# CLOUD ANALYTICS AND DATA WAREHOUSE IMPLEMENTATION FOR A HISTORICAL DATASET

## Group Project ( Team - 8 )

**Shashank Reddy Kandimalla (016799523), Rohith Reddy Vangala (016762109), Sakshi Manish Mukkirwar (016794765), Swati (016702413), Yamini Muthyala (016766165)**

**ABSTRACT** - The primary objective of this project is to analyze historical vehicle insurance data and predict customer interest in purchasing vehicle insurance from a health insurance provider. The project involves the implementation of a cloud-based data warehouse solution to store and process the historical data. The data is extracted, transformed, and loaded (ETL)[3] using Python scripts, orchestrated by Apache Airflow on Google Cloud Composer. The processed data is then stored in a private VPC network on AWS[5], utilizing a scalable Amazon Redshift database instance. A variety of queries are executed to gain insights into customer behavior and preferences. The abstract highlights a few important queries, such as segmenting customers by age group, analyzing the relationship between region code and interest in vehicle insurance, identifying previously insured customers, and examining the impact of vehicle damage on insurance interest. Additional queries and their results are discussed in a separate section of the report.

The insights derived from these queries are used to inform targeted marketing efforts and identify potential areas of growth for the insurance provider. Key Performance Metrics (KPIs) are extracted using business intelligence techniques and displayed on a Microsoft Power BI and Tableau dashboard, providing a comprehensive view of the historical data and trends.

**KEYWORDS** - cloud analytics, data warehouse, mysql, redshift, extract, transform, load (etl), cloud composer, apache airflow, dag, aws vpc, s3, power bi, tableau, key performance metrics (kpi), historical data, vehicle insurance, premium, policyholders, health insurance, business intelligence, visualization, query optimization, performance tuning, data partitioning, data indexing.

### 1. INTRODUCTION

The insurance industry has been undergoing a rapid transformation with the advent of digital technology, data analytics, and cloud computing. Traditional methods of predicting customer behavior and preferences are no longer adequate in the face of the ever-growing volume and complexity of data generated by customers and their interactions with various insurance products. In this context, the effective use of historical data to predict customer interest in purchasing vehicle insurance has become a critical factor in the success of insurance providers.

This report presents a project that aims to analyze historical vehicle insurance data and predict customer interest in purchasing vehicle insurance from a health insurance provider. The project leverages the power of cloud-based data warehousing, Extract, Transform, Load (ETL) processes, and advanced analytics to process and understand the historical data. The data is stored and processed using a cloud-based infrastructure, specifically a private VPC network on AWS[5] with a scalable Amazon Redshift database instance. The project employs Python scripts for ETL[3] processes, orchestrated by Apache Airflow on Google Cloud Composer. This setup allows for efficient processing and transformation of the raw data into a structured format suitable for analysis. Various queries are executed to gain insights into customer behavior and preferences, which are then visualized using Microsoft Power BI and Tableau dashboards. The insights derived from these analyses inform targeted marketing efforts and help identify potential areas of growth for the insurance provider.

This report details the project's methodology, the data processing pipeline, the queries executed, and the insights derived from the analyses. The final section presents the data visualization and discusses the implications of the findings for the insurance industry, emphasizing the value of cloud-based analytics and data warehousing in processing and understanding historical data.

# CLOUD ANALYTICS AND DATA WAREHOUSE IMPLEMENTATION FOR A HISTORICAL DATASET

## 2. PROBLEM STATEMENT

The insurance industry faces the challenge of understanding customer preferences and predicting their interest in purchasing various insurance products, such as vehicle insurance. Traditional methods of customer analysis often fall short in effectively utilizing the vast amounts of historical data available. As a result, insurance providers struggle to tailor their marketing efforts and identify potential growth areas.

The primary objective of this project is to develop a data-driven solution that can analyze historical vehicle insurance data and predict customer interest in purchasing vehicle insurance from a health insurance provider. By leveraging advanced data analytics, cloud-based data warehousing, and visualization techniques, the project aims to deliver valuable insights into customer behavior and preferences, which can be used to inform targeted marketing strategies and maximize business growth opportunities.

## 3. PROPOSED SOLUTION

The proposed solution for predicting customer interest in vehicle insurance and analyzing historical data involves the following components:

**1. Cloud-based Data Warehouse:** Implement a cloud-based data warehouse solution using a private VPC network on AWS with a scalable Amazon Redshift database instance. This setup allows for efficient storage and processing of large volumes of historical data.

**2. ETL Processes:** Develop and orchestrate Python scripts for Extract, Transform, Load (ETL) processes using Apache Airflow[8] on Google Cloud Composer[4]. This enables the extraction of data from the source, transformation into a structured format, and loading it into the data warehouse for further analysis.

**3. Data Analysis and Queries:** Execute various SQL queries on the processed data to gain insights into customer behavior and preferences, such as segmenting customers by age group, analyzing the relationship between region code and interest in vehicle insurance, and examining the impact of vehicle damage on insurance interest.

**4. Business Intelligence:** Extract Key Performance Metrics (KPIs) from the analyzed data using business intelligence techniques, which can be used to inform targeted marketing efforts and identify potential growth areas for the insurance provider.

**5. Data Visualization:** Display the results of the data analysis and KPIs on a Microsoft Power BI or Tableau dashboard, providing a comprehensive and easily understandable view of the historical data and trends.

The proposed solution harnesses the power of cloud computing, data analytics, and visualization techniques to deliver actionable insights into customer preferences and predict their interest in vehicle insurance. This approach enables insurance providers to tailor their marketing strategies effectively and capitalize on growth opportunities in the industry.

## 4. DATASET AND PRE-PROCESSING

The dataset[1] used in this project is obtained from Kaggle and contains information about policyholders, their vehicles, and their insurance policies. The dataset is stored as a CSV file and consists of various attributes, such as unique customer identifiers, gender, age, driving license status, region code, vehicle details, and policy-related information.

The processing of the dataset involves several steps, beginning with loading the CSV file from an Amazon S3 bucket. The data is then split into three separate tables: customers, vehicles, and policies. Each table undergoes specific preprocessing tasks, such as converting gender to binary values, filling in missing values, and converting columns to appropriate data types.

Once the data is preprocessed, it is uploaded back to the S3 bucket as individual CSV files for further analysis. By following this approach, the dataset is transformed and organized in a way that allows for efficient analysis, visualization, and querying.

## 5. CLOUD ARCHITECTURE

In this project, a cloud-based architecture is employed to process, analyze, and visualize the insurance dataset. The key components of this architecture are:

**1. Amazon S3**: The dataset is initially stored in an Amazon S3 bucket. After preprocessing, the cleaned and split data is

also stored back in the S3 bucket as separate CSV files for customers, vehicles, and policies.

**2. Google Cloud Composer**: The ETL (Extract, Transform, Load) process is orchestrated using Google Cloud Composer[4], which utilizes Apache Airflow[8] to create and manage workflows for data processing.

**3. Apache Airflow**: The ETL pipeline is built using Apache Airflow, which defines and orchestrates the tasks for loading, preprocessing, and splitting the dataset. A Directed Acyclic Graph (DAG) script is used to design and schedule the workflow.

**4. Amazon RDS**: After processing the data, it is stored in an Amazon RDS (Relational Database Service) instance using a scalable and secure cloud SQL database, such as Amazon Redshift.

**5. Amazon Redshift**: The processed data is transferred to Amazon Redshift for analytics and querying. The results of these queries are used for generating insights and identifying trends in the data.

**6. Amazon Glue:** The data is transferred from Amazon S3 to Amazon Redshift using Amazon Glue, which is a fully managed extract, transform, and load (ETL) service that makes it easy to move data between data stores.

**7. Tableau and Power BI**: The analyzed data is visualized using Tableau and Power BI, which connect to the Amazon Redshift instance. These business intelligence tools allow for the creation of interactive dashboards and reports that highlight key performance metrics and trends in the dataset.

By leveraging this cloud-based architecture, the project benefits from increased scalability, flexibility, and efficiency in processing and analyzing the insurance dataset.

## 6. AMAZON VIRTUAL PRIVATE CLOUD

Amazon Virtual Private Cloud (VPC) is a private virtual network hosted in the AWS cloud[5]. It provides a secure and isolated environment for your cloud resources, allowing you to launch AWS resources into a virtual network that you define.
Using a private VPC network in Amazon is an effective way to ensure the security and privacy of your data. With a private VPC network, you can isolate your resources from the public internet and create custom network topologies, giving you complete control over how your resources are exposed to the internet.
In our historical vehicle insurance data project, we have used a private VPC network to ensure the security and privacy of our data. By setting up our cloud-based database instance in a private VPC network, we are able to limit access to our database instance to only authorized users and applications, while also ensuring that all data transfer between our cloud resources is encrypted.

