STRAVA FITNESS DATA ANALYTICS CASE-STUDY

SQLite analytics

dailyActivity_merge.csv

Use the dailyActivity_merge.csv

```sql
--  STEP 1: Cleaning & Preprocessing in SQLite
-- ✓ 1.1. Check for NULL or Missing Values
SELECT
  SUM(CASE WHEN Id IS NULL THEN 1 ELSE 0 END) AS NullId,
  SUM(CASE WHEN ActivityDate IS NULL THEN 1 ELSE 0 END) AS NullActivityDate,
  SUM(CASE WHEN TotalSteps IS NULL THEN 1 ELSE 0 END) AS NullTotalSteps,
  SUM(CASE WHEN Calories IS NULL THEN 1 ELSE 0 END) AS NullCalories
FROM dailyActivity_merged;
```

	NullId	NullActivityDate	NullTotalSteps	NullCalories
1	0	0	0	0

```
Execution finished without errors.
Result: 1 rows returned in 14ms
At line 3:
--  STEP 1: Cleaning & Preprocessing in SQLite
-- ✓ 1.1. Check for NULL or Missing Values
SELECT
  SUM(CASE WHEN Id IS NULL THEN 1 ELSE 0 END) AS NullId,
  SUM(CASE WHEN ActivityDate IS NULL THEN 1 ELSE 0 END) AS NullActivityDate,
  SUM(CASE WHEN TotalSteps IS NULL THEN 1 ELSE 0 END) AS NullTotalSteps,
  SUM(CASE WHEN Calories IS NULL THEN 1 ELSE 0 END) AS NullCalories
FROM dailyActivity_merged;
```

```sql
--  1.2. Check for Duplicates
SELECT Id, ActivityDate, COUNT(*) AS cnt
FROM dailyActivity_merged
GROUP BY Id, ActivityDate
HAVING cnt > 1;
```

Id	ActivityDate	cnt

```
Execution finished without errors.
Result: 0 rows returned in 10ms
At line 21:
--  1.2. Check for Duplicates
SELECT Id, ActivityDate, COUNT(*) AS cnt
FROM dailyActivity_merged
GROUP BY Id, ActivityDate
HAVING cnt > 1;
```

```sql
-- 1.3. Check for Outliers or Invalids
-- Negative steps, distance, or calories
SELECT *
FROM dailyActivity_merged
WHERE TotalSteps < 0 OR TotalDistance < 0 OR Calories < 0;
```

Id	ActivityDate	TotalSteps	TotalDistance	TrackerDistance	LoggedActivitiesDistance	VeryActiveDistance	ModeratelyActiveDistance	LightActiveDistance	SedentaryActiveDistance	VeryActiveMinutes	FairlyActi

```
Execution finished without errors.
Result: 0 rows returned in 10ms
At line 28:
-- 1.3. Check for Outliers or Invalids
-- Negative steps, distance, or calories
SELECT *
FROM dailyActivity_merged
WHERE TotalSteps < 0 OR TotalDistance < 0 OR Calories < 0;
```

```sql
-- 1.4. Convert ActivityDate to usable format (if not already)
-- Create a new column with ISO format
ALTER TABLE dailyActivity_merged ADD COLUMN ActivityDateFormatted TEXT;

UPDATE dailyActivity_merged
SET ActivityDateFormatted =
  substr(ActivityDate, 7, 4) || '-' ||
  substr(ActivityDate, 1, 2) || '-' ||
  substr(ActivityDate, 4, 2);
```

Id	ActivityDate	TotalSteps	TotalDistance	TrackerDistance	LoggedActivitiesDistance	VeryActiveDistance	ModeratelyActiveDistance	LightActiveDistance	SedentaryActiveDistance	VeryActiveMinutes	FairlyActi

```
Execution finished without errors.
Result: query executed successfully. Took 5ms, 940 rows affected
At line 40:
UPDATE dailyActivity_merged
SET ActivityDateFormatted =
  substr(ActivityDate, 7, 4) || '-' ||
  substr(ActivityDate, 1, 2) || '-' ||
  substr(ActivityDate, 4, 2);
```

