# Lead Quality Analysis Tool

Daniel Coelho
Recruitment exercise for HomeBuddy

***Abstract*—This project focuses on analyzing and improving lead quality by processing data from Parquet files and client CRM reports. The goal is to determine conversion rates, clean and standardize datasets, and perform insightful data analysis using PySpark. To optimize performance and scalability, efficient functions were created for concatenating, reading, and processing Spark DataFrames. The technical stack employed for this project includes Python, PySpark, Parquet, and CSV formats.**

***Index Terms*—Analysis, Data Warehouse, Standarization, Python.**

## I. Pre-Analysis

### A. Problem Overview

This project is designed to match data from the backend system with the client's CRM reports to analyze and improve lead quality. The process runs twice daily and calculates key metrics such as the conversion rate from delivered leads to homeowner appointments and the progression of leads in the sales process.

### B. What did we start with

- **Parquet File**: Contains leads delivered to the client, exported from the backend system.
- **Client Reports**: The client manually creates reports with lead progress and uploads them to an SFTP server as CSV snapshots.
- **Matching Data**: The only link between our leads and the client's reports is the hashed email and phone numbers.
- **Key Metrics**
  - **Conversion Rate**: From delivered leads to appointments (tracked in the set column).
  - **Appointments**: Scheduled appointment dates (tracked in the appt date column).
  - **Demos**: Whether an appointment resulted in a demo, the next step in the sales process.

### C. Data Structure

- **Entry Date**: Date when the entry was made (e.g., 02/05/2023).
- **Lead Number**: Unique number identifying each lead.
- **Emai Hash**: A hashed value of the email.
- **Phone Hash**: A hashed value of the phone number.
- **City**: City name.
- **State**: State abbreviation (e.g., WA for Washington).
- **Zip**: Postal code.
- **Appt Date**: Date and time of an appointment, if available.
- **Set**: Binary field indicating whether something is set (0 or 1).
- **Demo**: Binary field, likely indicating whether a demo has occurred (0 or 1).
- **Dispo**: Disposition status (e.g., CXL-BT, DQ - ROBO-CALL, Data).
- **Job Status**: Status of the job, but there are many missing values here

### D. Initial Observations

- **Missing values**: The Appt Date and Job Status columns have a high number of missing or null values. More precisely Appt Date around 79%, and Job Status around 97%.
- **Date format**: Appt Date has a legacy date. To deal with this kind of date format, I need to enable a spark configuration called $spark.sql.legacy.timeParserPolicy$
- **Possible duplicates**: Multiple entries have the same email_hash and phone_hash but different LEADNUMBER. More precisely 4365 email and phone hash with different lead numbers.
- **Column Consistency**: While the CSV snapshots may vary slightly in structure, they generally share common elements. For instance, some snapshots include a $location$ column, a $cityname$ column, and a $city$ column, all of which essentially refer to the same information: the city. The location column often provides the city name in a specific format, such as $city|state$.

### E. Potential Data Quality Issues

- **Duplicate leads**: You might encounter duplicates based on email hash and phone hash even if the lead number differs.
- **Zip code inconsistencies**: Some ZIP codes are floating-point numbers, indicating potential issues with leading zeros (e.g., ZIP might be 97146.0).
- **Inconsistent boolean formats**: Demo and Set have different formats for booleans. This should be standarized.

### F. Potential Data Quality Tests

- A lead is a potential customer or client. In business and marketing, it refers to individuals or entities that have expressed interest in a product or service but haven't yet made a purchase. Typically, each unique email or phone number should correspond to a single lead. If multiple lead numbers are associated with the same contact information, it suggests there may be duplicate or incorrect data. However, in some cases, there might be valid reasons for this, such as different departments handling the same contact. It's important to carefully review such instances to ensure data accuracy and avoid duplication.
- Each city belongs to a unique state and has a specific zip code. While multiple cities can exist within a single state, no city can belong to more than one state.

- An appointment status of ”Set”requires a corresponding ”Appointment Date.”Additionally, a demo cannot occur without a previously scheduled appointment.
- Every report item represents a lead’s progression and must be assigned a specific work status. The absence of a work status would indicate an incomplete or inconsistent record.

## II. Development

This section will provide a comprehensive overview of the project’s design, including the organization of the ETL process and the rationale behind specific decisions.

### A. Project Structure

```
LEAD_QUALITY_ANALYSIS/
│
├── source/
│   ├── contractor_reports/   # Contains client report data in CSV format
│   └── leads/                # Contains leads data in Parquet format
│
├── spark_controller/
│   ├── __init__.py           # Package initialization
│   ├── controller.py         # Utils script with useful functions to help the process
│   └── schemas.json          # JSON file with schema definitions
│
├── 1_ingestion_and_cleaning.py     # Data Ingestion and Cleaning Logic
├── 2_data_modeling.py              # Data Matching and Modeling Logic
├── 3_analysis_and_calculations.py  # Analysis and Metrics Calculation Logic
│
├── docker-compose.yml          # Docker Compose setup for containerized execution
├── Dockerfile                  # Docker image definition
├── requirements.txt            # Project dependencies
```

Fig. 1. Project Structure

#### a.1) Source

This directory contains the raw data sources used for the analysis.