Overall, using a private VPC network in Amazon is an effective way to ensure the security and privacy of your data in a cloud-based environment, and is a key component of our data analytics solution.

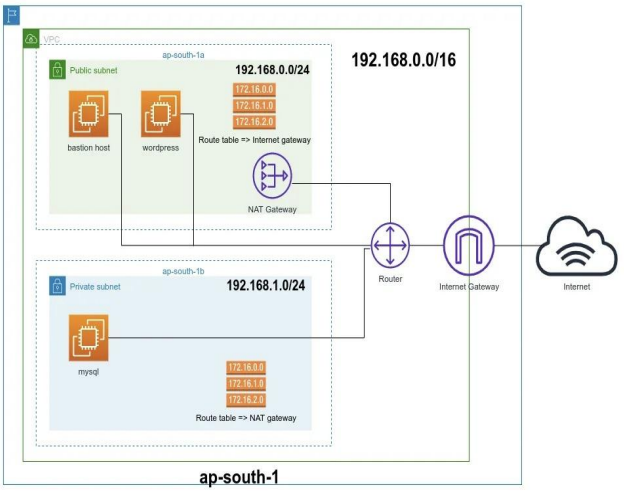


Fig-1 VPC Architecture

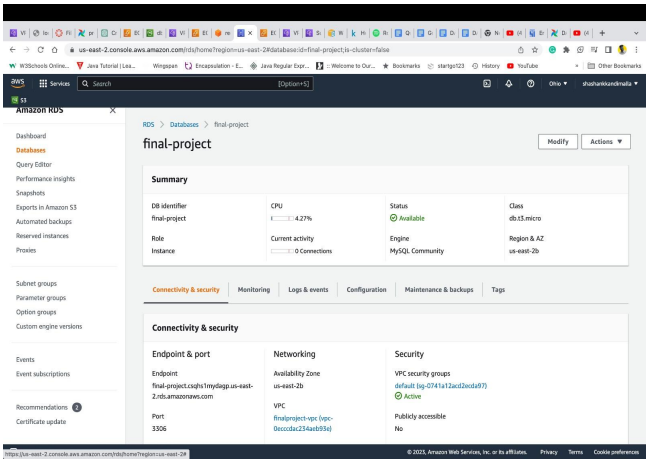


Fig-1.1 Private VPC

# CLOUD ANALYTICS AND DATA WAREHOUSE IMPLEMENTATION FOR A HISTORICAL DATASET

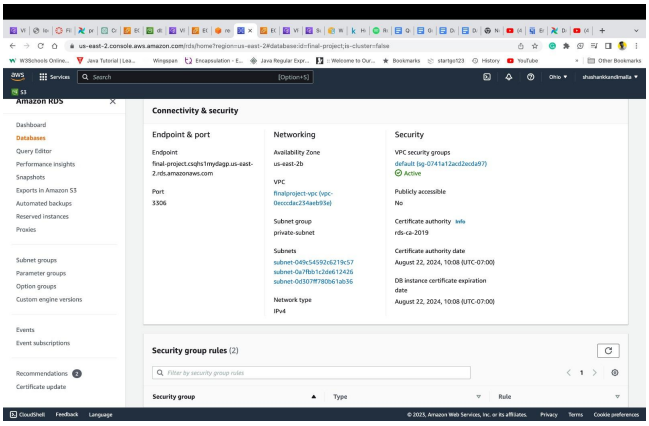


Fig-1.2 private VPC

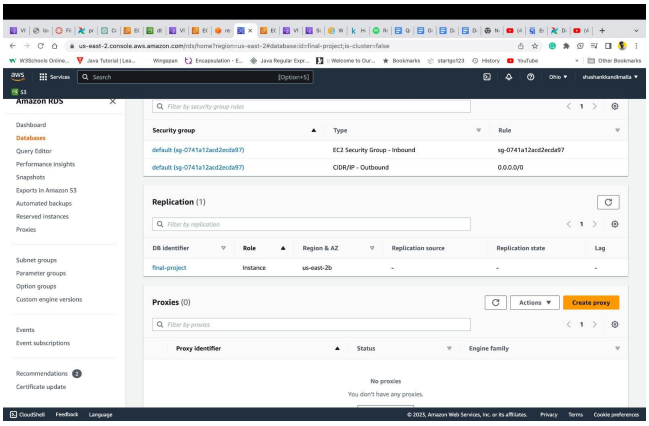


Fig-1.3 private VPC

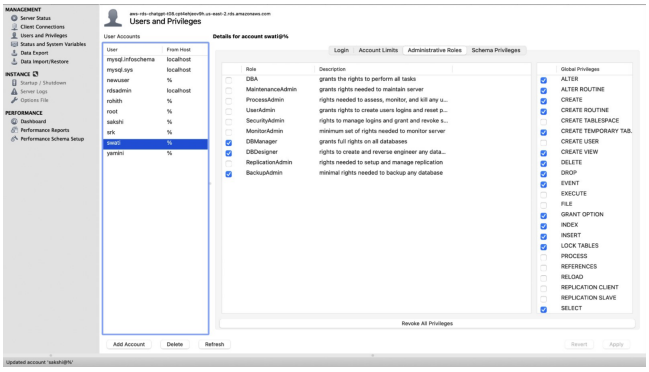


Fig-1.4 Access privileges

## 7. CLOUD SQL

A Cloud SQL database instance is used to store and manage the processed insurance dataset. The database is configured within a private VPC (Virtual Private Cloud) network, ensuring a secure and isolated environment for data storage and management. This setup provides enhanced security and allows for strict control over data access.

Amazon RDS, a cloud-based data warehouse, is employed as the primary database solution for the project. RDS offers scalability, high performance, and compatibility with various data processing tools and platforms. It is designed for handling large-scale data workloads and supports SQL querying for data analytics and insights.

By using a Cloud SQL database instance, such as Amazon RDS, the project benefits from a scalable, secure, and high-performance storage solution, which is essential for handling large volumes of data and performing complex analytics.

## 8. FIOW CHART

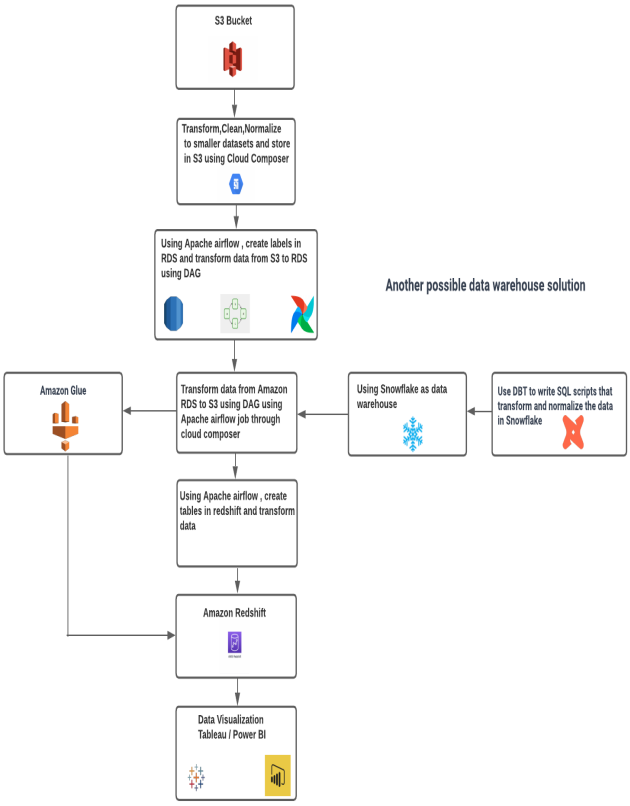


Fig-2 Flow chart

The project starts with loading the dataset into an S3 bucket, which is then transformed, cleaned, and normalized using Google Cloud Composer and Apache Airflow. The normalized data is then loaded into a relational database service (RDS) instance using DAGs.

# CLOUD ANALYTICS AND DATA WAREHOUSE IMPLEMENTATION FOR A HISTORICAL DATASET

Next, Amazon Glue is utilized to further transform and prepare the data for analysis. The transformed data is then transferred back to S3 using the Apache Airflow job. After that, a table is created in Amazon Redshift using Apache Airflow, and the data is transferred from S3 to Redshift.
Finally, the data is analyzed using SQL queries, and key performance metrics (KPIs) are extracted and visualized using Tableau and Power BI. This entire process is designed to provide a comprehensive and accurate understanding of historical data trends, enabling the identification of suitable entities from the data.

