
SUM
GARMIN
CONNECT
AYSHA
NIDA
MOHAMMED


A LOVE STORY
GARMIN
Connect

This is the story of a man and his one true love…

… his fitness tracker.

12.2% of Americans use a smartwatch or a fitness tracker.

World's population in 2023 that used a smart wristband device **14.4%**

Source: Dataportal

PROJECT OVERVIEW

GOAL

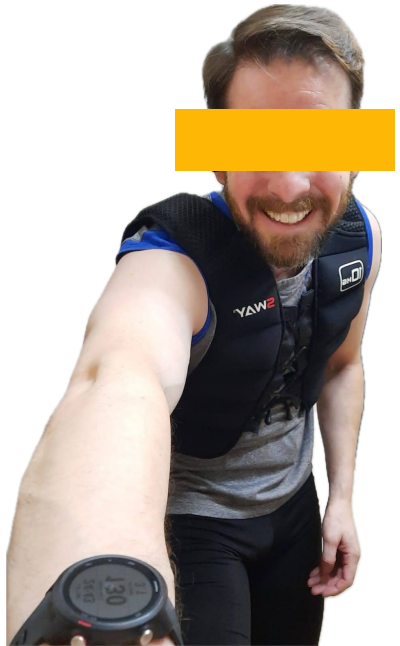
Develop a data engineering solution to analyse fitness patterns from Garmin wearable devices.

DATA SOURCE

Anonymised data from a Garmin Connect watch, used with consent.

PURPOSE

To structure rich datasets into a database, enabling potential visualisation and tracking of health and fitness.



PRIVACY FIRST

Strict adherence to privacy, with no personally identifiable information (PII) used.

SCALABILITY

Designed for expansion, adding more data under strict ethical standards.

ETHICAL CONSIDERATIONS

Commitment to Privacy and Ethics in Data

Informed Consent

Data provided voluntarily for non-commercial use.

Anonymity

Analysis ensuring individual privacy and confidentiality.

Ethical Use

Data handling aligns with high ethical values, prioritising user trust and integrity.

FUTURE OUTLOOK

Building a Foundation for Data-Driven Fitness Analysis

Expandable Model

Ready to integrate additional data following ethical guidelines.

Research and Development

Aiding individuals and researchers in understanding physical activity.

Data Protection

Ongoing commitment to robust privacy and data security practices.

ETL PROCESS

TRANSFORMATION

Data type conversions

Convert date and time strings to datetime objects.

```
# Function to convert time strings to minutes using regex
def convert_time_str_to_minutes(time_str):
    # For 'Best Lap Time' format MM:SS.HH
    if '.' in time_str:
        match = re.match(r'(\d+):(\d{2}).(\d{2})', time_str)
        if match:
            minutes, seconds, hundredths = map(int, match.groups())
            return round(minutes + seconds / 60 + hundredths / 6000, 2)
    # For 'Moving Time' and 'Elapsed Time' format HH:MM:SS or MM:SS
    else:
        match = re.match(r'(\d+):(\d{2}):?(\d{2})?', time_str)
        if match:
            parts = match.groups()
            if len(parts) == 3 and parts[2] is not None:
                hours, minutes, seconds = map(int, parts)
                return round(hours * 60 + minutes + seconds / 60, 2)
            elif len(parts) >= 2:
                minutes, seconds = map(int, parts[:2])
                return round(minutes + seconds / 60, 2)
    return None
```

Cleaning missing or invalid values

Remove rows with missing or invalid data.

Feature engineering

Calculate additional metrics such as average pace and lap distance.

Data filtering

Retain rows for specific activity types (e.g., Running and Swimming).

Concatenation of text columns for composite keys

Combine Activity ID and Activity Type ID to create unique metric IDs.

Resetting DataFrame indices

Reset DataFrame indices to ensure consistency and avoid index-related errors.

LOADING

activities.csv

Contains basic details about each activity, such as activity ID, activity type ID, date, and title.

activity_types.csv

Provides a reference table for activity types, with unique activity type IDs and corresponding names.

performance_metrics.csv

Stores specific performance metrics for deeper analysis, including distance, calories burned, time, heart rate, and pace.

lap_metrics.csv

Captures lap-specific metrics such as lap time, number of laps, total distance, and moving/elapsed time.

elevation_metrics.csv

Records elevation-related metrics, including total ascent, total descent, and minimum/maximum elevation.