- $contractor\_reports/$: This folder stores client report data in CSV format.
- $leads/$: This folder stores Leads data in Parquet format.

#### a.2) Spark Controller

This directory contains the core Spark-based logic and utility functions. For a detailed explanation of this controller’s functionality, please refer to the **README.md** file within this module’s dedicated repository.

- $init.py$: This file is used for package initialization and imports in Python.
- $controller.py$: This script contains helper methods used throughout the analysis process.
- $schemas.json$: This JSON file defines the schema or structure of the data source, including column names, data types, and other metadata.

#### a.3) Python Scripts

These python scripts reflects the project stages mention before.

- $1\_ingestion\_and\_cleaning.py$: This script handles the initial data ingestion and cleaning tasks, such as reading data from files, handling missing values, and transforming data into a suitable format for analysis.
- $2\_data\_modeling.py$: This script focuses on data matching, modeling, and creating relationships between different datasets. It involve tasks like joining tables, creating derived columns, and aggregating data.
- $3\_analysis\_and\_calculations.py$: This script performs the actual data analysis, calculations, and metric generation. It involve statistical analysis, and custom calculations based on the specific goals of the project.

#### a.4) Docker-related files

- $docker-compose.yml$: This file defines the dependencies for running the project in a containerized environment.
- $Dockerfile$: This file contains instructions for building the Docker image, including specifying the base image, copying files, installing dependencies, and setting environment variables.
- $requirements.txt$: This file lists the Python packages for the project.

### B. Project Stages

- Data Ingestion and Cleaning
  - Load the Parquet file containing lead data.
  - Load CSV snapshots from the client’s CRM.
  - Standardize the data by cleaning and preparing it for matching.
- Data Matching and Modeling
  - Match leads based on hashed email and phone numbers.
  - Create a data warehouse model to link all data.
- Analysis and Metrics Calculation
  - Calculate conversion rates for lead quality analysis.

#### b.1) Data Ingestion and Cleaning

The data ingestion and cleaning stage starts with reading the source data. The Spark Controller is the key tool to process the data along the project.

```
# Extract information of the Leads
folder_path = os.getenv('LEADS_PREFIX', 'leads')

dataframes = []
for filename in os.listdir(folder_path):
    dataframe = controller.read_parquet_file(
    folder_path=folder_path, filename=filename
    )
    dataframe = Dataframe(dataframe=dataframe)\
            .clean_dataframe()\
            .enforce_schema(schema_name='leads')\
            .to_dataframe()

    dataframes.append(dataframe)

leads = controller.concatenate(dataframes=dataframes)
leads.write.mode("overwrite").parquet('step1/leads/')
```

```python
# Extract information from Client Reports
folder_path = os.getenv(
    'REPORTS_PREFIX', 'contractor_reports'
)

dataframes = []
for filename in os.listdir(folder_path):
    dataframe = controller.read_csv_file(
        folder_path=folder_path, filename=filename
    )
    dataframe = Dataframe(dataframe=dataframe)\
        .clean_dataframe()\
        .enforce_schema(schema_name='reports')\
        .clean_invalid_values(
            "email_hash", invalid_values=['-----']
        )\
        .clean_invalid_values(
            "phone_hash", invalid_values=['-----']
        )\
        .clean_invalid_values(
            "dispo", invalid_values=['-----']
        )\
        .clean_invalid_values(
            "job_status", invalid_values=['-----']
        )\
        .clean_invalid_values(
            "city_name", invalid_values=['-----']
        )\
        .to_dataframe()

    dataframes.append(dataframe)

reports = controller.concatenate(dataframes=dataframes)
reports.write.mode("overwrite")
            .parquet('step1/reports/')
```

Both sections perform comparable tasks, beginning with the reading of CSV or Parquet files using helper functions provided by the Spark Controller. Subsequently, they instantiate the DataFrame class, which offers a range of functions for data manipulation. From those data manipulation methods, both sections begin with the $cleaning\_dataframe()$ function, which cleans the data by removing extra spaces, duplicated rows, and rows where all the columns are NULLs.

