Table of Contents

Abstract

Project Overview

2.1 Project Question

2.2 About Data

Problem Statement

Technologies Used

4.1 Apache Airflow

4.2 Google Cloud Platform (GCP)

4.3 Terraform

System Architecture

5.1 Airflow DAG Diagram

Step-by-Step Procedure

6.1 Setup and Configuration

6.2 Workflow Execution

Code Sections

7.1 Airflow DAG Code

7.2 Terraform Infrastructure Code

Dashboard Implementation

8.1 Dashboard Overview

8.2 Explanation of Charts

Conclusion

Abstract:

In a rapidly evolving data technology environment where the flood of information is growing exponentially, the demand for flexible, scalable data pipelines has never been more critical This project appears as a strategy responding to this need, which wants to not only meet but overcome challenges presented by today's data landscape. Its primary goal is to structure, validate, and optimize data migration leveraging the collective capabilities of Apache Airflow, Google Cloud Platform (GCP), and Terraform

Project Overview

At the heart of this initiative is fundamental research on how to design a data acquisition device that is not only robust but also scalable to handle a myriad of data sources The important question drives the project: How can we create a flexible system that can consume simple data from different origins? To answer this question, our approach analyzes the complexity of data in depth, and shows that the strength of any data pipeline depends not only on the ability to transfer data, but also on the ability to understand it , maintain, and adapt to the nuances of various also data types and systems.

2.1 Project Question:

The goal of this project is to apply everything we learned in this course and build an end-to-end data pipeline.

Remember that to pass the project, you must evaluate 5 peers. If you don't do that, your project can't be considered complete.

For the project, you are asked to build a complete data pipeline for data ingestion and further analytical processing to gain some insights from the report/dashboard.

For that, you will need:

Select a dataset that you're interested in (e.g. https://github.com/awesomedata/awesome-public-datasets, https://github.com/DataTalksClub/data-engineering-zoomcamp/blob/main/week_7_project/datasets.md)

Create a pipeline for processing this dataset and putting it into a data lake

Create a pipeline for moving the data from the lake to a Data Warehouse

Transform the data in the data warehouse: prepare it for the dashboard

Create an analytical dashboard/report

Your dashboard should contain at least two tiles, I suggest you include:

1 graph that shows the distribution of some categorical data

1 graph that shows the distribution of the data across a temporal line

Make sure that your graph is clear to understand by adding references and titles.

2.2 About Data

A brief introduction on the data set :

The dataset provides information about passengers on the Titanic, including details such as their names, ages, genders, ticket information, cabin details, and whether they survived the tragic sinking of the ship. The dataset is commonly used for exploring patterns and trends related to passenger demographics and survival rates.

Passenger ID:

It appears that each passenger is uniquely identified by this column. Although it isn't a valuable source of information for analysis, it can be helpful in locating particular passengers.

Survived:

The value in this column designates if a passenger made it through (1) or did not (0).

It can be applied to examine the Titanic passenger survival rate as a whole.

Pclass (Ticket class):

The ticket's class (1st, 2nd, or 3rd class) is indicated in this column.

If there is any evidence of a relationship between ticket class and survival, it can be examined.

Name:

The names of the passengers are listed in this column.

Although it has limited analytical usefulness by itself, it can be used to identify certain people or families.

Sex:

Denotes the passenger's gender (male or female).

It can be examined to determine whether gender and survival are related.

Age:

Indicates the passengers' ages.

Understanding the age distribution of the passengers can help with their demographic analysis.

A passenger's SibSp (Number of siblings/spouses aboard):

indicates how many siblings or spouses they had with them. This can be examined to learn more about Titanic family dynamics.

Parch (Number of parents/children aboard):

Indicates how many parents or kids a traveller has travelling with them. It can be applied to groups and family ties analysis.

Ticket:

The passenger's ticket number is shown in this column. It might not be immediately helpful for analysis if no patterns are found.

Fee :

The amount paid for the ticket is shown by the fare. An examination of the fares can reveal information about how passengers are charged for their tickets.

Cabin:

Identifies the passenger's actual cabin number. Though there are some missing statistics, this information can be helpful for figuring out where the cabins are on the ship.

Entered:

Indicates the port of entry (C stands for Cherbourg, Q for Queenstown, and S for Southampton).

Now some trends in the data :

The 'Survived' column analysis can be used to determine the overall survival rates of the passengers.

Class Distribution: Knowing how passengers are distributed among various classes (Pclass) can reveal socioeconomic trends.

Age Distribution: Examining the 'Age' column can reveal information about the passenger population's age distribution.

Gender Distribution: The percentage of male and female passengers may be seen by looking at the 'Sex' column.