USE OF NBCONVERT

```
#Use nbconvert to export noteboook to transform.py in the etl directory
!jupyter nbconvert --to script etl_test.ipynb --output-dir ../etl --output transform
```

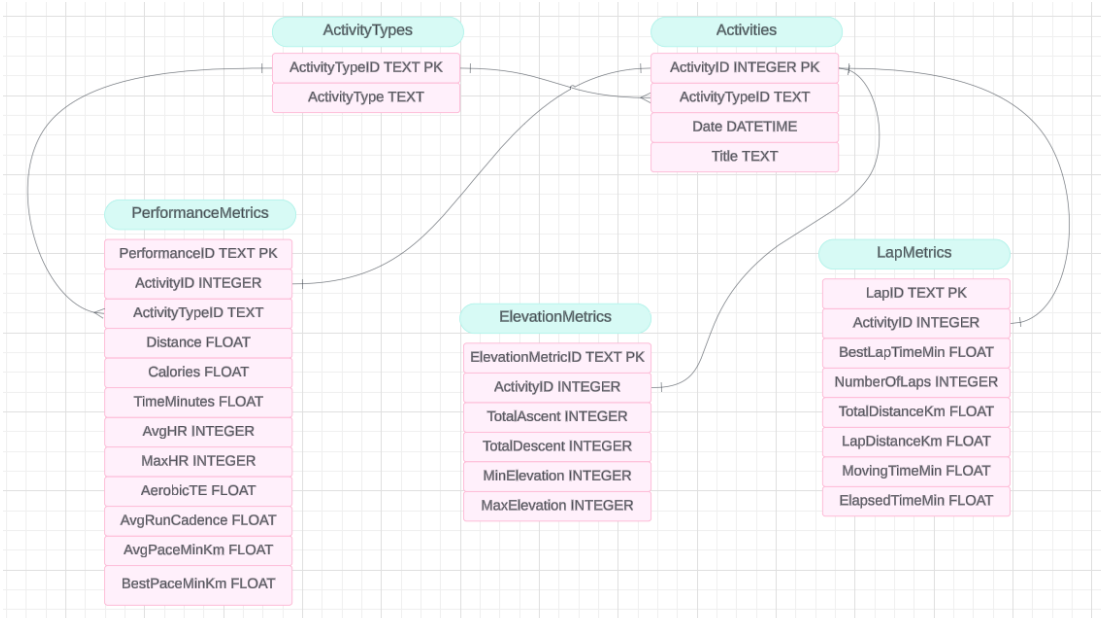
```
[NbConvertApp] Converting notebook etl_test.ipynb to script
[NbConvertApp] Writing 12685 bytes to ../etl/transform.py
```

DATABASE DESIGN

SAMPLE CODE

```
# Creating SQLite database
conn = sqlite3.connect('../database/database.sqlite')
cursor = conn.cursor()


# Table Definition
create_table_1 = '''CREATE TABLE IF NOT EXISTS ActivityTypes(
                    ActivityTypeID Text PRIMARY KEY,
                    ActivityType Text NOT NULL);
                    '''

# Creating the table into our database
cursor.execute(create_table_1)


# insert the data from the DataFrame into the SQLite table
activity_types_df.to_sql('ActivityTypes', conn, if_exists='replace', index = False)

# Printing pandas dataframe
pd.read_sql('''SELECT * FROM ActivityTypes''', conn)


# Table Definition
create_table_2 = '''CREATE TABLE IF NOT EXISTS Activities (
                    ActivityID Integer PRIMARY KEY,
                    ActivityTypeID Text NOT NULL,
                    Date Datetime,
                    Title Text NOT NULL,
                    FOREIGN KEY (ActivityTypeID) REFERENCES ActivityTypes (ActivityTypeID)
                    );
                    '''

# Creating the table into our database
cursor.execute(create_table_2)

# insert the data from the DataFrame into the SQLite table
activities_df.to_sql('Activities', conn, if_exists='replace', index = False)

# Printing pandas dataframe
pd.read_sql('''SELECT * FROM Activities''', conn)
```

ERD



DATABASE INTEGRITY CONSTRAINTS

Primary Keys

Each table has a primary key constraint to ensure uniqueness and identify each record uniquely.