The next part, the $enforce\_schema(schema\_name)$ function, is indeed the most interesting part of the process. This schema enforcement works as a Service Level Agreement (SLA). Both establish standards and expectations. The function defines the expected data structure and quality, while an SLA outlines service performance metrics like uptime, response time, and accuracy. Both strive to ensure consistency and reliability in their respective domains. The $enforce\_schema$ function maintains data integrity and consistency, while an SLA guarantees a certain level of service quality and reliability.

```javascript
// schemas.json
{
    "name": "reports",
    "entries": [
        //...
        {
            "name": "ENTRYDATE",
            "new_name": "entry_date",
            "type": "date",
            "format": "mm/dd/yyy"
        },
        {
            "name": "STATE",
            "type": "string",
            "format": "[a-zA-Z]{2}"
        },
        {
            "name": ["CITY", "CityName"] ,
            "new_name": "city_name",
            "type": "string"
        }
        //...
    ]
}
```

Here's a breakdown of what this function typically does:

- **Retrieves Schema**: The function obtains the specified schema definition from a configuration file called $schemas.json$.
- **Enforces Data Types**: The schemas.json file outlines the expected data types and the expected formats for each column. These data types and formats are strictly enforced within their corresponding columns. If a value cannot be successfully converted to its designated type, it indicates a data inconsistency. In such cases, the value is replaced with null to maintain data integrity.

  For instance, if a column is defined as a date, but a value within that column is not a valid date format, it will be replaced with null. However, even when dealing with string columns, inconsistencies can arise. We'll delve into those scenarios later.

  To ensure transparency, the function always logs any detected invalid values, providing valuable insights for data quality assessment.
- **Enforces Column Names**: The schemas.json file not only specifies the expected data types but also outlines the desired column names. If a column name differs from the expected name, the function will attempt to replace it.

  When multiple expected names are provided as a list, it indicates that these columns are synonymous. In such cases, the function will prioritize the first matching column found.

  If no new column name is specified, the function will automatically determine the name by converting the existing name to snake case. This involves replacing spaces with underscores.

By enforcing the schema, this function plays a crucial role in safeguarding data quality, which prevents problems later on.

By using this function, we can keep only the correct values, like making sure the 'entry date' is always in the mm/dd/yyyy format and that 'demo' and 'set' are true or false. Also, the 'state' is always two capital letters.

The lead extraction process culminates in a Spark DataFrame returned by the function $to_dataframe()$. Subsequently, the report extraction phase proceeds with a $clean\_invalid\_values$ function that addresses string column inconsistencies. While the schema enforcement mechanism effectively handles anomalies with data type casting, it not captures string-based anomalies.

To rectify this, the $clean\_invalid\_values$ function is employed to replace a specified list of invalid values with NULL. This function operates straightforwardly, but a preliminary analysis by the developer is essential to identify those invalid values.
At the end, both extractions finished with a write of the data.

### *b.2) Data Modeling - Star Schema*

A Star Schema is an ideal data modeling approach for this project. A Star Schema is a type of database schema architecture commonly used in data warehousing and business intelligence to organize data in a way that facilitates efficient querying and reporting. It is called a "star"schema because its structure resembles a star: there is a central fact table connected to multiple surrounding dimension tables. Here's a breakdown of the key components and characteristics:

Key Components of Star Schema

- Fact Table: The central table in the schema.
  - Stores quantitative data or measures (e.g., sales amount, revenue, quantities).
  - Contains foreign keys that link to the dimension tables.
  - Often contains aggregated data (e.g., total sales per product, total revenue per region).
  - Rows in the fact table are typically transactional records with facts or metrics that describe business processes.
- Dimension Tables: Surround the fact table and provide context for the measures.
  - Store descriptive attributes related to the facts (e.g., product name, customer location, time of purchase).
  - Are usually smaller than fact tables.
  - Contain a primary key that corresponds to a foreign key in the fact table.
  - Dimensions of this project
    - Date/Time Dimension (for tracking periods like day, month, quarter)
    - Leads Dimension (for tracking lead-related information)
    - Places Dimension (for tracking city-related information)
    - Status Dimension (for tracking changing-status information)

The Star Schema provides:

- **Clarity and Simplicity**: Star Schema's straightforward structure makes it easy to understand and visualize data relationships.
- **Scalability**: As our data grows, Star Schema can easily accommodate additional information.
- **Business Alignment**: Star Schema aligns with how businesses often think about data, making analysis more intuitive.
- **Data Integrity**: The schema is designed to ensure consistent data (e.g., report items are consistently tied to specific lead and time).