Step 2: Insights and analytics Quesries

```sql
--  STEP 2: Insights and Analytics Queries
--   2.1. Average Steps, Calories, Distance
SELECT
  ROUND(AVG(TotalSteps), 2) AS AvgSteps,
  ROUND(AVG(TotalDistance), 2) AS AvgDistance,
  ROUND(AVG(Calories), 2) AS AvgCalories
FROM dailyActivity_merged;
```

	AvgSteps	AvgDistance	AvgCalories
1	7637.91	5.49	2303.61

```
Execution finished without errors.
```

--🧠 Insight: On average, users take around AvgSteps steps, cover AvgDistance km, and burn approximately AvgCalories calories daily.

```sql
-- 2.2. Users With Highest Average Activity
SELECT Id,
       ROUND(AVG(TotalSteps), 2) AS AvgSteps,
       ROUND(AVG(Calories), 2) AS AvgCalories
FROM dailyActivity_merged
GROUP BY Id
ORDER BY AvgSteps DESC
LIMIT 5;
```

	Id	AvgSteps	AvgCalories
1	8877689391	16040.03	3420.26
2	8053475328	14763.29	2945.81
3	1503960366	12116.74	1816.42
4	2022484408	11370.65	2509.97
5	7007744171	11323.42	2544.0

```
Execution finished without errors.
```

--🧠 Insight: The top 5 most active users averaged the highest number of steps and calorie burn per day, indicating consistent high activity.

```sql
--  2.3. Relationship Between Steps and Calories
SELECT TotalSteps, Calories
FROM dailyActivity_merged
ORDER BY TotalSteps DESC
LIMIT 20;
```

	TotalSteps	Calories
1	36019	2690
2	29326	4547
3	27745	4398
4	23629	3808
5	23186	3921
6	22988	3577
7	22770	4022
8	22359	3554
9	22244	2670
10	22026	3589
11	21727	4015
12	21420	3832
13	21129	3793
14	20669	3410
15	20500	3403

-- 🧠 Insight: As step count increases, calories burned also tend to increase, showing a positive correlation. This can be better visualized through a scatter plot.

```sql
--2.4. Active vs Sedentary Minutes Summary
SELECT
  ROUND(AVG(VeryActiveMinutes), 2) AS AvgVeryActive,
  ROUND(AVG(LightlyActiveMinutes), 2) AS AvgLightlyActive,
  ROUND(AVG(SedentaryMinutes), 2) AS AvgSedentary
FROM dailyActivity_merged;
```

	AvgVeryActive	AvgLightlyActive	AvgSedentary
1	21.16	192.81	991.21

Execution finished without errors.

-- 🧠 Insight: On average, users spend AvgVeryActive minutes being very active, AvgLightlyActive minutes lightly active, and AvgSedentary minutes being sedentary, showing a need for more active movement.

```sql
-- 2.5. Days with Over 10k Steps
SELECT COUNT(*) AS DaysOver10KSteps
FROM dailyActivity_merged
WHERE TotalSteps >= 10000;
```

	DaysOver10KSteps
1	303

Execution finished without errors.

-- 🧠 Insight: The number of days users walked more than 10,000 steps shows how often they meet or exceed the daily recommended activity level.

```sql
-- 2.6. Analyzing Daily Activity Distribution Across Users
WITH user_avg_steps AS (
  SELECT Id, AVG(TotalSteps) AS avg_steps
  FROM dailyActivity_merged
  GROUP BY Id
)
SELECT
  CASE
    WHEN avg_steps < 5000 THEN 'Sedentary (<5k)'
    WHEN avg_steps BETWEEN 5000 AND 9999 THEN 'Moderately Active (5k-10k)'
    ELSE 'Highly Active (10k+)'
  END AS ActivityLevel,
  COUNT(*) AS UserCount
FROM user_avg_steps
GROUP BY ActivityLevel;
```

	ActivityLevel	UserCount
1	Highly Active (10k+)	7
2	Moderately Active (5k-10k)	18
3	Sedentary (<5k)	8

--🧠 Insight: This breakdown reveals the activity levels of users, highlighting how many are sedentary (<5k steps), moderately active (5k–10k), and highly active (>10k).