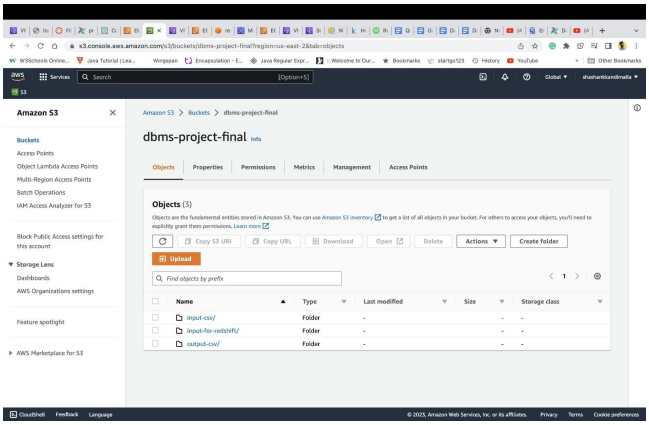


Fig-3 S3 bucket

## 9. ETL PROCESS

The ETL (Extract, Transform, Load) process in this project consists of the following steps:

**1. Extract:** The insurance dataset is extracted from an Amazon S3 bucket as a CSV file. The file contains information about policyholders, their vehicles, and their insurance policies.

**2. Transform:** The data is transformed through several preprocessing steps. This includes splitting the dataset into three separate tables: customers, vehicles, and policies. Each table undergoes specific transformations, such as converting gender to binary values, filling in missing values, and converting columns to appropriate data types. This ensures that the data is clean, consistent, and ready for analysis.

**3. Load:** After the transformation process, the cleaned and organized data is loaded into a scalable and secure cloud SQL database, such as Amazon Redshift. This database serves as the foundation for further analytics and querying.

The ETL process is orchestrated using Google Cloud Composer and Apache Airflow, which define and manage workflows for data processing. A Directed Acyclic Graph (DAG) script is used to design and schedule the ETL workflow, allowing for efficient and automated data processing.
By following this ETL process, the insurance dataset is transformed from its raw state into a structured and organized format, making it suitable for analytics and visualization.

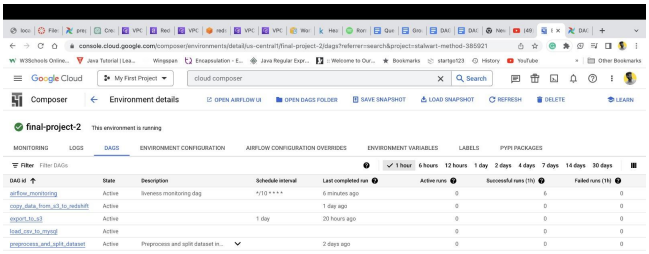


Fig-4 Cloud composer showing DAGS

## 10. CLOUD COMPOSER HOSTING.

In our project, we have utilized the power of Cloud Composer for efficient orchestration of the ETL pipeline. The Cloud Composer is hosted on a Kubernetes Engine cluster, which provides a container-based platform for executing the workflows. This enables horizontal and vertical scaling of the cluster, allowing for efficient resource allocation and better performance. Additionally, Kubernetes Engine provides built-in support for auto-scaling, which ensures that the cluster can automatically scale up or down based on the workload. This helps to optimize the usage of resources and ensures that the workflows are executed in a timely and efficient manner.

Overall, the use of a container-based cluster for hosting our Cloud Composer instance has been crucial in ensuring that our data processing pipeline is scalable, reliable, and

efficient, making it an essential component of our cloud-based analytics and data warehousing solution.

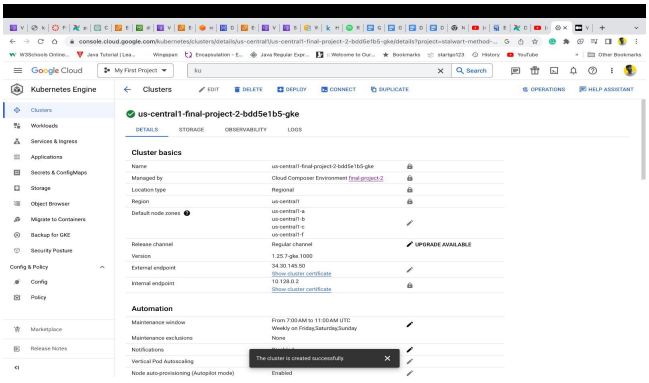


Fig-5 Cloud composer hosting

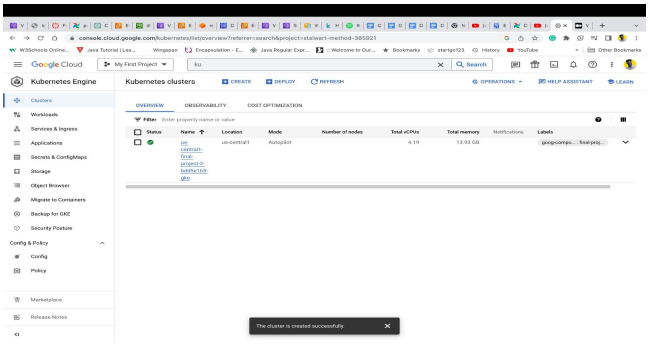


Fig-5.1 Cloud composer hosting

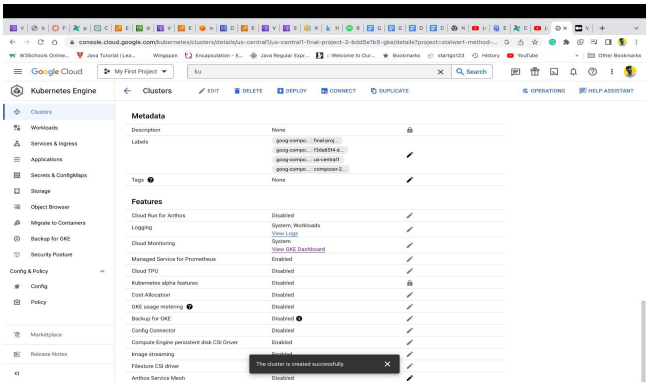


Fig-5.2 Cloud composer hosting

## 11. AIRFLOW PIPELINE

The Airflow pipeline in this project is responsible for orchestrating the entire ETL process, automating the extraction, transformation, and loading of the insurance dataset. Apache Airflow, an open-source platform, is used for managing and scheduling complex data workflows. The pipeline consists of the following components:

**1. DAG (Directed Acyclic Graph):** The DAG script is a crucial part of the Airflow pipeline (all the dags are presented at the end of the report), as it defines the workflow's tasks, dependencies, and execution schedules. In this project, the DAG script outlines the steps for preprocessing the insurance dataset, splitting it into separate tables, and loading the data into Amazon Redshift.

**2. Tasks:** The tasks within the DAG script represent individual operations in the ETL process. In this project, a PythonOperator task is used to preprocess the dataset, which involves reading the data from an Amazon S3 bucket, transforming it, and storing the cleaned data in separate CSV files.

**3. Orchestration:** Apache Airflow manages the orchestration of tasks, ensuring that they are executed in the correct order based on their dependencies. This allows for an efficient and automated ETL process, minimizing the risk of errors and reducing manual intervention.

**4. Execution:** The Airflow pipeline is executed using Google Cloud Composer, a managed Apache Airflow service. This enables the project to leverage cloud-based resources for scalable and reliable pipeline execution. Cloud Composer is hosted on a container-based cluster, allowing for auto-scaling both horizontally and vertically.

By utilizing an Airflow pipeline, the project can streamline and automate the ETL process, ensuring that the insurance dataset is consistently processed, transformed, and loaded into the cloud SQL database for further analysis and visualization.

## 12. PERFORMANCE OPTIMIZATION

Performance optimization is an important aspect of any data analytics project, and our cloud-based solution for historical vehicle insurance data is no exception. In order to ensure optimal performance and scalability, we have implemented several key strategies:

**1. Use of Cloud Resources:** By leveraging cloud resources such as Amazon Redshift and Google BigQuery, we are able to take advantage of their distributed computing power

and scale up or down as needed to handle larger or smaller datasets. This allows us to achieve high levels of performance and scalability without the need for significant upfront investments in hardware or infrastructure.

**2. Data Partitioning:** We have partitioned our data into smaller chunks to make it easier to handle and process. This enables us to distribute the data across multiple nodes, reducing the load on individual nodes and improving overall performance.

**3. Indexing:** We have implemented indexing on key columns in our database to speed up query performance. By indexing frequently queried columns, we can significantly reduce the time it takes to retrieve data and generate reports.

**4. Query Optimization:** We have optimized our SQL queries to ensure that they are as efficient as possible. This includes avoiding unnecessary joins or subqueries, optimizing the order of operations, and minimizing data transfers between nodes.

**5. Monitoring and Tuning:** Finally, we continuously monitor and tune our system to ensure optimal performance. This includes monitoring query execution times, system resource usage, and network performance, and adjusting our system configurations as needed to optimize performance.

By implementing these strategies, we have been able to achieve high levels of performance and scalability for our cloud-based data analytics solution, enabling us to handle large and complex datasets with ease.

