**Note:** The demo video of my solution is attached to the same zip file. Would recommend watching the demo of the running code.

**Solution:**

* The chosen solution is a combination of relational database like MySQL and OLAP database for analytical processing using Presto providing a structured way to organize and store data in MySQL and making it easier to query the aggregated the data in Presto for analytics. Querying the RDBMS for analytical or aggregate queries can be very expensive, hence, I have separated the solution into relational and analytical data lake.
* The solution utilizes the 'jsonschema' and 'pandas' libraries for validation and transformation offering robust ecosystem.
* It employs the 'mysql-connector-python' and 'sqlalchemy' libraries enabling seamless database connectivity and ORM.
* The solution ensures data integrity, scalability, and flexibility while maintaining security by loading the database configurations from environment variables.

**Data Pipeline Architecture:**

JSON files are dumped into AWS S3 bucket, containerized through Docker by running dockerfile via a bash script, orchestrated through Airflow and finally loaded to MySQL Datawarehouse tables and aggregated data is processed using Presto engine which can be used by all consumers like Data Analysts, Data Scientists and Software Engineering team.

A picture containing text, diagram, font, screenshot

Description automatically generated

1. Load all the JSON files from AWS S3 data lake
2. Initial data extraction and data validation leveraging JSON Schema Validator in Python
3. Generating erroneous files in csv format with filename and error message into “roof\_json\_errors” folder to identify any syntax issues or missing fields in JSON files at early stage of the pipeline.
4. transform\_data() function in python script does all the necessary Data Transformation using Pandas DF
5. Finally, the transformed data is loaded into MySQL database tables and the aggregate data is processed in Presto data lake.
6. The final tables aggregated tables in Presto are used for data visualization tools like tableau, PowerBI for data analysis by data analyst and the MySQL staging data can be used by data scientists for predictions and can also be parsed to API as input parameters.

**Assumptions**:

Since the data provided was not clean and a few attributes were missing, the data pipeline is

based on certain assumptions:

1. The files are only added once per day and only the latest data is dumped into the folder

with version columns being modified. If in future the requirement changes to load the JSON files immediately, we can use messaging queue like **Kafka** where the submitted JSONs will be picked at a time and ingested into the database.

1. We can have multiple installationId in json files, hence, the sites table will have a

combination of id, installationId as its primary key.

1. “Building” object inside JSON does not have a ‘id’ field, hence, I have created an auto increment id to link buildings to its sites. We could have also used the combination of ‘isPrimaryBuilding’ and ‘totalRoofArea’ as its primary key because the ‘totalRoofArea’ field has a long decimal value. However, it might fail if two buildings have the same totalRoofArea.
2. I have also assumed that the structure of the json remains the same for all the files.
3. Certain fields are mandatory and cannot be null. Example: azimuthAngle, roofMaterialType

**Analyst:**

A screenshot of a computer

Description automatically generated with low confidence

A screenshot of a computer

Description automatically generated with medium confidence

**Data Scientist:** They can leverage the following columns from the tables created for feature engineering of roof properties:

1. Roof Area
2. pitchAngle
3. azimuthAngle
4. centroid\_x
5. centroid\_y
6. centroid\_z
7. azimuthVector\_x
8. azimuthVector\_y
9. azimuthVector\_z
10. northVector\_x
11. northVector\_y
12. northVector\_z
13. headingVector\_x
14. headingVector\_y
15. headingVector\_z

**Roof damage prediction** using linear regression can be used to identify roofs that are at risk of damage based on features such as age of the roof, type of roof material, mounting plane angles, climate. This information can be used to prioritize repairs and prevent costly damage.

**Solar panel placement optimization** can be used to place solar panels in the most efficient locations based on features like slope of the roof, location of obstructions, azimuth angle, pitch angle, coordinates. This can help to maximize the amount of energy that solar panels generate.

**Insurance pricing** can be used to price insurance policies using decision trees or support vector machines based on features such as age of the roof, roof material, missing shingles, etc. This information can be used to calculate the risk of roof damage, so that insurance premiums can be set accordingly.

**Software Engineering:**

**Endpoint**: POST /tiling

**Input**: JSON payload with “installationId”, "mounting\_plane\_id", "pitch\_angle", and "azimuth\_angle", “obstructions”

**Output**: JSON response with “installationId”, "mounting\_plane\_id”, “tile\_coordinates", "tile\_orientation", “slope”

The tiling API would use the mounting plane angles to determine the best placement for the solar panels. The solar panels would be placed in a way that maximizes the amount of sunlight that they receive. The tiling API would also consider the slope of the roof and the location of any obstructions.

The output of the tiling API would be a JSON object for installing the solar panels. These instructions would include the location of each solar panel, the angle at which each solar panel should be mounted, and the type of mounting hardware that should be used.

**Data Issues discovery:**

While examining the JSON files and running validations in python script, I had identified following issues:

1. Roof\_9.json has JSON syntax error - Invalid JSON format in file
2. Roof\_2.json has a missing field – azimuthAngle
3. Roof\_6.json has a misspelled field – mountingPlanes

**Setup:**

Graphical user interface

Description automatically generated with medium confidence

Graphical user interface, text, application, email

Description automatically generated

A screenshot of a computer program

Description automatically generated with medium confidence