1. System Architecture & Approach

Our project implements a **TCP client-server system** that allows users to run specific queries on IoT data retrieved from a PostgreSQL (NeonDB) database. The **client** prompts the user to choose one of three queries regarding moisture, water usage, or electricity consumption. This message is sent to the **server**, which processes the request by retrieving relevant sensor data from the database and returns the result.

Key components:

- client.py: Handles user input and TCP message sending.
- server.py: Handles connections, queries NeonDB, and responds.
- NeonDB: Stores both metadata and virtual sensor data downloaded from Dataniz.com.
- Dataniz: Provides realistic metadata/sensor data structure for IoT devices.

2. IoT Sensor & Data Research

Our research focused on simulating realistic smart home appliances:

- **Kitchen Fridge:** Contains a **Moisture Sensor** measuring relative humidity (%RH).
- **Smart Dishwasher:** Contains a **Float Switch Sensor** measuring water consumption (liters).
- **All devices** use an **Ammeter Sensor** for electricity usage (kilowatt-hours, estimated via current).

We ensured sensor data units were respected during calculations and modeled expected behaviors (e.g., cyclical water use for dishwashers, ongoing current draw for fridges).

3. Metadata Usage from Dataniz

Dataniz metadata was essential for filtering and mapping sensor data correctly:

- Queried smart_fridge_data_metadata to obtain "assetUid" for the **Kitchen Fridge** and **Smart Dishwasher**.
- Used "parent_asset_uid" in smart_fridge_data_virtual to ensure only relevant data for each device was analyzed.
- This prevented data mixing and improved the accuracy of time- and device-based queries.

For electricity data, metadata filtering was **not applied** directly, as the virtual table contained all ammeter values with distinct field names. In future iterations, mapping each ammeter to a UID could improve precision.

4. Algorithms & Calculations

We implemented the following:

- **Average Moisture (Last 3 Hours):** Filtered moisture readings by updatedAt timestamps within a 3-hour range. Converted string values to float and calculated the mean.

- **Average Water Usage per Cycle:** Aggregated float values from the dishwasher's water sensor and computed the mean.

- **Highest Electricity Consumption:** Summed total current draw values across three devices and identified the maximum.

No unit conversions were required as the data values were stored and interpreted directly using standardized units (e.g., %, liters, amps).

5. Challenges & Resolutions

- **Challenge:** Matching metadata with virtual sensor data.
    - **Solution:** Used "assetUid" and "parent_asset_uid" fields to join and filter data correctly.
- **Challenge:** Understanding timestamp behavior.
    - **Solution:** Used pytz.UTC and datetime.now() in the server to match the virtual data's UTC timezone.
- **Challenge:** Data sparsity and occasional nulls.
    - **Solution:** Added conditionals to ignore None or missing values during aggregation.

6. Feedback on Dataniz.com

Positive Aspects:

- Clean and intuitive UI.
- Realistic simulation of smart devices.

- Easy export of metadata and payload data.

Suggestions for Improvement:

- Add **live payload previews** within metadata view to speed up development.
- Improve **filter and search** options to allow quick identification of devices or sensors.
- Add **timezone configuration support** or at least display timezones clearly to reduce confusion.