Family Size: The distribution of family sizes on board the Titanic can be examined using the 'SibSp' and 'Parch' columns.

Problem Statement

1. Survival Rate Analysis: in the survival rate analysis we use the survival column and perform an analysis to understand the following.

· To determine and visually represent the overall survival rate of passengers aboard the Titanic. To find the total number of passengers that have survived.

· Conduct an in-depth analysis to understand factors influencing survival outcomes.

2. Age Distribution Exploration: In this age distribution analysis, we have used the data to.

· Investigate the age distribution of Titanic passengers.

· Examine the general age of survivors to identify any patterns or trends.

3. Gender Analysis: by the gender analysis we

· Visualize the gender distribution among Titanic passengers.

· Explore the average age of the population, focusing on the ages of survivors and non-survivors.

· Determine and visualize the gender-specific survival rates to identify any disparities.

4. Family Size Investigation: as a lot of passengers have been identified to have families in the family size analysis we

· Examine the distribution of family sizes (SibSp + ParCh) among Titanic passengers.

5. Fare Distribution Analysis: this analysis uses the price range of the various tickets purchased by the passengers to

· Visualize the distribution of ticket fares paid by Titanic passengers.

· Identify the general fare range, as well as the highest and lowest fare rates.

· Explore potential correlations between fare amounts and survival rates, presenting findings visually.

Technologies Used:

Apache Airflow:

In the world of data engineering and business process management. Apache Airflow is used as a very flexible and scalable solution. Airflow is a growing open-source platform that offers control and visibility to manage data systems

The concepts like Directed Acyclic Graphs (DAGs), operators, schedulers all are based on the unique approach to performance management. DAGs allow us to have performance relationships. It is very dynamic and takes care of the workflow.

Airflow improves the user interaction because it gives us details of DAGs task status and application information. A unique feature of the platform is its ability to handle complex workflows while enabling parallel execution.

Airflow has a modular architecture that allows users to customise their own applications and it also supports various integrations that ensure the seamless integration into other existing technologies.

In terms of workflow execution, it needs effective monitoring and logging. Apache Airflow is known for providing real time insights into the task execution status through its user interface and detail logs. Airflow has very active community around it, that contributes vastly to its development that resulted in a lot of plugins, operators and integrations that enhanced its capabilities

In the realm of data engineering, Apache Airflow has become a central tool for computing ETL processes, managing data migrations and handling various data related workflows. Despite all the benefits, Apache also comes with some challenges, as it requires the best practices, proper configuration, optimization of DAGs designs, and a thorough understanding of scheduler and executor options to achieve optimal performance.

Apache airflow is still on its path of innovation with plans for user experience enhancement, scalability and improvement. It has goal to become an even faster tool for workflow processes. In the complex landscape of data engineering, Apache Airflow emerges as a very dynamic and empowering solutions that equips users with the necessary tools to navigate the complexity of modern data pipelines with precision

4.2 Google Cloud Platform (GCP)

In the domain of cloud computing, Google cloud platform (GCP) emerged as a solution that provides a plethora of services that help in addressing various needs of businesses, developers and data engineers. It was established in 2008. GCP utilizes Google's reliable infrastructures that offers a scalable, flexible and secure environment for application hosting, data management and utilization of advanced technologies.

Google cloud platform has key components and services that help in computing services, storage, databases, data analytics, networking, security and developer tools. It has noticeable offerings like Compute Engine for Infrastructure as a service (IaaS), App Engine for Platform as a service (PaaS), and Kubernetes to manage containerized application. The storage and databases services like Cloud Storage, Cloud SQL and Cloud Firestore help in diverse data storage and management requirements.

Other than that, data and analytics services like BigQuery that help in serverless data warehousing, Dataflow for stream and batch processing. Whereas Dataproc helps in managing big data clusters. GCP also provides networking infrastructure and security tools like Virtual Private Cloud (VPC) along with developer tools like Cloud Build, Cloud Source Repositories and Cloud Trace.

It has advantages like scalability, global reach through strategically located data centers, security measures, robust big data and analytical capabilities and flexible pay-as-you-go pricing models. But still organizations can encounter challenges related to cost management and resource optimization that have been mitigated through practices life utilizing cost management tools and regularly reviewing and optimizing resource configurations.

In short, Google cloud platform is committed to providing a reliable and innovative cloud computing environment. It has an extensive range of services, scalability and security features. It is also counted in cutting-edge technologies, that makes it an appealing choice for organizations whose aim is to leverage cloud capabilities. In a constantly changing digital landscape, it remains a leading force that helps businesses to empower and data engineers to build scale and innovate in the cloud.