-- This can help in segmenting users for targeted fitness recommendations or challenges.

```sql
-- 2.7. Calorie Burn Efficiency: Calories per Distance vs Active Minutes
SELECT
  Id,
  ROUND(SUM(Calories)/SUM(TotalDistance), 2) AS CaloriesPerKM,
  ROUND(SUM(Calories)/NULLIF(SUM(VeryActiveMinutes + FairlyActiveMinutes + LightlyActiveMinutes), 0), 2) AS CaloriesPerActiveM
FROM dailyActivity_merged
GROUP BY Id
HAVING CaloriesPerKM IS NOT NULL AND CaloriesPerActiveMinute IS NOT NULL
ORDER BY CaloriesPerKM DESC
LIMIT 10;
```

	Id	CaloriesPerKM	CaloriesPerActiveMinute
1	1927972279	3424.35	53.0
2	8792009665	1653.79	20.0
3	4020332650	1467.17	27.0
4	6775888955	1175.52	32.0
5	1844505072	922.25	13.0
6	4057192912	689.52	18.0
7	4445114986	673.54	10.0
8	6290855005	608.47	11.0
9	2320127002	540.87	8.0
10	5577150313	540.71	12.0

Execution finished without errors.

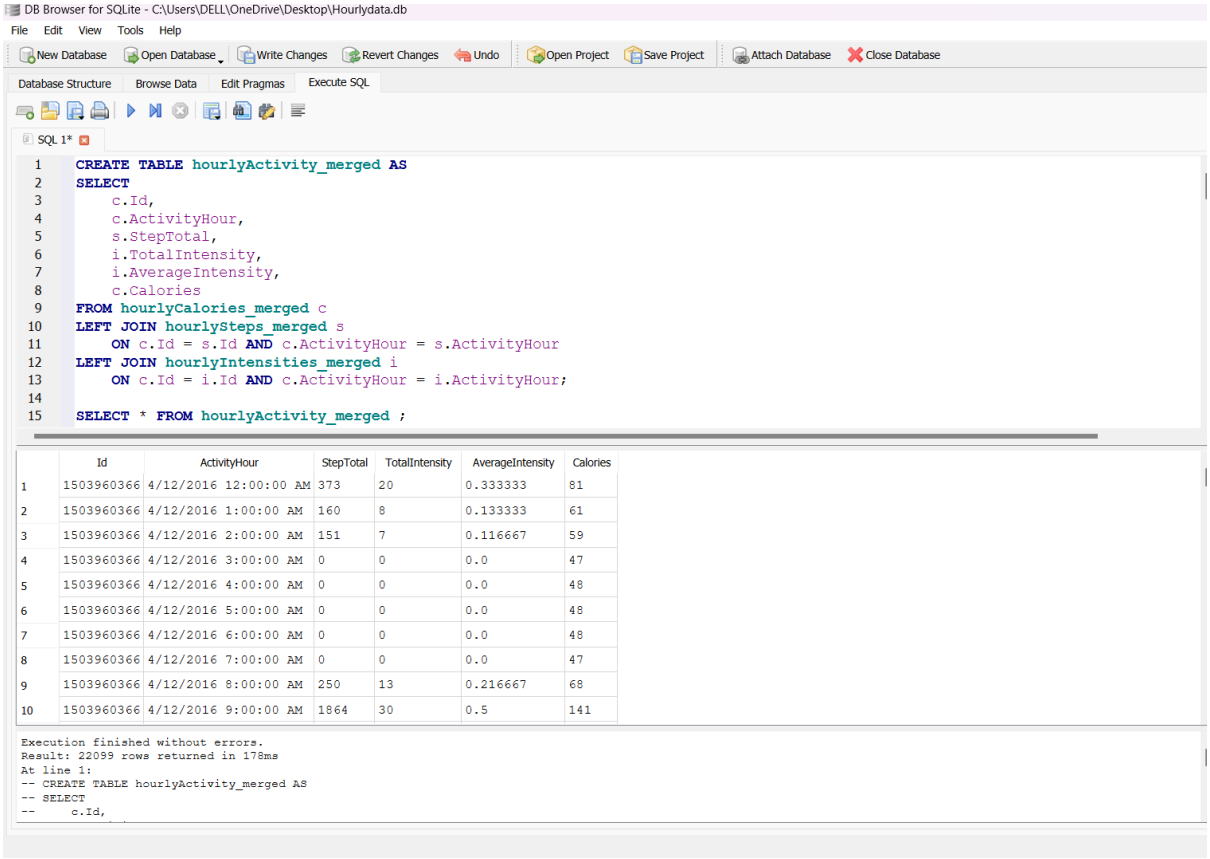
--🧠 Insight: By comparing calories burned per km and per active minute, we can spot users who burn more calories efficiently (either due to intensity or metabolic rate).