Foreign Keys

Foreign key constraints establish relationships between tables, ensuring referential integrity and enforcing data consistency.

Unique Constraints

Certain attributes, such as ActivityTypeID and ActivityType in the ActivityTypes table, have unique constraints to prevent duplicate entries.

API

LAUNCHING FLASK APP

```
api — Python ◂ Python app.py — 81×24
Last login: Mon Feb 12 08:47:57 on ttys000
(base) nida@Nidas-Macbook api % conda activate dev
(dev) nida@Nidas-Macbook api % ls
app.py          static          templates
(dev) nida@Nidas-Macbook api % Python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 337-789-030
```


```
{
  "meta": {
    "title": "Garmin device - activities",
    "Access time": "14/02/2024 10:52:55",
    "Num of records": 413,
    "data": {
      "Activity id": {
        "ActivityTypeID": "string",
        "Date": "YYYY-MM-DD HH-MM-SS",
        "Title": "Activity name"
      }
    }
  },
  "data": {
    "1": {
      "ActivityTypeID": "AT001",
      "Date": "2024-01-22 20:19:55",
      "Title": "Treadmill Running"
    },
    "2": {
      "ActivityTypeID": "AT001",
      "Date": "2024-01-18 17:02:50",
      "Title": "Treadmill Running"
    },
    "3": {
      "ActivityTypeID": "AT001",
      "Date": "2024-01-15 19:33:44",
      "Title": "Treadmill Running"
    },
```


JSON RESPONSE STRUCTURE

The JSON response from each endpoint includes both metadata and data sections.

Responses are structured to include metadata for informational purposes and a data array containing the requested records.

API ENDPOINTS

Activity Types

- Description: Retrieve information on the various types of activities.

Activities

- **GET** `/api/activities/all`
  - Description: Fetch details on all activities undertaken.
- **GET** `/api/activities/activity-type/{ActivityTypeID}`
  - Description: Obtain activities filtered by a specific activity type.
  - Parameters:
    - `ActivityTypeID` : The ID representing the type of activity.
- **GET** `/api/activities/limit/{number_of_records}`
  - Description: Retrieve a limited number of activity records.
  - Parameters:
    - `number_of_records` : The maximum number of records to return.
- **GET** `/api/activities/date/{start_date}/{end_date}`
  - Description: Get activities within a specified date range.
  - Parameters:
    - `start_date` : The start date in the format YYYY-MM-DD.
    - `end_date` : The end date in the format YYYY-MM-DD.

Elevation Metrics

- **GET** `/api/elevation-metrics/all`
  - Description: Access all elevation data during running activities.
- **GET** `/api/elev-metrics/limit/{number_of_records}`
  - Description: Fetch a limited number of elevation metrics records.
  - Parameters:
    - `number_of_records` : The maximum number of records to retrieve.

Performance Metrics

- **GET** `/api/performance-metrics/all`
  - Description: Access all data on performance metrics during activities.
- **GET** `/api/performance-metrics/activity-type/{activity_type_id}`
  - Description: Fetch performance metrics filtered by activity type.
  - Parameters:
    - `activity_type_id` : The ID of the activity type.
- **GET** `/api/performance-metrics/limit/{number_of_records}`
  - Description: Limit the number of performance metrics records returned.
  - Parameters:
    - `number_of_records` : The maximum number of records to fetch.

Lap Metrics

- **GET** `/api/lap-metrics/all`
  - Description: Retrieve all lap data during running activities.
- **GET** `/api/lap-metrics/limit/{number_of_records}`
  - Description: Obtain a limited number of lap metrics records.
  - Parameters:
    - `number_of_records` : The maximum number of records to return.

USAGE NOTES

Replace placeholders in curly braces {} with actual values.

The date format for endpoints involving dates is **YYYY-MM-DD**.

And they all exercised …

… happily ever after.