Task 2: Becoming a Data Detective

For the SQL exploration I will use duckdb as a SQL engine because of its easy of use and setup.

Importing duckdb and creating views for the given CSV. This makes writing SQL a bit more easier.

In [19]:
```python
# import the duckdb
import duckdb

# Connect to an in-memory DuckDB database
con = duckdb.connect(database=':memory:')

# Create views for the CSV files
con.execute("CREATE VIEW maintenance AS SELECT * FROM '../data/raw/maintenance_events.csv'")
con.execute("CREATE VIEW manufacturing AS SELECT * FROM '../data/raw/manufacturing_factory_dataset.csv'")
con.execute("CREATE VIEW operator AS SELECT * FROM '../data/raw/operators_roster.csv'")
```

Out[19]:
```
<_duckdb.DuckDBPyConnection at 0x7eff3f6a42f0>
```

Before answering the question I performed basic SQL check on the dataset to ensure that the dataset was reliable:

Uniqueness Check

Ensures that event_id is unique and there are no duplicate records for the same event.

```sql
SELECT
    event_id,
    COUNT(*)
FROM maintenance
GROUP BY event_id
HAVING COUNT(*) > 1;
```

NULL Value Checks

Identifies missing data in critical fields that could break your calculations (like downtime_min or cost_eur)

```sql
SELECT
    COUNT(*) - COUNT(event_id) AS missing_ids,
    COUNT(*) - COUNT(start_time) AS missing_start_times,
    COUNT(*) - COUNT(cost_eur) AS missing_costs,
    COUNT(*) - COUNT(technician_id) AS missing_technicians
FROM maintenance_table;
```

Chronological Checks

Checks for time errors where the end time occurs before the start time, or the next maintenance is scheduled in the past.

```sql
SELECT * FROM maintenance
WHERE end_time < start_time
   OR next_due_date < end_time::date;
```

Calculation Consistency

Verify that the downtime_min column actually matches the difference between start_time and end_time.

```sql
SELECT
    event_id,
    start_time,
    end_time,
    downtime_min,
    date_diff('minute', start_time, end_time)  AS calculated_min
FROM maintenance
WHERE downtime_min != date_diff('minute', start_time, end_time)
```

Once satisfied with the reliability of the data I went forward and queried the data to answer the questions.

In [20]:
```python
query = """
SELECT
    *
FROM maintenance
"""


df = con.execute(query).df()
print(df.head())
```

```
                event_id    factory_id line_id maintenance_type  \
0  MEV-202511031011-1684  FRA-PLANT-01  Line-B       Inspection
1  MEV-202511031035-9997  FRA-PLANT-01  Line-A       Preventive
2  MEV-202511031207-5834  FRA-PLANT-01  Line-C       Preventive
3  MEV-202511031312-3195  FRA-PLANT-01  Line-B       Preventive
4  MEV-202511031603-1385  FRA-PLANT-01  Line-A       Corrective

                reason          start_time            end_time  downtime_min  \
0  Planned Maintenance 2025-11-03 10:11:00 2025-11-03 10:41:00            30
1  Planned Maintenance 2025-11-03 10:35:00 2025-11-03 10:50:00            15
2  Planned Maintenance 2025-11-03 12:07:00 2025-11-03 13:37:00            90
3  Planned Maintenance 2025-11-03 13:12:00 2025-11-03 14:42:00            90
4  Unplanned Breakdown 2025-11-03 16:03:00 2025-11-03 16:48:00            45

  technician_id         parts_used  cost_eur                outcome  \
0      TECH-010  PART-022,PART-040   2201.88  Root Cause Identified
1      TECH-012  PART-010,PART-038   1719.94               Restored
2      TECH-002               None    683.64               Restored
3      TECH-004  PART-034,PART-006   1439.96      Temporarily Fixed
4      TECH-024  PART-021,PART-040   2615.84               Restored

  next_due_date
0    2025-11-06
1    2025-11-04
2    2025-11-04
3    2025-11-17
4    2025-11-06
```

1. What's the total maintenance cost?

The total cost comes out at **total_maintenance_cost:** 169389.62

All the events in the table are unique, so a simple sum of the *cost_eur* column give me the answer.

In [21]:
```
query = """
SELECT
    SUM(cost_eur) AS total_maintenance_cost
FROM maintenance
"""


df = con.execute(query).df()
print(df)
```

```
   total_maintenance_cost
0               169389.62
```

2. How many minutes of downtime have there been?

Total downtime is **6180 minutes**.

The query in itself is simple, but I first checked the min, max values to make sure there were no negative values, checked if there were NULL rows:

```
SELECT
    COUNT(*) AS total_rows,
    COUNT(downtime_min) AS non_null_rows,
    MIN(downtime_min) AS min_val,
    MAX(downtime_min) AS max_val,
    SUM(downtime_min) AS total_downtime
FROM maintenance
```

I also doubled checked my results by comparing difference b/w the start and end time of maintenance by using the following query:

```
SELECT
    date_diff('minute', start_time, end_time) AS calculated_downtime,
    downtime_min
FROM maintenance
```

Once satisfied with the initial query results I went ahead and calculated the downtime by summing the *downtime_min* column.

In [22]:
```
query = """
SELECT
    SUM(downtime_min) AS total_downtime
FROM maintenance
"""


df = con.execute(query).df()
print(df)
```

```
   total_downtime
0          6180.0
```

3.How many maintenance events occurred?

Total maintenance event occurred: 94

In [23]:
```
query = """
SELECT
    COUNT(*) AS maintenance_event_count
FROM maintenance
"""


df = con.execute(query).df()
print(df)
```

```
   maintenance_event_count
0                       94
```

4. How many breakdowns (unplanned) happened?

Total Unplanned Events: 23

I first checked the values in the *reason* column by running **distinct** on the column. This revealed that there were only two values in the column (Planned Maintenance, Unplanned Breakdown). A simple filter on Unplanned Breakdown and count(*) gave me the unplanned breakdowns.

In [24]:
```
query = """
SELECT
    count(*) AS unplanned_maintenance_events
FROM maintenance
WHERE reason = 'Unplanned Breakdown'
"""


df = con.execute(query).df()
print(df)
```

```
   unplanned_maintenance_events
0                            23
```

5. What’s the average downtime per event?

Average Downtime (min): 65.7 minutes

Considering that there were no NULL events in the dataset the AVG downtime was calculated via AVG func.

In [25]:
```
query = """
SELECT
    count(*) AS planned_maintenance_events,
    AVG(downtime_min) AS average_downtime
FROM maintenance
where downtime_min > 0
"""


df = con.execute(query).df()
print(df)
```

```
   planned_maintenance_events  average_downtime
0                          94         65.744681
```