4.3 Terraform

Terraform is a very robust infrastructure as Code (IaC) tool that was designed by Hachiko. The goal was to simplify the management of complex infrastructure across different cloud providers and on-premises environments. This tool helps users to define their own infrastructures by using a declarative configuration language, that brings efficiency and speed to the deployment process.

With the usage of terraform, users can easily explain the desired state of their infrastructure by specifying the resources and their configurations. The tool also automates the creation, modification and destruction of these resources, that ensures consistency across the deployment. It's a very versatile platform and its nature is evident in its support for multiple cloud providers including AWS, Azure and Google cloud.

Terraform has modular design and extensibility that lets users organize and manage their complex infrastructures effortlessly. It also has a state management system that keeps track of the current infrastructure state, facilitates collaboration among teams. It also computes and processes cloud resources, configures networks and manages on-premises systems. Terraform streamlines the deployment of life cycles, providing a very standardized and robust approach to infrastructure management.

System Architecture

5.1 Airflow DAG Diagram

Step-by-Step Procedure

Creating a data pipeline in Google Cloud using Apache Airflow involves several steps. Here's a step-by-step procedure to guide you through the process:

Set Up Google Cloud Platform (GCP):

- Create a GCP account if you don't have one already.
- Set up a project in the GCP Console.
- Enable necessary APIs and services such as BigQuery, Cloud Storage, and Dataflow.

Install and Configure Apache Airflow on GCP:

- Set up Airflow on Cloud Composer (managed Airflow service on GCP).
- Configure Airflow connections and variables for authentication with GCP services.

6.1 Setup and Configuration

1. **Extraction of Data**

   To extract data from GitHub and load a CSV file in buckets using Google Cloud Platform (GCP) Apache Airflow, we need to follow the below steps:

   - **GitHub Data Extraction**:
     We have used GitHub API to access the data from an open GitHub repository. This involves accessing the repository contents programmatically.
   - **Data Processing**:
     Process the extracted data as needed. This involves cleaning the data and transforming it to a suitable format for storage.
   - **CSV File Creation**:
     Once the data is processed, we then generate a CSV file containing the extracted information.
   - **Upload to GCP Bucket**:
     The next step is to utilize the Google Cloud Storage Python client library to interact with GCP buckets. Authenticate with GCP credentials and use functions like upload_blob() to upload the CSV file to a designated bucket.

2. **Aggregate all the data**

   Below are the steps to transform and aggregate data in a CSV file using Google Cloud Platform (GCP) Apache Airflow:

   - **Define DAG Parameters**:
     Define parameters for your DAG, such as the CSV file(titanic.csv) location in a GCP bucket, the output location for transformed data, and any other configuration settings needed for data transformation and aggregation.
   - **Extract Data from CSV**:
     Use Airflow operator – **GoogleCloudStorageToBigQueryOperator** to extract the CSV file from a GCP bucket to BigQuery for processing.
   - **Data Transformation**:
     Implement Python functions to perform data transformation tasks. This will involve cleaning the data, converting data types, filtering rows as required.
   - **Data Aggregation**:
     Aggregate the transformed data based on your requirements. This could involve grouping data by certain fields and computing aggregate functions - statistics = df.describe(), which will provide the sum, average, count, mean, etc.
   - **Load Transformed Data**:
     Use Airflow operators - **PythonOperator** to load the transformed and aggregated data into BigQuery tables to the GCP storage buckets.

3.Visualisation:

- **Access Google Data Studio:**
  - Navigate to Google Data Studio (https://datastudio.google.com/).
  - Sign in with your Google account credentials.
- **Create a New Report:**
  - Navigate to “Create" to start a new report.
  - Choose Big Query as the data source.
- **Select Dataset and Dimensions:**
  - Select the Big Query dataset containing the Titanic data.
  - Choose the dimensions (e.g., passenger class, sex, age) and metrics (e.g., count, survival rate) you want to visualize.
- **Design Your Report:**
  - Drag and drop elements onto the canvas, such as scorecards, tables, graphs, and charts.
  - Set up each component's dimensions and metrics, as well as the data source.
  - Personalize your visuals' format, color scheme, and style.
- **Add Interactivity:**
  - To improve the user experience, add interactive features like drill-down capabilities, date range selectors, and filters.

4. Store the data:

Step 1: Choose the Data Storage Solution

- **Google Cloud Storage (GCS):**
  - Store unstructured data in GCS, including files, backups, and multimedia.
  - GCS provides many storage classes that are tailored to suit varying access patterns and budgets.
- **Google Big Query:**
  - Use Big Query to store organized data and execute quick, SQL-like queries on huge datasets.
  - Big Query is appropriate for activities related to analytics, machine learning, and data warehousing.