-- Users with high calories/km and low active minutes may be engaging in high-effort, short-duration exercises.

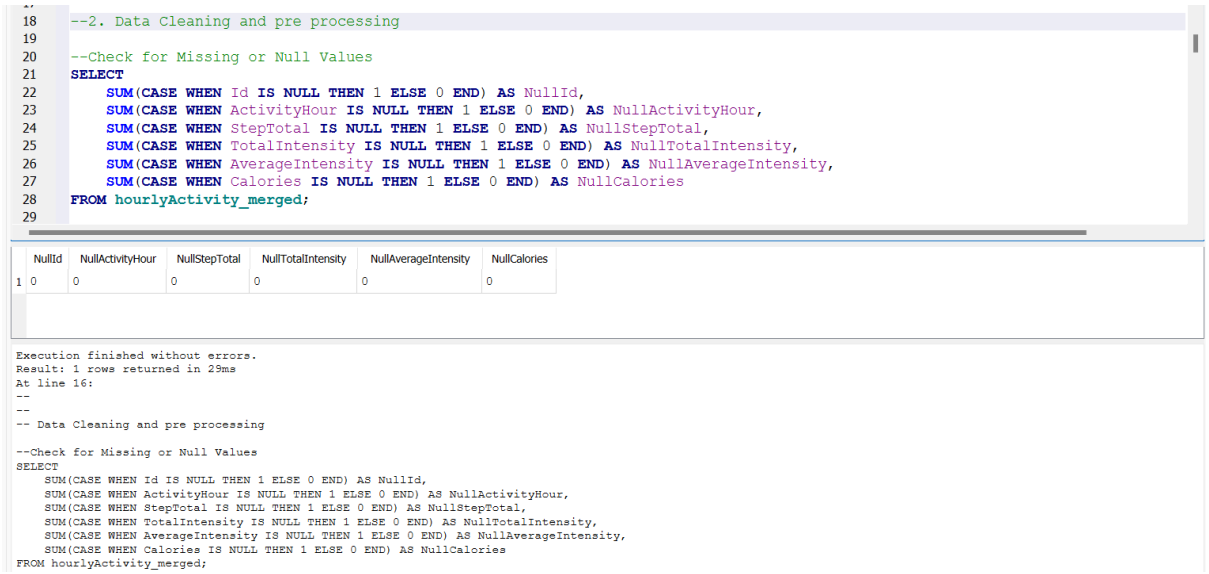
hourlyActivity_merge.csv

Created a merge file name **hourlyActivity_merge.csv** using hourlyCalories_merged.csv hourlyIntensities_merged.csv hourlySteps_merged.csv