## 13. QUERIES

**1. Age Group Wise Analysis of Customers' Interest in Vehicle Insurance:**

**Query :**

```
SELECT
    CASE
      WHEN Age BETWEEN 20 AND 30 THEN '20-30'
      WHEN Age BETWEEN 31 AND 40 THEN '31-40'
      WHEN Age BETWEEN 41 AND 50 THEN '41-50'
      WHEN Age BETWEEN 51 AND 60 THEN '51-60'
      ELSE '60+'
    END AS Age_Group,
    COUNT(*) AS Total_Customers,
    SUM(Response) AS Interested_In_Vehicle_Insurance
FROM "dbms".Customers
GROUP BY Age_Group order by
Interested_In_Vehicle_Insurance desc ;
```

**Interaction with database:** The query selects data from the "Customers" table in the "dbms" database and groups the customers by age range, calculated using a CASE statement. It then counts the total number of customers in each age range and calculates the sum of their responses indicating interest in vehicle insurance. The data is then sorted by the count of interested customers in descending order.

| age_group | total_customers | interested_in_vehicle_i... |
|---|---|---|
| 41-50 | 101475 | 16025 |
| 31-40 | 73498 | 11597 |
| 51-60 | 60288 | 7716 |
| 20-30 | 215338 | 7183 |
| 60+ | 57547 | 4189 |

Fig-6 Output for Age Group Wise Analysis of Customers' Interest in Vehicle Insurance

**2. Number of Customers Interested in Vehicle Insurance by Region Code:**

**Query:**

```
SELECT Region_Code,
    COUNT(*) AS Total_Customers,
    SUM(Response) AS Interested_In_Vehicle_Insurance
FROM "dbms".Customers
GROUP BY Region_Code order by
Interested_In_Vehicle_Insurance desc;
```

**Interaction with database:** The query is executed on the "dbms" database and selects data from the "Customers" table. It groups the customers by region code and calculates the count and sum of their responses. The query then sorts the data in descending order based on the count of interested customers.

| region_code | total_customers | interested_in_vehicle_i... |
|---|---|---|
| 28 | 141937 | 19917 |
| 8 | 44900 | 3257 |
| 41 | 24400 | 2224 |
| 46 | 26357 | 2032 |
| 29 | 14843 | 1365 |
| 3 | 12349 | 1181 |
| 11 | 12328 | 1041 |
| 15 | 17750 | 958 |

Fig-7 Output for number of customers interested in vehicle insurance by region code

**3. Total number of previously insured customers interested in vehicle insurance:**

**Query:**

```
SELECT COUNT(*) AS Total_Customers,
    SUM(Response) AS Interested_In_Vehicle_Insurance
FROM "dbms".Customers
WHERE Previously_Insured = 1;
```

**Interaction with database:** The query will be sent to the "dbms" database and executed. The database will scan the "Customers" table to identify the customers who have previously purchased insurance and match the criteria. It will then count the total number of such customers and calculate the sum of their responses indicating interest in vehicle insurance.

| total_customers | interested_in_vehicle_i... |
|---|---|
| 233070 | 158 |

Fig-8 Output for Total number of previously insured customers interested in vehicle insurance.

**4. Count of Customers and Their Interest in Vehicle Insurance by Vehicle Age:**

**Query:**

```
SELECT Vehicle_Age,
    COUNT(*) AS Total_Customers,
    SUM(Response) AS Interested_In_Vehicle_Insurance
FROM "dbms".Customers
INNER JOIN "dbms".Vehicles ON Customers.id =
Vehicles.id
GROUP BY Vehicle_Age;
```

**Interaction with database:** The query fetches the required data by joining the "Customers" and "Vehicles" tables in the "dbms" database using the "INNER JOIN" clause. It groups the data based on the "Vehicle_Age" column and calculates the count of customers and their interest in vehicle insurance for each age group using the "COUNT" and "SUM" aggregate functions, respectively. The query returns the result set showing the count of customers and their interest in vehicle insurance for each vehicle age group.

| vehicle_age | total_customers | interested_in_vehicle_i... |
|---|---|---|
| 2 | 21326 | 4702 |
| 0 | 219805 | 7202 |
| 1 | 267015 | 34806 |

Fig-9 Output for count of customers and their interest in vehicle insurance by vehicle age..

**5. Grouping customers by premium group and calculating the count of total customers and sum of responses interested in vehicle insurance:**

**Query:**

```
SELECT CASE
    WHEN p.Annual_Premium < 50000 THEN 'Less than
50k'
    WHEN p.Annual_Premium BETWEEN 50000 AND
100000 THEN '50k-100k'
    ELSE 'More than 100k'
  END AS Premium_Group,
  COUNT(*) AS Total_Customers,
  SUM(c.Response) AS Interested_In_Vehicle_Insurance
FROM "dbms".Policies
p inner join "dbms".customers c on p.id=c.id
GROUP BY Premium_Group;
```

**Interaction with database:** The above SQL query uses a JOIN statement to combine data from the "Policies" and "Customers" tables in the "dbms" database. It calculates the premium group based on the value of the Annual_Premium column in the Policies table, and groups the customers by the premium group. It then counts the total number of customers in each premium group and calculates the sum of their responses indicating interest in vehicle insurance. The result is sorted by the Premium_Group column.

| total_customers | interested_in_vehicle_i... | premium_group |
|---|---|---|
| 465146 | 41811 | Less than 50k |
| 1049 | 123 | More than 100k |
| 41951 | 4776 | 50k-100k |

Fig-10 Output for Grouping customers by premium group and calculating the count of total customers and sum of responses interested in vehicle insurance.

# CLOUD ANALYTICS AND DATA WAREHOUSE IMPLEMENTATION FOR A HISTORICAL DATASET

**6. Identifying customers who have a high policy sales channel value and target them with vehicle insurance offers:**

**Query:**

```
SELECT p.policy_sales_channel,COUNT(*) AS
Total_Customers
,SUM(Response) AS Interested_In_Vehicle_Insurance
FROM "dbms".Policies p inner join "dbms".customers c on
p.id=c.id
GROUP BY p.policy_sales_channel order by
Interested_In_Vehicle_Insurance desc ;
```

**Interaction with database:** The query interacts with the Policies and Customers tables in the dbms database using an inner join to combine the data based on matching id values. The resulting data is grouped by policy_sales_channel, and the query counts the total number of customers for each group and calculates the sum of their responses indicating interest in vehicle insurance. The resulting data is sorted in descending order based on the count of interested customers.

| total_customers | interested_in_vehicle_i... | policy_sales_channel |
|---|---|---|
| 106594 | 15891 | 26 |
| 98299 | 13996 | 124 |
| 179523 | 3858 | 152 |
| 14313 | 2297 | 156 |
| 8958 | 1794 | 157 |
| 13239 | 1720 | 122 |
| 7988 | 1474 | 154 |
| 3850 | 880 | 163 |

Fig-11 Output for Identifying customers who have a high policy sales channel value and target them with vehicle insurance offers.

**7. Total Customers and Interested Customers in Vehicle Insurance by Vintage:**

**Query:**

```
SELECT Vintage,
    COUNT(*) AS Total_Customers,
    SUM(Response) AS Interested_In_Vehicle_Insurance
FROM "dbms".Policies p inner join "dbms".customers c on
p.id=c.id
GROUP BY p.Vintage order by
Interested_In_Vehicle_Insurance  desc;
```

**Interaction with database:** The query runs a join between the "Policies" and "Customers" tables in the "dbms" database using the customer ID as the join key. It groups the data by vintage, counts the total number of customers in each group, and calculates the sum of their responses indicating interest in vehicle insurance. Finally, the data is sorted in descending order based on the count of interested customers.

| total_customers | interested_in_vehicle_i... | vintage |
|---|---|---|
| 1828 | 192 | 84 |
| 1817 | 191 | 11 |
| 1795 | 190 | 189 |
| 1773 | 190 | 34 |
| 1821 | 190 | 282 |
| 1823 | 189 | 165 |
| 1697 | 186 | 220 |
| 1789 | 186 | 298 |

Fig-12 Output for Total Customers and Interested Customers in Vehicle Insurance by Vintage.

**8. Analysis of Customers' Interest in Vehicle Insurance based on Vehicle Damage:**

**Query:**

```
SELECT Vehicle_Damage,
    COUNT(*) AS Total_Customers,
    SUM(Response) AS Interested_In_Vehicle_Insurance
FROM "dbms".Customers
INNER JOIN "dbms".Vehicles ON Customers.id =
Vehicles.id
GROUP BY Vehicle_Damage;
```

**Interaction with database:** The query performs an inner join on the "Customers" and "Vehicles" tables using the "id" column. It then groups the data by the "Vehicle_Damage" column from the "Vehicles" table, and calculates the count of customers and the sum of their responses for each group. The data is then sorted by the count of interested customers in descending order.

| total_customers | interested_in_vehicle_i... | vehicle_damage |
|---|---|---|
| 256248 | 45728 | 1 |
| 251898 | 982 | 0 |

Fig-13 Output for Analysis of Customers' Interest in Vehicle Insurance based on Vehicle Damage.