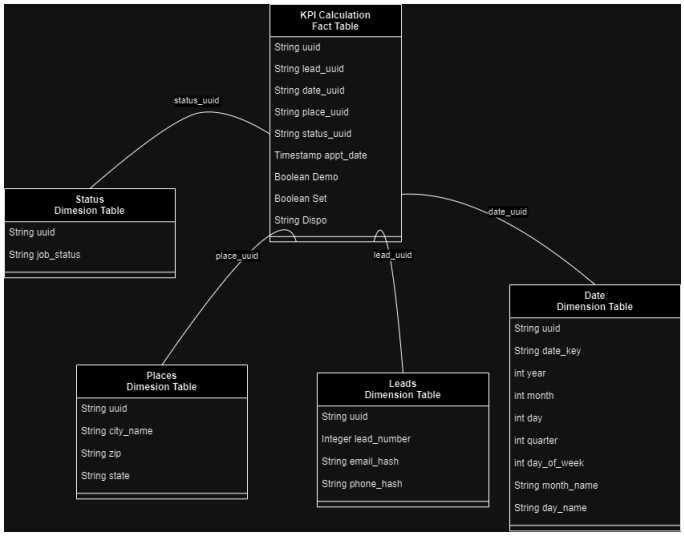


Fig. 2. Lead Quality Analysis Tool

### *b.3) Data Modeling - Step-by-Step*

Using the helper functions, we start with getting the information from the Stage 1, the raw data standardized.

```
# Read information from the STEP 1
leads = controller.read_parquet_files(
    folder_path='step1/leads/'
)
leads.createOrReplaceTempView('leads')
reports = controller.read_parquet_files(
    folder_path='step1/reports/'
)
reports.createOrReplaceTempView('reports')
```

Before spliting the raw data into Dimension Tables. We build a Date Dimension with date attributes since 2000 until 2030. To that data, we give a deterministic UUID based in the primary key of the table $date\_key$. Is this UUID that will link the fact table with the Date Dimension. It will be the entry date the initial link, and afterwards, we will use only this UUID to link the Date Dimension with the Fact Table.

```
# Define the start and end dates
start_date = "2000-01-01"
end_date = "2030-12-31"
```

```
# Create a temporary view to generate a date range
date_range_df = controller.spark.sql(f"""
    SELECT explode(sequence(
        to_date('{start_date}'),
        to_date('{end_date}'),
    interval 1 day)) AS date_key
""")
date_range_df.createOrReplaceTempView('date_range')

# Create the Date Dimension with additional attributes
date_dimension_sql = """
SELECT date_key,
    year(date_key) AS year,
    month(date_key) AS month,
    day(date_key) AS day,
    quarter(date_key) AS quarter,
    dayofweek(date_key) AS day_of_week,
    date_format(date_key, 'MMMM') AS month_name,
    date_format(date_key, 'E') AS day_name
FROM date_range
"""

# Create the Date Dimension DataFrame
date_dimension_df = controller.spark.sql(
    date_dimension_sql
)
date_dimension_df = date_dimension_df.withColumn(
    "uuid",
    deterministic_uuid_udf(*[F.col("date_key")])
)
date_dimension_df.createOrReplaceTempView('d_date')
```

| date_key | year | month | day | quarter | day_of_week | month_name | day_name | uuid |
|---|---|---|---|---|---|---|---|---|
| 2000-01-01 | 2000 | 1 | 1 | 1 | 7 | January | Sat | 5c9f9511-d6f4-577... |
| 2000-01-02 | 2000 | 1 | 2 | 1 | 1 | January | Sun | 9d0392c5-2cfe-570... |
| 2000-01-03 | 2000 | 1 | 3 | 1 | 2 | January | Mon | d59deeb7-f90b-5a8... |
| 2000-01-04 | 2000 | 1 | 4 | 1 | 3 | January | Tue | 3e73c301-af32-525... |
| 2000-01-05 | 2000 | 1 | 5 | 1 | 4 | January | Wed | 5f16ce0c-b7ec-5d2... |
| 2000-01-06 | 2000 | 1 | 6 | 1 | 5 | January | Thu | 0fd36c92-2b91-53f... |
| 2000-01-07 | 2000 | 1 | 7 | 1 | 6 | January | Fri | 3926f0fc-8cea-5f6... |
| 2000-01-08 | 2000 | 1 | 8 | 1 | 7 | January | Sat | 27374de3-a7c1-53b... |
| 2000-01-09 | 2000 | 1 | 9 | 1 | 1 | January | Sun | 9d20ab62-a651-5f0... |
| 2000-01-10 | 2000 | 1 | 10 | 1 | 2 | January | Mon | 3e9f550a-711f-5af... |

TABELA I
DATE DIMENSION

```
leads_dimension_sql = f"""
    SELECT DISTINCT
        leads.lead_uuid AS uuid,
        reports.lead_number,
        leads.phone_hash,
        leads.email_hash
    FROM leads
    LEFT JOIN reports
        ON leads.email_hash = reports.email_hash
        AND leads.phone_hash = reports.phone_hash
"""

# Create the Leads Dimension DataFrame
leads_dimension = controller.spark.sql(
    leads_dimension_sql
)
leads_dimension.createOrReplaceTempView('d_leads')
```