Step 2: Upload Data to Google Cloud Storage (GCS)

- **Create a Bucket:**
  - Open the Storage section of the GCP Console and create a new bucket.
  - Give your bucket a name that is unique around the globe.
- **Upload Data:**
  - Use the GCP Console, the gs-util command-line tool, or Cloud Storage APIs to upload your data files to the bucket.
  - Within the bucket, data can be arranged using prefixes or folders.

Step 3: Load Data into Google Big Query

- **Create a Dataset:**

- To include your tables, build a new dataset via the Big Query Console.
- Select a dataset ID and, if required, establish the default table expiration time.

➢ **Load Data:**

- Use a variety of techniques to load data into Big Query tables:
- Straight from GCS: Fill Big Query tables with information taken from files kept in GCS.
- Streaming: Use streaming inserts to instantly add data to Big Query tables.
- From other sources: To ingest data into Big Query, use tools like Dataflow, Cloud Functions, or others.

Step 4: Data Organization and Access Control

➢ **Organize Data:**

- Organize your data in GCS buckets and Big Query datasets according to your data management and access requirements.
- To maintain data organization and manageability, we use folder hierarchies and naming conventions.

➢ **Access Control:**

- Arrange your data in Big Query datasets and GCS buckets according to your needs for access and data management.
- Allow users and service accounts the appropriate access in accordance with the least privileged notion.

Step 5: Monitor and Manage Data

➢ **Monitoring:**

- Use GCP monitoring tools to keep an eye on performance indicators, access trends, and data storage consumption.
- Create alerts to be notified about unusual behaviors or resource use.

➢ **Data Lifecycle Management:**

- Adopt data lifecycle policies to control data archiving, deletion, and retention.
- Take into account utilizing GCS storage classes to optimize storage expenses depending on data access patterns.

5. Dashboard:

➢ **Access Looker Studio**: Open Looker Studio, Looker's dashboard and report creation environment, after logging into your Looker instance.

➢ **Data Modelling**: Make sure your Looker data model is well-defined and structured before building a dashboard.

➢ **Add Tiles**: You can begin adding tiles to your dashboard as soon as it is built. Data visualizations or pieces are called tiles. Click the "Add Tile" button in the dashboard editor to add a tile.

- **Select Visualization Type**: Select the kind of visual aid you wish to have on your dashboard. Looker offers a range of visualization options, including custom HTML, tables, charts, and maps.

- **Configure Visualization**: By choosing the proper data fields, measurements, dimensions, and filters, you may configure the visualization. To see the data in the format you want, adjust the visualization parameters.

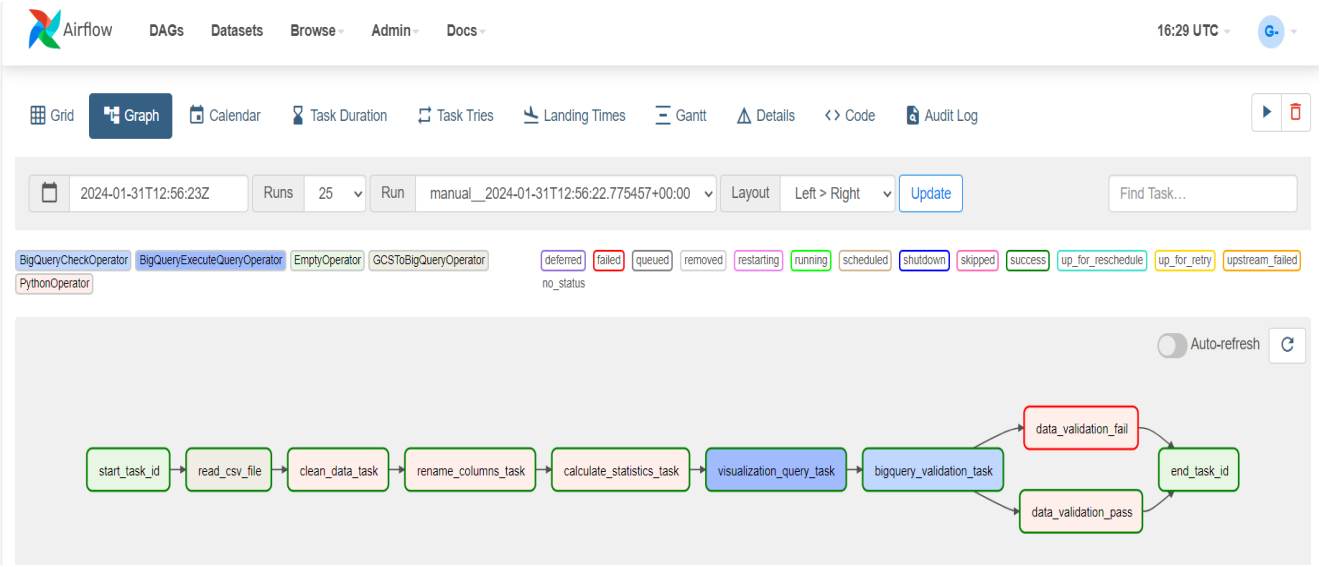
- **Arrange Tiles**: Logically arrange the tiles on the dashboard layout to arrange the data. Tiles can be resized and rearranged to fit the layout by simply dragging and dropping them.