1. merged table created



2. Data Cleaning and pre-processing



```sql
-- Check for Duplicates
SELECT
    Id, ActivityHour, COUNT(*) AS Count
FROM hourlyActivity_merged
GROUP BY Id, ActivityHour
HAVING Count > 1;
```

Id	ActivityHour	Count

```
Execution finished without errors.
Result: 0 rows returned in 33ms
At line 33:
-- Check for Duplicates
SELECT
    Id, ActivityHour, COUNT(*) AS Count
FROM hourlyActivity_merged
GROUP BY Id, ActivityHour
HAVING Count > 1;
```

```sql
--Standardize Date Format (Optional, for easier filtering)
SELECT DISTINCT ActivityHour
FROM hourlyActivity_merged
LIMIT 5;

--Steps or Calories Negative?
SELECT *
FROM hourlyActivity_merged
WHERE StepTotal < 0 OR Calories < 0;
```

Id	ActivityHour	StepTotal	TotalIntensity	AverageIntensity	Calories

```
Execution finished without errors.
Result: 0 rows returned in 9ms
At line 45:
--Steps or Calories Negative?
SELECT *
FROM hourlyActivity_merged
WHERE StepTotal < 0 OR Calories < 0;
```

3. Data Analytics & insights

```sql
--3. Data Analysis & insights
-- Total Steps, Calories, and Intensity by User

SELECT
    Id,
    SUM(StepTotal) AS TotalSteps,
    SUM(Calories) AS TotalCalories,
    SUM(TotalIntensity) AS TotalIntensity
FROM hourlyActivity_merged
GROUP BY Id
ORDER BY TotalSteps DESC;
-- Insight: Identifies the most active users.
```

	Id	TotalSteps	TotalCalories	TotalIntensity
1	8877689391	495623	105746	14025
2	8053475328	457463	91310	13193
3	1503960366	374546	56287	11594
4	2022484408	351712	77633	12535
5	3977333714	328244	45302	10599
6	4388161847	319765	94167	10519

```
Execution finished without errors.
Result: 33 rows returned in 31ms
At line 60:
--3. Data Analysis & insights
-- Total Steps, Calories, and Intensity by User

SELECT
    Id,
    SUM(StepTotal) AS TotalSteps,
    SUM(Calories) AS TotalCalories,
    SUM(TotalIntensity) AS TotalIntensity
FROM hourlyActivity_merged
GROUP BY Id
ORDER BY TotalSteps DESC;
```

```sql
-- Average Hourly Activity Across All Users

SELECT
    strftime('%H', ActivityHour) AS Hour,
    ROUND(AVG(StepTotal), 2) AS AvgSteps,
    ROUND(AVG(Calories), 2) AS AvgCalories,
    ROUND(AVG(TotalIntensity), 2) AS AvgIntensity
FROM hourlyActivity_merged
GROUP BY Hour
ORDER BY Hour;

-- 🧠 Insight: Shows which hours of the day are most active across all users. Useful for Bellabeat to suggest
-- optimal workout times.
```

	Hour	AvgSteps	AvgCalories	AvgIntensity
1	NULL	320.17	97.39	12.04

```
Execution finished without errors.
Result: 1 rows returned in 32ms
At line 71:
-- Insight: Identifies the most active users.


-- Average Hourly Activity Across All Users

SELECT
    strftime('%H', ActivityHour) AS Hour,
    ROUND(AVG(StepTotal), 2) AS AvgSteps,
    ROUND(AVG(Calories), 2) AS AvgCalories,
    ROUND(AVG(TotalIntensity), 2) AS AvgIntensity
FROM hourlyActivity_merged
GROUP BY Hour
ORDER BY Hour;
```

```sql
-- Top 5 Most Active Hours for a Specific User
SELECT
    ActivityHour,
    StepTotal,
    Calories,
    TotalIntensity
FROM hourlyActivity_merged
WHERE Id = 1503960366
ORDER BY StepTotal DESC
LIMIT 5;
-- 🧠 Insight: Helps understand user behavior at peak performance times.
```