| uuid | lead_number | phone_hash | email_hash |
|---|---|---|---|
| f5ac6be9f8b60942f... | 295372 | bbb4a2d9b9271f8aa... | 1a366316a53642d02... |
| b3ef13509305d6a3e... | 226989 | 92ec6af1b6a71194b... | 8ae8b47e7c66072ee... |
| c4559ed210430a250... | 222649 | 7ee9024dd8d734233... | ce57ed3f309cd59ab... |
| 22f3f1d858386a0ae... | 224580 | ddf5e0ab0585c5546... | 7a874f28910543475... |
| bbe289d3dd005c0bb... | 291911 | 77e0d4082c6343335... | 9f6b3be479a692f3e... |
| 431a7e9120a09dd25... | 225469 | 6f5c3f5b1a5802c75... | 820778c0ec775fedf... |
| 4be094e8ae87f72e3... | 303363 | 4690c2592454b3b45... | 6fb664f8514afb21f... |
| 9ba31d8e726aeb61a... | 227772 | e15f0bdbea792e7c7... | 9c8ecea77d5895144... |
| 3669f20d075e7cc15... | 292667 | 3ebae78134b14e85e... | dc24e88b0147756f5... |
| 6fed42f8531093eae... | 224714 | 62875935e00ec57a0... | 0c75e416a4fc90f03... |

TABELA II
LEADS DIMENSION

```
places_dimension_sql = f"""
    SELECT DISTINCT
        city_name,
        zip,
        state
    FROM reports
"""

# Create the Places Dimension DataFrame
places_dimension = controller.spark.sql(
    places_dimension_sql
)
places_dimension = places_dimension.withColumn(
    "uuid",
    deterministic_uuid_udf(*[
        F.col("city_name"),
        F.col("zip"),
        F.col("state")
    ])
)
places_dimension.createOrReplaceTempView('d_places')
```

| city_name | zip | state | uuid |
|---|---|---|---|
| Las Vegas | 89102 | NV | b6314b7c-8f74-50a... |
| Henderson | 89002 | NV | 4c2a5423-aa69-583... |
| Suquamish | 98392 | WA | 0805827c-b570-523... |
| Seattle | 98199 | WA | 052fdc15-b877-568... |
| Tucson | 85745 | AZ | 3cdeb7bd-dc7a-5e5... |

TABELA III
PLACE DIMENSION

```
status_dimension_sql = f"""
    SELECT DISTINCT job_status FROM reports
"""

# Create the Status Dimension DataFrame
status_dimension = controller.spark.sql(
    status_dimension_sql
)
status_dimension = status_dimension.withColumn(
    "uuid",
    deterministic_uuid_udf(*[F.col("job_status")])
)
status_dimension.createOrReplaceTempView('d_status')
```

| job_status | uuid |
|---|---|
| Cancel After Resc... | c63cab33-2f82-554... |
| Sub Completed | 11a61524-9c8e-52f... |
| Remediaton Hold | 417fa00d-d1bb-512... |
| Completed | be975548-a354-55c... |
| Materials Ordered | fa9254cb-29ba-529... |

TABELA IV
JOB STATUS DIMENSION

```
kpi_calculation_sql = f"""
    SELECT
        reports.entry_date,
        d_leads.uuid AS lead_uuid,
        d_places.uuid AS place_uuid,
        d_status.uuid AS status_uuid,
        d_date.uuid AS date_uuid,
        SUBSTRING(
            reports.appt_date, 12, 8
        ) AS appointment_time,
        reports.demo,
        reports.set,
        reports.dispo
    FROM reports
    LEFT JOIN d_leads
        ON d_leads.email_hash = reports.email_hash
        AND d_leads.phone_hash = reports.phone_hash
        AND d_leads.lead_number = reports.lead_number
    LEFT JOIN d_places
        ON d_places.city_name = reports.city_name
        AND d_places.zip = reports.zip
        AND d_places.state = reports.state
    LEFT JOIN d_date
        ON d_date.date_key = CAST(
                                reports.appt_date
                            AS DATE)
    LEFT JOIN d_status
        ON d_status.job_status = reports.job_status
"""

# Create the KPI Calculation DataFrame
kpi_calculation = controller.spark.sql(kpi_calculation_sql)
kpi_calculation = kpi_calculation.withColumn(
    "uuid",
    deterministic_uuid_udf(*[
        F.col("entry_date"),
        F.col("lead_uuid"),
        F.col("place_uuid"),
        F.col("status_uuid"),
        F.col("date_uuid"),
        F.col("appointment_time"),
        F.col("demo"),
        F.col("set"),
        F.col("dispo")
    ])
)
```

