization:
Tableau Documentation. https://help.tableau.com/
Project Repository:
www.github.com/StratosDns/F1_PROJECT