**9. Identifying customers who have a high annual premium but have not yet purchased vehicle insurance:**

**Query:**

```
SELECT Customers.id,
      Policies.Annual_Premium,
      Vehicles.Vehicle_Age,
      Vehicles.Vehicle_Damage
FROM "dbms".Customers
INNER JOIN "dbms".Policies ON Customers.id =
Policies.id
INNER JOIN "dbms".Vehicles ON Customers.id =
Vehicles.id
WHERE Customers.Response = 0
  AND Policies.Annual_Premium > 100000 ;
```

**Interaction with database:** The query interacts with the "Customers", "Policies", and "Vehicles" tables in the "dbms" database, performing inner joins to match records based on their ID. It then applies filters to the records to retrieve only those where the customer did not respond and the annual premium for their policy is greater than 100,000. The resulting data is then returned, including the customer ID, annual premium, vehicle age, and vehicle damage status.

| id | annual_premium | vehicle_age | vehicle_damage |
|---|---|---|---|
| 481 | 104002 | 2 | 1 |
| 568 | 112974 | 1 | 1 |
| 1141 | 139130 | 1 | 0 |
| 2229 | 125643 | 1 | 0 |
| 3912 | 117799 | 1 | 1 |
| 4798 | 133098 | 1 | 0 |
| 4889 | 103026 | 1 | 1 |
| 5422 | 131469 | 0 | 0 |

Fig-14 Output for Identifying customers who have a high annual premium but have not yet purchased vehicle insurance.

**10. Retrieving the average annual premium for policies associated with vehicles that have a specified vehicle age:**

**Query:**

```
SELECT AVG(Annual_Premium) AS
Avg_Annual_Premium
FROM "dbms".Policies
WHERE id IN (
  SELECT id
  FROM "dbms".Vehicles
  WHERE Vehicle_Age IS NOT NULL
);
```

**Interaction with database:** In this query, we select the average of the Annual_Premium attribute from the Policies table in the "dbms" database. We then use a subquery to select the IDs of all vehicles from the Vehicles table that have a non-null value for the Vehicle_Age attribute. We use the IN keyword to filter the policies based on those IDs, and calculate the average annual premium for those policies.

| avg_annual_premium |
|---|
| 30554.45304105513 |

Fig-15 Output for Retrieving the average annual premium for policies associated with vehicles that have a specified vehicle age.

**11. Grouping Customers by Age and Gender:**

**Query:**

```
SELECT Gender,
      CASE
        WHEN Age BETWEEN 18 AND 25 THEN '18-25'
        WHEN Age BETWEEN 26 AND 35 THEN '26-35'
        WHEN Age BETWEEN 36 AND 45 THEN '36-45'
        WHEN Age BETWEEN 46 AND 55 THEN '46-55'
        WHEN Age BETWEEN 56 AND 65 THEN '56-65'
        ELSE '65+'
      END AS Age_Group,
      COUNT(DISTINCT id) AS Num_Customers
FROM "dbms".Customers
WHERE id IN (
  SELECT id
  FROM "dbms".Vehicles
  WHERE Vehicle_Age IS NOT NULL
)
GROUP BY Gender, Age_Group;
```

**Interaction with database:** In this interaction, we are selecting the gender, age group (based on the age of customers), and the number of distinct customers in each group. We are joining the Customers and Vehicles tables on id and filtering out customers who don't have a vehicle age. We group the results by gender and age group and return the output. The output shows the number of customers in each age group for each gender.

# CLOUD ANALYTICS AND DATA WAREHOUSE IMPLEMENTATION FOR A HISTORICAL DATASET

| gender | age_group | num_customers |
|---|---|---|
| 1 | 36-45 | 55592 |
| 1 | 46-55 | 53118 |
| 0 | 18-25 | 85724 |
| 0 | 56-65 | 15863 |
| 0 | 46-55 | 28855 |
| 1 | 65+ | 21764 |
| 1 | 56-65 | 29674 |
| 0 | 26-35 | 49286 |

Fig-16 Output for grouping customers by age and gender.

**12. Getting the average annual premium of customers who responded positively to the vehicle insurance policy and whose vehicle age is known:**

**Query:**

```
SELECT AVG(p.Annual_Premium) AS
Average_Annual_Premium
FROM "dbms".Customers c
INNER JOIN "dbms".Policies p ON c.id = p.id
INNER JOIN "dbms".Vehicles v ON c.id = v.id
WHERE c.Response = 1 AND v.Vehicle_Age IS NOT
NULL;
```

**Interaction with database:** This SQL query interacts with the database to calculate the average annual premium for customers who responded positively to the vehicle insurance offer and have a non-null vehicle age. The query uses inner joins to combine data from the Customers, Policies, and Vehicles tables and applies filters to retrieve only the relevant data. The result is a single value representing the average annual premium.

| average_annual_premi… |
|---|
| 31604.092742453435 |

Fig-17 Output for the average annual premium of customers who responded positively to the vehicle insurance policy and whose vehicle age is known.

**13. Calculating Average Annual Premiums by Vehicle Age:**

**Query:**

```
SELECT Vehicle_Age, AVG(Annual_Premium) AS
Average_Annual_Premium
FROM "dbms".Policies p INNER JOIN "dbms".Vehicles v
ON p.id = v.id
WHERE Vehicle_Age IS NOT NULL
GROUP BY Vehicle_Age;
```

**Interaction with database:** The query retrieves data from two tables ("Policies" and "Vehicles") in the "dbms" database, and performs an inner join to match records based on their ID. It then filters records where the vehicle age is not null and groups the results by vehicle age to calculate the average annual premium for each age group.

| vehicle_age | average_annual_premi… |
|---|---|
| 2 | 35619.1395010785 |
| 0 | 30110.78465003071 |
| 1 | 30515.170705765595 |

Fig-18 Output for calculating average annual premiums by vehicle.

**14. Finding the Policy Sales Channel with the Highest Average Annual Premium:**

**Query:**

```
SELECT Policy_Sales_Channel, AVG(Annual_Premium)
AS Average_Annual_Premium
FROM "dbms".Policies
GROUP BY Policy_Sales_Channel
ORDER BY Average_Annual_Premium DESC
LIMIT 1;
```

**Interaction with database:** The query retrieves data from the "Policies" table in the "dbms" database and groups the results by policy sales channel to calculate the average annual premium for each channel. It then sorts the results in descending order by the average annual premium and limits the output to the top result (i.e., the policy sales channel with the highest average annual premium).

| average_annual_premi… | policy_sales_channel |
|---|---|
| 60095 | 74 |

Fig-19 Output showing the Policy Sales Channel with the Highest Average Annual Premium

**15. Count of Vehicles by Damage Status:**

**Query:**

```
SELECT Vehicle_Damage, COUNT(*) AS count
FROM dbms.Vehicles
GROUP BY Vehicle_Damage;
```

**Interaction with database:** The query is executed by accessing the "Vehicles" table in the "dbms" database and performing a grouping operation based on the

"Vehicle_Damage" column. The "COUNT" function is used to count the number of vehicles with each damage status. The result is a table with two columns: "Vehicle_Damage" and "count".

| vehicle_damage | count |
|---|---|
| 0 | 251898 |
| 1 | 256248 |

Fig-20 Output for Count of Vehicles by Damage Status.

**16. Retrieving Maximum, Minimum, and Average Annual Premiums for Customers who Responded and have a Vehicle with Damage:**

**Query:**

```
SELECT MAX(p.Annual_Premium) AS max_premium,
MIN(p.Annual_Premium) AS min_premium,
AVG(p.Annual_Premium) AS avg_premium
FROM dbms.Customers c
INNER JOIN dbms.Policies p ON c.id = p.id
INNER JOIN dbms.Vehicles v ON c.id = v.id
WHERE c.Response = 1 AND v.Vehicle_Damage = 1;
```

**Interaction with database:** The database is queried to retrieve data from three tables using an inner join. The query filters records based on two conditions, where the customer responded and their vehicle has damage. The MAX, MIN, and AVG functions are then used to calculate the maximum, minimum, and average annual premiums for the customers who meet these conditions.

| max_premium | min_premium | avg_premium |
|---|---|---|
| 540165 | 2630 | 31765.94979006298 |

| max_premium | min_premium | avg_premium |
|---|---|---|
| 540165 | 2630 | 31765.94979006298 |

Fig-21 Output for Retrieving Maximum, Minimum, and Average Annual Premiums for Customers who Responded and have a Vehicle with Damage.