| entry_date | lead_uuid | place_uuid | status_uuid | date_uuid | appointment_time | demo | set | dispo | uuid |
|---|---|---|---|---|---|---|---|---|---|
| 2023-01-19 | be671f5198750b83... | 16fe6d8e-b156-5e4... | 417fa00d-d1bb-512... | dc7b170d-9d1d-502... | 12:00:00 | true | true | Sale | ec0ae656-ea21-52c... |
| 2023-01-26 | cc563d7ddc097117b... | 5ddca93a-66d5-54a... | b422dd82-dd2f-551... | edb91fda-2895-5f7... | 13:00:00 | true | true | Sale | c2031809-2a31-565... |
| 2023-01-10 | a8bc0d4e546345d12ad... | a9722057-54b1-5ae... | 59e8e313-6f60-52f... | 758be6b0-19d6-531... | 13:00:00 | true | true | Sale | 915c63c6-6fcb-507... |
| 2022-01-30 | 8017e4d6f95f3c577... | d8f46f42-4710-54d... | be975548-a354-55c... | 8876fdca-88bd-577... | 12:00:00 | true | true | Sale | 9dfd8bd7-6861-522... |
| 2023-01-14 | 98ddf2e648e9ae6b... | 9dc49e4d-0058-5e2... | 79048bfd-6576-529... | 7f9fa64f-21ed-550... | 09:00:00 | true | true | Sale | a26df9d0-2b4c-515... |
| 2023-01-02 | 3153dfaf0fb9827fe... | 5b40216e-5a01-557... | ef270cb-3017-5e7... | ef6e9001-1cf8-5d4... | 13:00:00 | true | true | Sale | f9141c66-c265-567... |
| 2023-01-15 | 6b49f211c19d81384... | b5083e10-2b61-5e8... | 79048bfd-6576-529... | 074b8859-147c-5c8... | 17:30:00 | true | true | Sale | e4a4338f3-4e77-55d... |
| 2022-01-01 | d66cc810339cb1829... | 7a290957-14c6-54c... | be975548-a354-55c... | 106fa726-a39c-541... | 13:00:00 | true | true | Sale | 9f39c33b-b4ca-512... |
| 2023-01-25 | 7caafdadb619d24ca... | 97eeebcd-7870-581... | 79048bfd-6576-529... | ef6e9001-1cf8-5d4... | 17:30:00 | true | true | Sale | 24bd99d1-f069-5f9... |
| 2023-01-03 | 30510e3426af310e5... | a9929e68-0084-598... | 79048bfd-6576-529... | 758be6b0-19d6-531... | 14:00:00 | true | true | Sale | 4aab6089-09a9-53e... |

TABELA V
KPI CALCULATION FACT TABLE

### C. Using Task Scheduler to Run Docker Compose

In order to run this project two times per day, it was used the Task Scheduler of Windows, with the following manner:

- Open Task Scheduler
- Press Win + R, type taskschd.msc, and hit Enter.
- Create a New Task
- In the right pane, click on Create Basic Task or Create Task.
  - Name Your Task
  - Provide a name and description for your task, then click Next.
  - Set the Trigger
  - Choose Daily and set the start time.
  - Check Repeat task every and set it to 12 hours or your desired interval, ensuring that the for a duration of option is set to Indefinitely.
- Configure the Action
  - Select Start a program and click Next.
  - In the Program/script field, enter the path to your Docker executable.
  - In the Add arguments (optional) field, enter the Docker Compose command you want to run. You need to specify both compose and the command you wish to execute, like up -d.

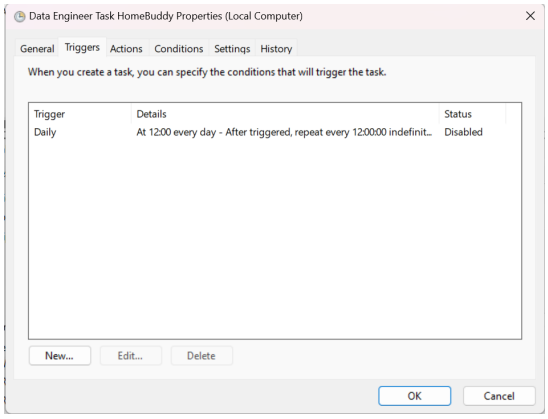


Fig. 3. Trigger of Task Scheduler for 2 times per day

## III. ANALYSIS

First of all, lets understand the overall feedback, taking into account the metrics defined to this exercise.

| Total Delivered Leads | Total Scheduled Appointments | Total Demos | Conversion Rate | Demo Conversion Rate |
|---|---|---|---|---|
| 82304 | 12260 | 8313 | 14.9 | 67.81 |

TABELA VI
OVERALL FEEDBACK

The total number of delivered leads stands at 82304, reflecting the effectiveness of marketing strategies in attracting