	ActivityHour	StepTotal	Calories	TotalIntensity
1	4/22/2016 9:00:00 PM	5934	311	159
2	4/27/2016 7:00:00 PM	5786	318	150
3	4/25/2016 9:00:00 PM	4895	267	132
4	4/24/2016 7:00:00 PM	4871	270	131
5	5/8/2016 6:00:00 PM	4472	261	115

```
At line 85:
-- 🧠 Insight: Shows which hours of the day are most active across all users. Useful for Bellabeat to suggest
-- optimal workout times.



-- Top 5 Most Active Hours for a Specific User
SELECT
    ActivityHour,
    StepTotal,
    Calories,
    TotalIntensity
FROM hourlyActivity_merged
WHERE Id = 1503960366
ORDER BY StepTotal DESC
LIMIT 5;
```

```sql
-- Sedentary Hours (Zero Steps)
SELECT COUNT(*) AS SedentaryHours
FROM hourlyActivity_merged
WHERE StepTotal = 0;
--🧠 Insight: Shows how often users are inactive — Bellabeat could encourage movement reminders.
```

	SedentaryHours
1	9297

```
Execution finished without errors.
Result: 1 rows returned in 9ms
At line 101:
-- 🧠 Insight: Helps understand user behavior at peak performance times.


-- Sedentary Hours (Zero Steps)
SELECT COUNT(*) AS SedentaryHours
FROM hourlyActivity_merged
WHERE StepTotal = 0;
```

```sql
-- Active vs Inactive Hours Distribution
SELECT
    CASE
        WHEN StepTotal = 0 THEN 'Inactive'
        ELSE 'Active'
    END AS ActivityStatus,
    COUNT(*) AS HourCount
FROM hourlyActivity_merged
GROUP BY ActivityStatus;
--🧠 Insight: Bellabeat can tailor health prompts or coaching messages around inactivity.
```

	ActivityStatus	HourCount
1	Active	12802
2	Inactive	9297

```
Execution finished without errors.
Result: 2 rows returned in 22ms
At line 108:
--🧠 Insight: Shows how often users are inactive — Bellabeat could encourage movement reminders.


-- Active vs Inactive Hours Distribution
SELECT
    CASE
        WHEN StepTotal = 0 THEN 'Inactive'
        ELSE 'Active'
    END AS ActivityStatus,
    COUNT(*) AS HourCount
FROM hourlyActivity_merged
GROUP BY ActivityStatus;
```

```sql
--Calories Burned by Hour Across All Users
SELECT
    strftime('%H', ActivityHour) AS Hour,
    ROUND(SUM(Calories), 2) AS TotalCaloriesBurned
FROM hourlyActivity_merged
GROUP BY Hour
ORDER BY TotalCaloriesBurned DESC;
--🧠 Insight: Ideal for plotting a heatmap or bar chart in Power BI to visualize energy expenditure patterns.
```

	Hour	TotalCaloriesBurned
1	*NULL*	2152150.0

```sql
-- Users with Consistently High Hourly Steps (>500 avg/hour)
SELECT
    Id,
    ROUND(AVG(StepTotal), 2) AS AvgStepsPerHour
FROM hourlyActivity_merged
GROUP BY Id
HAVING AvgStepsPerHour > 500
ORDER BY AvgStepsPerHour DESC;
-- 🧠 Insight: Target high-performance users for premium product marketing or gamification.

```

	Id	AvgStepsPerHour
1	8877689391	674.32
2	8053475328	622.4
3	1503960366	522.38