**17. Counting the number of customers with response=1 and vehicle damage=1 grouped by gender:**

**Query:**

```
SELECT c.Gender, COUNT(*) AS count
FROM dbms.Customers c
INNER JOIN dbms.Vehicles v ON c.id = v.id
WHERE c.Response = 1 AND v.Vehicle_Damage = 1
GROUP BY c.Gender;
```

**Interaction with database:** The query interacts with the "Customers" and "Vehicles" tables in the "dbms" database and performs an inner join to match records based on their ID. It then filters the records based on specific conditions and groups the results by gender to count the number of customers in each category.

| gender | count |
|---|---|
| 1 | 27961 |
| 0 | 17767 |

Fig-22 Output for counting the number of customers with response=1 and vehicle damage =1 and grouped by gender

**18. Retrieving customer information with high annual premium:**

**Query:**

```
SELECT c.id, c.Age, c.Gender, p.Annual_Premium,
v.Vehicle_Age, v.Vehicle_Damage
FROM "dbms".Customers c
JOIN "dbms".Policies p ON c.id = p.id
JOIN "dbms".Vehicles v ON c.id = v.id
WHERE c.Response = 1 AND p.Annual_Premium >=
(SELECT AVG(Annual_Premium) FROM "dbms".Policies
WHERE id = c.id)
ORDER BY p.Annual_Premium DESC;
```

**Interaction with database:** The query involves joining three tables ("Customers", "Policies", and "Vehicles") in the "dbms" database based on their ID columns. It uses a subquery to calculate the average annual premium for each customer's policy ID and filters records based on the customer response and policy annual premium. The results are sorted by policy annual premium in descending order and include customer information and policy and vehicle details.

| id | age | gender | annual_premium | vehicle_age |
|---|---|---|---|---|
| 54744 | 26 | 1 | 540165 | 0 |
| 172258 | 40 | 1 | 489663 | 1 |
| 136305 | 50 | 1 | 472042 | 1 |
| 281680 | 45 | 0 | 472042 | 1 |
| 59101 | 41 | 0 | 340439 | 1 |
| 102294 | 43 | 1 | 336395 | 2 |
| 37856 | 47 | 1 | 336395 | 1 |
| 170381 | 44 | 1 | 316563 | 1 |

Fig-23 Output for Retrieving customer information with high annual premium.

# CLOUD ANALYTICS AND DATA WAREHOUSE IMPLEMENTATION FOR A HISTORICAL DATASET

## 14. DAGS

**DAG1 : preprocess_and_split_dataset.py**
**Description:**
The above DAG (Directed Acyclic Graph) in Airflow is named "preprocess_and_split_dataset" and is designed to preprocess and split a dataset into three tables, namely customer, vehicle, and policy tables. The DAG contains one task named "preprocess_dataset" which executes a Python function to load a CSV file from an S3 bucket, preprocess the data, and split it into three tables. The resulting tables are then converted to CSV format and uploaded to the same S3 bucket in a separate folder. The DAG is scheduled to run manually and has one dependency, i.e., it does not depend on any previous task.

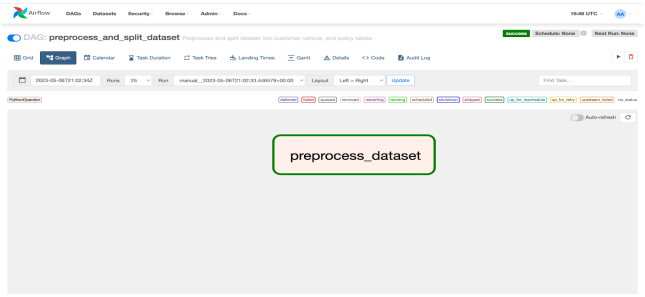


Fig-24 Dag1

**DAG2 :load_csv_to_mysql.py**
**Description:**
The above code defines an Airflow DAG named 'load_csv_to_mysql' that loads data from CSV files stored in an S3 bucket and writes them to a MySQL database using SQLAlchemy. The DAG consists of three tasks, each of which calls a Python function to load a specific CSV file into MySQL. The S3 bucket and object keys, as well as the MySQL database credentials, are specified in the Python functions. The task dependencies are defined such that the 'load_customers_to_rds' task is executed first, followed by 'load_vehicles_to_rds', and finally 'load_policies_to_rds'. The DAG is scheduled to run on a manual trigger only, as indicated by the 'schedule_interval=None' argument in the DAG definition.

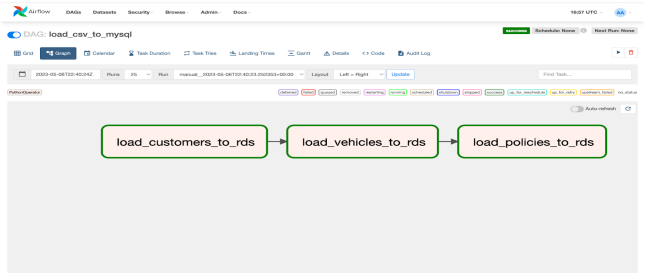


Fig-25 Dag2

**DAG3 :export_to_s3.py**
**Description:**
This is an Airflow DAG that exports data from a MySQL RDS instance to CSV files and then uploads them to an S3 bucket. The exported tables include Customers, Vehicles, and Policies. Each table has a corresponding Python function that exports the data and uploads the CSV file to S3 using an S3Hook. The DAG is scheduled to run once a day.

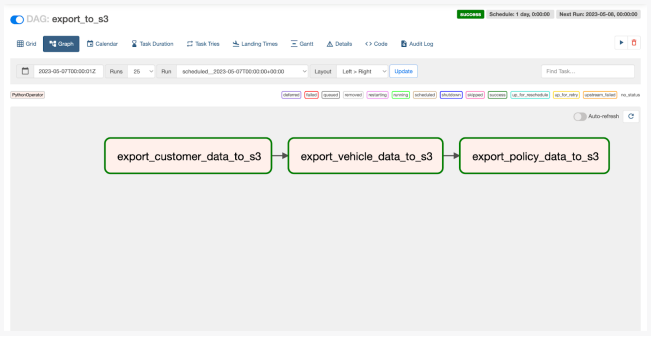


Fig-26 Dag3

**DAG4 :copy_data_from_s3_to_redshift.py**
**Description:**
This DAG copies data from S3 to Redshift in three tables: customers, vehicles, and policies. It uses PythonOperator to call three separate functions, each of which connects to Redshift and S3, truncates the target table, and copies the data from S3 to Redshift using the COPY command. The DAG has default arguments such as retries and retry delay, and it does not have a scheduled interval, meaning it will only run when manually triggered.

# CLOUD ANALYTICS AND DATA WAREHOUSE IMPLEMENTATION FOR A HISTORICAL DATASET

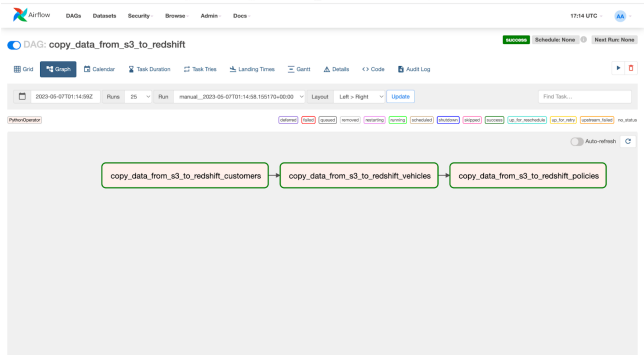


Fig-27 Dag4

## 15. KEY PERFORMANCE METRICS THROUGH VISUALIZATION

### 15.1 PowerBi :

Power BI is a business analytics tool that allows users to create interactive visualizations and reports from various data sources. It provides a wide range of visualization options, including bar charts, line charts, scatter plots, heat maps, and more. Power BI has a user-friendly interface and allows users to drag and drop fields to create visualizations. It also allows users to create dashboards, which are interactive presentations that can be shared with others. Power BI is tightly integrated with other Microsoft tools, such as Excel and SharePoint, and can be accessed from various devices, including desktops, tablets, and mobile phones.

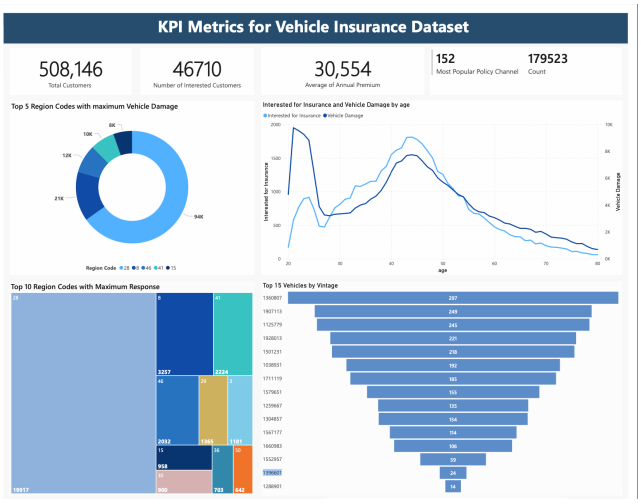


Fig-28 Powerbi Dashboard

### 15.2 Tableau :

Tableau is a powerful data visualization tool that enables users to create interactive and dynamic visualizations from various data sources. It provides a wide range of visualization options, including bar charts, line charts, scatter plots, heat maps, and more. Tableau has a user-friendly interface and allows users to drag and drop fields to create visualizations. It also allows users to create dashboards and stories, which are interactive presentations that can be shared with others.