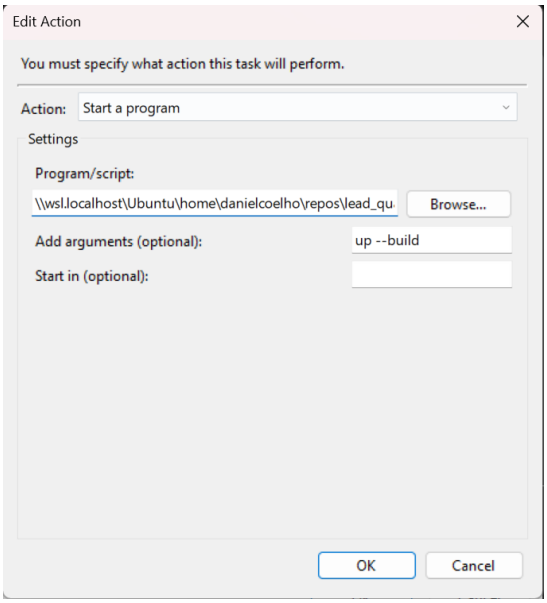


Fig. 4. Docker Compose Build command

potential customers. However, the number of appointments are not that high. Only 15% of that leads was successfully converted into actionable opportunities. This conversion rate suggests that approximately 1 in 7 leads are being converted into appointments, indicating a need for improvement in the lead qualification or nurturing process. On the other hand, the demo conversion rate of 67.81% indicates that, of those who attended a demo, this percentage was converted into further engagement.

The high number of total leads and a strong demo conversion rate suggest effective marketing efforts and engaging demos. However, the overall conversion rate from leads to appointments (14.9%) may indicate potential issues with the initial engagement strategies. By improving the quality of leads and enhancing follow-up processes, more appointments could be scheduled.

Strategically, it would be beneficial to analyze lead sources to identify which channels yield the highest conversion rates.

Lets check the lead sources to identify where we havee the highest and the lowest conversion rates.

TABELA VII
LEAD QUALITY ANALYSIS BY CITY AND STATE

| State | City Name | Total Delivered Leads | Total Scheduled Appointments | Total Demos | Conversion Rate | Demo Conversion Rate |
|---|---|---|---|---|---|---|
| NV | Las Vegas | 12464 | 1997 | 1375 | 16.02 | 68.85 |
| NULL | NULL | 7786 | 0 | 0 | 0.0 | 0.0 |
| NV | Henderson | 3583 | 731 | 582 | 20.4 | 79.62 |
| AZ | Tucson | 1821 | 496 | 384 | 27.24 | 77.42 |
| AZ | Phoenix | 1799 | 254 | 151 | 14.12 | 59.45 |
| AZ | Coronado/Tucson | 2 | 0 | 0 | 0.0 | 0.0 |
| OR | West Linn | 2 | 0 | 0 | 0.0 | 0.0 |
| WA | Malaga | 3 | 0 | 0 | 0.0 | 0.0 |
| IL | Northwoods | 3 | 0 | 0 | 0.0 | 0.0 |
| WA | Duwamish | 3 | 0 | 0 | 0.0 | 0.0 |

The data reveals that Las Vegas stands out as the top city with 12464 total delivered leads, indicating it as a major hub for lead generation in Nevada. It also has the highest number of scheduled appointments (1997) and demos (1375), leading to a conversion rate of 16.02% and a demo conversion rate of 68.85%. This suggests that the leads in Las Vegas are more engaged, resulting in higher demo success.

In contrast, several cities show minimal activity, with low total delivered leads. For instance, Coronado/Tucson, West Linn, Malaga, Northwoods, and Duwamish all have 2 or 3 total delivered leads, with zero scheduled appointments and demos, resulting in 0% conversion rates. This indicates a lack of lead engagement in these locations.

The presence of a row with NULL values for city and state but a count of 7786 total delivered leads suggests that there may be unidentified or unclassified leads in the dataset.

Overall, the data reveals significant disparities in engagement levels across different cities, which may necessitate targeted marketing strategies to improve lead generation and conversion rates in lower-performing regions.

| Year | Month | Total Delivered Leads | Total Scheduled Appointments | Total Demos | Conversion Rate (%) | Demo Conversion Rate (%) |
|---|---|---|---|---|---|---|
| 2022 | 11 | 392 | 392 | 280 | 100.0 | 71.43 |
| 2022 | 12 | 4259 | 3223 | 2337 | 75.68 | 72.51 |
| 2023 | 1 | 5489 | 3775 | 2635 | 68.77 | 69.80 |
| 2023 | 2 | 3806 | 2595 | 1753 | 68.18 | 67.55 |
| 2023 | 3 | 3477 | 2238 | 1308 | 64.37 | 58.45 |
| 2023 | 4 | 364 | 37 | 0 | 10.16 | 0.0 |
| NULL | NULL | 64517 | 0 | 0 | 0.0 | 0.0 |

TABELA VIII
LEAD QUALITY ANALYSIS BY YEAR AND MONTH

In January 2023, the month recorded the highest number of 5489 total delivered leads, leading to 3,775 scheduled appointments and 2635 demos. The conversion rate for this month was 68.77%, indicating a healthy engagement level with the leads generated.

As the months progressed, we see a slight decline in total delivered leads. February 2023 and March 2023 still maintained substantial lead counts of 3806 and 3477, respectively, with conversion rates of 68.18% and 64.37%. This suggests that the engagement with leads remained strong during these months.