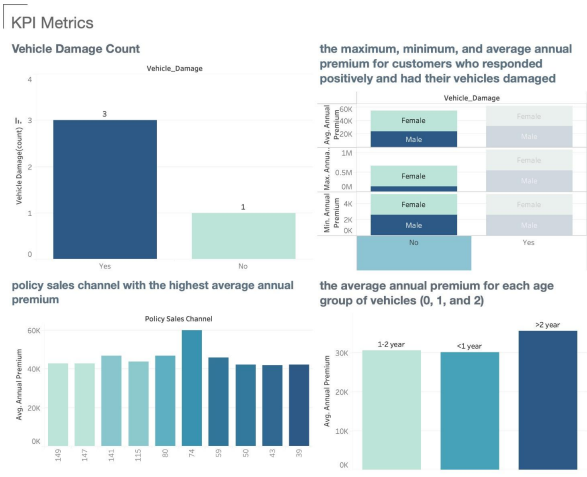


Fig-29 Tableau Dashboard

### 15.3 Visualizations :

**1.Top 5 regions with maximum vehicle damage:**

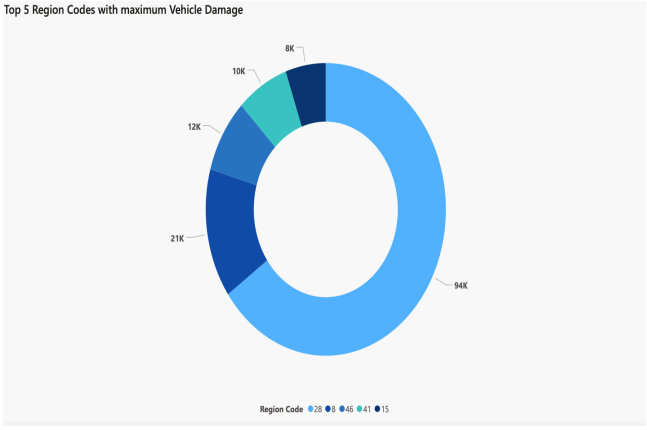


Fig-30 Top 5 regions with maximum vehicle damage

The above visual shows us the top 5 regions with the maximum vehicle damage. This visual gives us an overall

picture of the top most regions with most vehicle damage that ultimately helps us to boil down to our area of interest who should consider taking the vehicle insurance.

**2.Trendlines for Vehicle Damage and the people interested in insurance by Age:**

This is a line graph that shows the trendlines for Vehicle Damage and the people interested in insurance by Age. We can clearly infer from this visual that the majority of vehicle damage and interested people fall between the range of 30-60. We can a spike in the 20 -30 years, stating that vehicle damage is more for this age group.

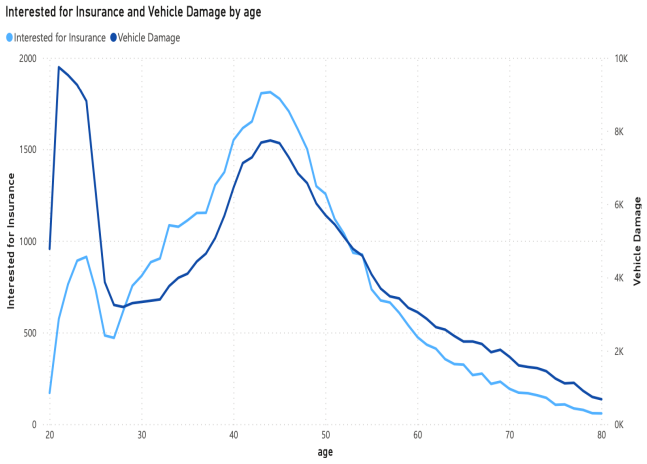


Fig-31 Trendlines for Vehicle Damage and the people interested in insurance by Age

**3.Top 15 Vehicles by Vintage:**

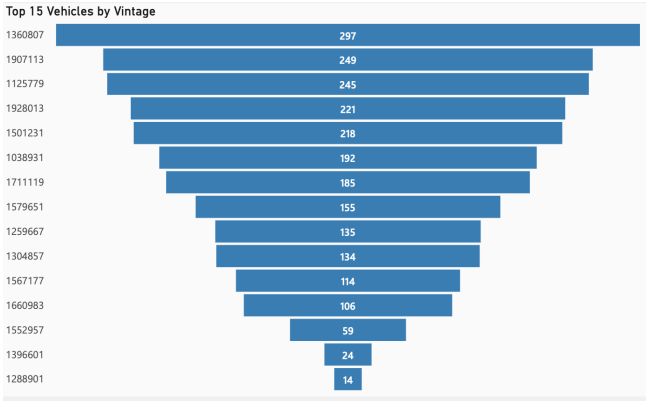


Fig-32 Top 15 vehicle by vintage

The above funnel chart shows us the top 15 Vehicles by Vintage (Number of days of association with the organization). This visual will help to recognize the key customers for the insurance company.

**4.Top 10 region codes with maximum response:**

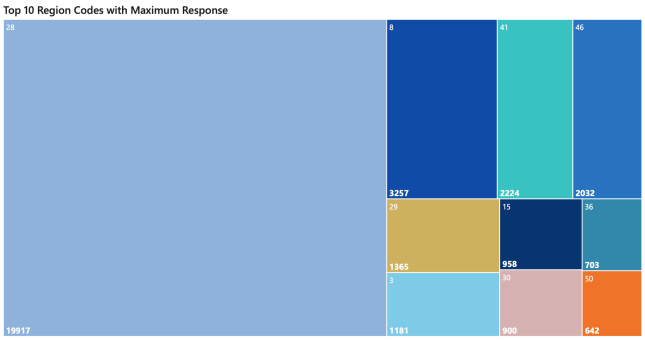


Fig-33 Top 10 region codes with maximum response

The above tree-map shows us the top 10 regions which have maximum response for the insurance policy. This visual will help the business to understand which region to target based on the count of the responses.

**5. Average annual premium for Vehicle Age groups:**

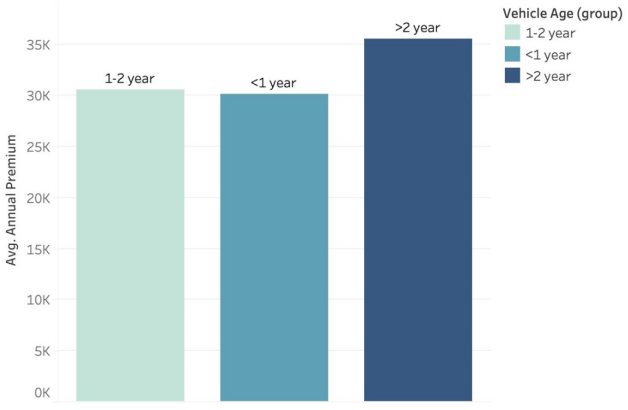


Fig-34 Average annual premium for Vehicle Age groups

The above bar graph plots the average annual premium for Vehicle Age groups ( 1-2 years, <1 year, >2 years). This visual provides us a greater picture as to how old the vehicle is and what is the average premium cost they are paying to the insurance company. It provides the business

with the existing status of the company.

**6. Average annual premium for Vehicle Age groups:**

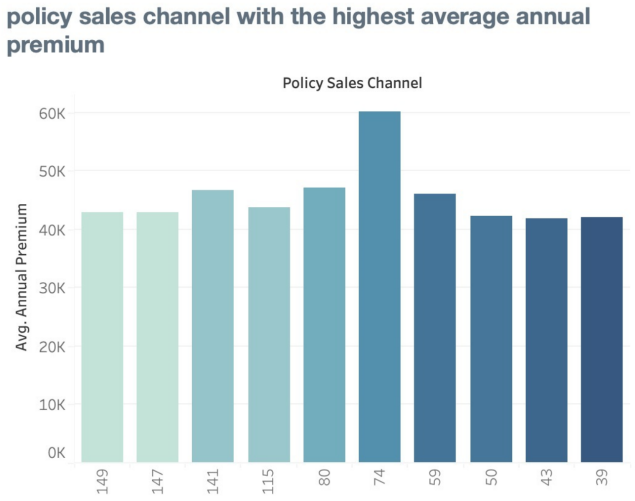


Fig-35 Top 10 policy sales channels with the highest average annual premium.