In April 2023, the total delivered leads dropped significantly to 364, with only 37 scheduled appointments and no demos. The conversion rate of 10.16% reflects a notable decline in engagement. This decrease may warrant further investigation to understand potential causes, such as changes in marketing strategies or shifts in customer interest.

In contrast, November 2022 saw an unusual situation where the total leads (392) equaled the total scheduled appointments, leading to a 100% conversion rate. This indicates that all leads converted into appointments, with 280 demos scheduled, achieving a demo conversion rate of 71.43%. However, this performance could also indicate a small sample size for this month, potentially skewing the results.

The final row, containing NULL values, sums up to 64517 total delivered leads, but with no scheduled appointments or demos recorded. This discrepancy suggests an unidentified leads that need further classification.

| Job Status | Total Delivered Leads | Total Scheduled Appointments | Total Demos | Conversion Rate (%) | Demo Conversion Rate (%) |
|---|---|---|---|---|---|
| Mgmt Hold | 8 | 2 | 2 | 25.0 | 100.0 |
| Sub Completed | 21 | 15 | 15 | 71.43 | 100.0 |
| Customer Hold | 42 | 39 | 39 | 92.86 | 100.0 |
| Installed & Unpaid | 45 | 30 | 30 | 66.67 | 100.0 |
| Cancel After Rescission | 63 | 49 | 49 | 77.78 | 100.0 |
| Remediation Hold | 67 | 53 | 53 | 79.1 | 100.0 |
| Materials Ordered | 88 | 50 | 50 | 56.82 | 100.0 |
| Rel To Production | 114 | 87 | 87 | 76.32 | 100.0 |
| New | 124 | 111 | 111 | 89.52 | 100.0 |
| Ready to Schedule | 189 | 135 | 135 | 71.43 | 100.0 |
| Credit Decline | 217 | 156 | 156 | 71.89 | 100.0 |
| Scheduled | 234 | 146 | 146 | 62.39 | 100.0 |
| Cancel In Rescission | 264 | 185 | 185 | 70.08 | 100.0 |
| Completed | 1333 | 901 | 901 | 67.59 | 100.0 |
| NULL | 79495 | 10301 | 6354 | 12.96 | 61.68 |

TABELA IX
LEAD QUALITY ANALYSIS BY JOB STATUS

The job status ”Completed”has the highest number of total delivered leads at 1333, with 901 scheduled appointments and

901 demos. This results in a conversion rate of 67.59% and a demo conversion rate of 100%, indicating effective engagement and successful conversions from leads to scheduled appointments.

In contrast, the job status ”Mgmt Hold”shows the lowest total delivered leads at 8. Here, the conversion rate is significantly lower at 25.0%, although all scheduled appointments resulted in demos, as reflected in the 100% demo conversion rate. This suggests that while few leads reached this status, those that did were fully engaged when they were scheduled.

The ”Customer Hold”and ”Sub Completed”statuses demonstrate strong performance as well, with conversion rates of 92.86% and 71.43%, respectively, along with a full demo conversion rate.

Interestingly, the ”NULL”status indicates a large number of total delivered leads (79495), but with a low conversion rate of 12.96%. This may point to a data aggregation issue or a high number of leads that have not progressed further down the funnel.

### *A. Improvements*

#### *a.1) Proposed Architecture Description*

This architecture diagram illustrates a data pipeline on AWS specifically tailored to our project's requirements. The pipeline processes raw data residing in an S3 bucket, transforming it for subsequent analysis and visualization using AWS Glue, AWS Lambda, and AWS QuickSight. EventBridge Schedules triggers the data processing pipeline twice daily (cron expression: ”0 0 1,13 * * *”). An AWS Lambda function, activated by EventBridge, initiates the AWS Glue process twice per day, ensuring regular data extraction, manipulation, and datamodel updates. To optimize performance, the extraction process employs incremental loads, processing only new data with each run rather than reprocessing the entire dataset. AWS Glue Catalog stores metadata about the data within the S3 buckets, encompassing schema information. AWS QuickSight serves as a business intelligence tool, connecting to the AWS Glue Data Catalog to access the transformed data and generate insightful visualizations and dashboards. Additionally, AWS Athena can be leveraged as a serverless query engine to delve deeper into the output data by executing standard SQL queries against the data stored in S3.

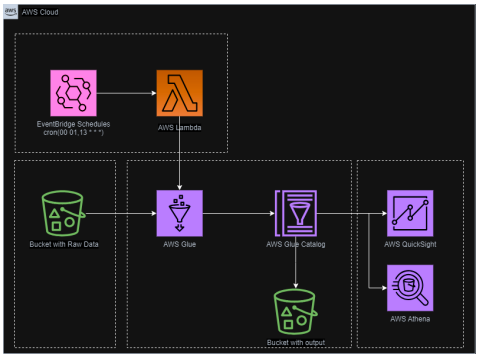


Fig. 5. Project Setup Idea