The above bar chart shows us the Top 10 policy sales channels with the highest average annual premium. This visual provides the business the existing status of the company. We can infer that policy channel 74 has the highest annual premium over the years.

**7. Vehicle Damage Count:**

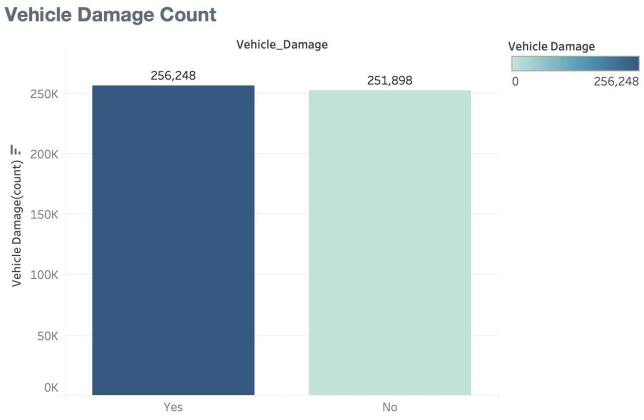


Fig-36 Powerbi Dashboard

The above bar chart shows us the overall vehicle damage count, the bar chart shows that out of a total of 508,146 vehicles in the dataset, 256,248 vehicles are damaged and 251,898 vehicles are not damaged.

**8. Maximum, minimum and average annual premium for customers who responded positively and had their vehicles damaged:**

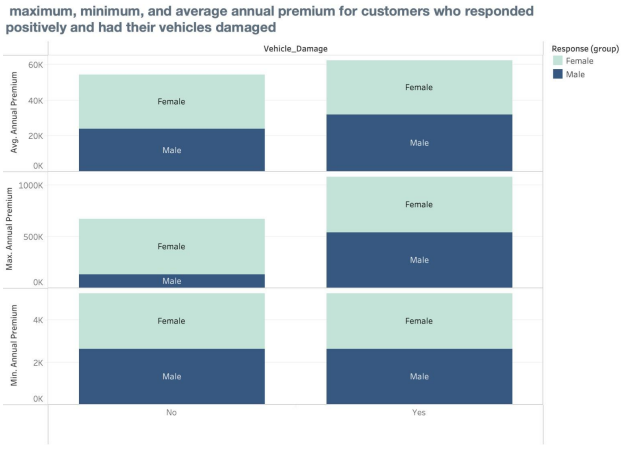


Fig-37 Powerbi Dashboard

The above stacked bar chart shows us the Maximum, minimum and average annual premium for customers who responded positively and had their vehicles damaged and they are grouped by their gender.

## 16. PERFORMANCE MEASUREMENT

Database Performance depends on a variety of factors, including hardware, indexing, query optimization, data volume, and concurrency. By addressing these factors, database administrators can improve the performance of their databases and ensure that they are able to meet their needs.

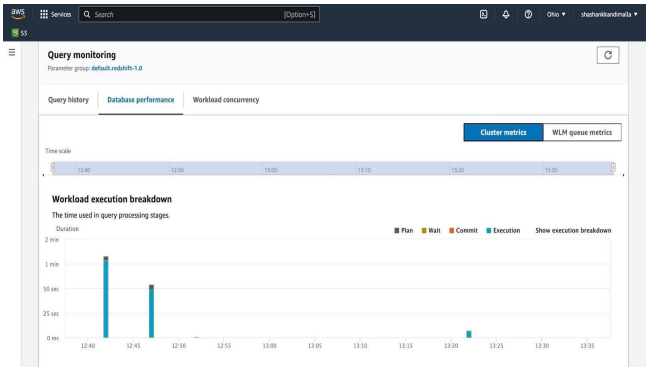


Fig-38 performance dashboard

# CLOUD ANALYTICS AND DATA WAREHOUSE IMPLEMENTATION FOR A HISTORICAL DATASET

## 16.1 QUERY PERFORMANCE

We can see that some of the queries below are taking longer to execute due to the use of JOINS and subqueries. On the other hand, simple queries such as counting the number of customers by a particular attribute or grouping by a single attribute tend to execute faster. Additionally, queries that utilize indexes or optimized queries can also improve performance.

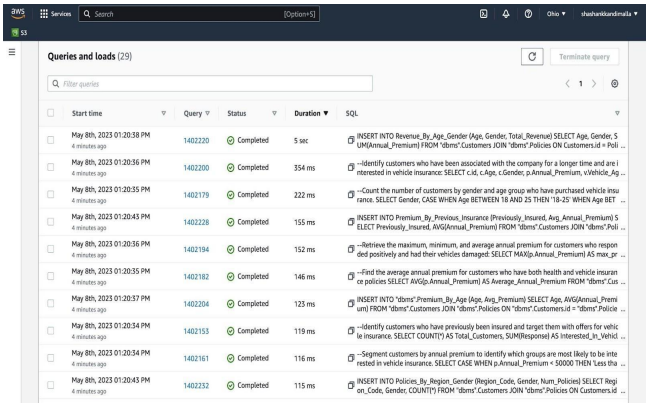


Fig-39.a Query performance.

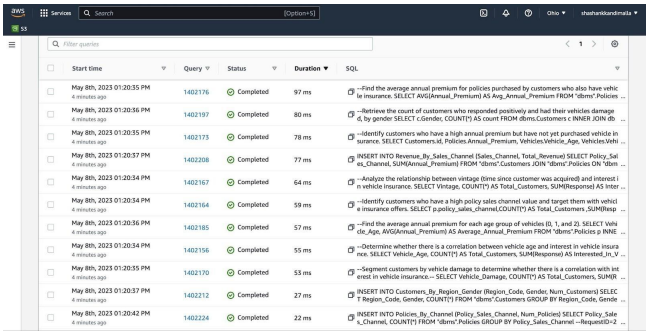


Fig-39.b Query performance.

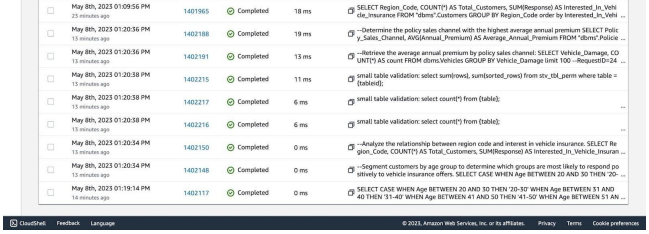

Fig-39.c Query performance.

## 17. CONCLUSION AND FUTURE WORK

In conclusion, the implementation of a cloud analytics[2] and data warehouse solution for historical vehicle insurance data has enabled our team to gain insights into customer behavior and preferences. By leveraging cloud-based resources and technologies, such as Amazon Redshift, Apache Airflow, PowerBI and Tableau, we have created a scalable and efficient solution for analyzing and visualizing historical data.

Through the use of ETL processes, we were able to preprocess and transform the vehicle insurance dataset, storing it in a cloud SQL database. From there, we performed analytics and generated visualizations using Tableau, allowing us to explore trends and identify key performance metrics (KPIs) related to customer behavior.

Looking towards future work, there are several potential areas for further development and improvement. For example, the integration of additional data sources, such as demographic or economic data, could provide more comprehensive insights into customer behavior. Additionally, the incorporation of machine learning models could enable us to make predictions and generate insights based on the historical data. Finally, further optimization of the ETL pipeline could enhance the efficiency and scalability of the solution, allowing us to handle larger and more complex datasets in the future.

## 18. REFERENCES


[1] Dataset
https://www.kaggle.com/code/yashvi/vehicle-insurance-eda-and-boosting-models/input

[2] Cloud Analytics
https://www.softwareag.com/en_corporate/resources/mainframe-modernization/wp/mainframe-data-integration.html?utm_source=google&utm_medium=cpc&utm_campaign=a-n_freedom-for-legacy&utm_region=hq&utm_subcampaign=stg-1&utm_content=whitepaper_mainframe-data-integration-connx&gclid=Cj0KCQjwu-KiBhCsARIsAPztUF32mh46NjB17DtO4k-atFAYXNPtl-gYBvEezKjo30FM56LKTvQOk8saAmf1EALw_wcB

[3] ETL
https://www.ibm.com/topics/etl

[4] Cloud Composer

https://cloud.google.com/composer/docs/concepts/overview

[5] Amazon Web Services
https://docs.aws.amazon.com/?nc2=h_ql_doc_do

[6] Fundamentals of Database Systems, 7th edition, by Elmasri and Navathe, Addison Wesley Publishing Company, Menlo Park, CA.

[7] Jeffrey A. Hoffer, Ramesh Venkataraman, and Heikki Topi. Modern Database Management.Pearson. 13th Edition. ISBN-13: 978-1292263359

[8]Apache Airflow
https://airflow.apache.org/docs/apache-airflow/stable/