- **Apply Filters and Interactivity**: To enable users to interactively examine the data, incorporate filters and interactive elements into your dashboard. By choosing filters, digging into data, and switching between displays, users may engage with the dashboard.

- **Save and Share**: Save your modifications after you're happy with the content and layout of the dashboard. After that, you can allow additional people or groups in your business to access the dashboard.

- **Iterate and Improve**: Review and update your dashboard frequently in response to user comments and evolving business needs. To keep the dashboard current and helpful, make iterations to the layout, design, and information.

6.2 Workflow Execution:



- **start_task_id**: It signifies the commencement of a task of DAG, it does not hold so much meaning in execution. Its appearance in green indicates the successful running of the corresponding step.
- **read_csv_file**: The green highlight also signifies that the dataset has been successfully loaded from Google Cloud Storage Bucket to BigQuery table and is now ready for further processing.
- **clean_data_task**: In this stage, we are eliminating any flaws like duplicity from the dataset to advance with a refined dataset. Upon the successful completion of this step, we can proceed with our prepared dataset, replacing it with the original. The green indicator confirms the success of this step.
- **rename_columns_task**: In it, we renamed certain columns in the dataset according to our requirements and replaced the original with it. This step also indicates the success with green.
- **calculate_statistics_task**: In this phase, we are in the process of calculating the necessary statistics for our dataset. This step has been successfully initiated.
- **visualization_query_task:** This task involves generating visualizations to better understand the dataset. In this, we collect data for various visualizations, such as passenger counts by gender and survival by class. Currently, it is also in the stage of success.
- **bigquery_validation_task** Data validation is an important step which determines if our flow is error free in BigQuery. It involves two steps:
    1) **data_validation_fail**: This step shows us that our data validation has been failed and it shows ValueError, but as in our case, it highlights in yellow, that indicates "up_for_retry". It means this step is not successfully executed and need another try, which shows our data validation is not failed and is error-free.
    2) **data_validation_pass**: This step shows our data validation has been successfully completed without any error. In our case, it is highlighted in green which means we have successfully executed data validation step without any error.
- **end_task_id** All the steps have been successfully executed, and we are now concluding the task successfully with endpoint.

Code Sections

7.1 Airflow DAG Code

1. Import Statements:

The import statements at the beginning of the script bring in necessary libraries and modules to support different functionalities within the DAG. Key imports include modules for Airflow operators (e.g., GCSToBigQueryOperator, PythonOperator, BigQueryExecuteQueryOperator), Google Cloud services (e.g., storage, bigquery), and data processing tools (e.g., pandas).

```python
# Import necessary libraries
from airflow import models
from airflow import DAG
import airflow
from datetime import datetime, timedelta
from airflow.contrib.operators.dataflow_operator import DataFlowPythonOperator
from airflow.operators.dummy_operator import DummyOperator
from airflow.providers.google.cloud.transfers.local_to_gcs import LocalFilesystemToGCSOperator
import os
from airflow.operators.bash import BashOperator
from google.cloud import storage, bigquery
import pandas as pd
from airflow.operators.python import PythonOperator
from airflow.providers.google.cloud.transfers.gcs_to_bigquery import GCSToBigQueryOperator
from airflow.providers.google.cloud.operators.bigquery import BigQueryExecuteQueryOperator
from airflow.providers.google.cloud.operators.bigquery import BigQueryCheckOperator
```

2. Default Arguments:

The default_args dictionary sets default parameters for the DAG, including:

owner: The owner of the DAG.

retries: The number of retries each task should have.

retry_delay: The delay between task retries.

dataflow_default_options: Default options for Dataflow jobs, specifying the project, region, and runner.

These default arguments provide a foundation for the DAG's behavior.

```python
default_args = {
    'owner': 'Airflow',
    'retries': 1,
    'retry_delay': timedelta(seconds=50),
    'dataflow_default_options': {
        'project': 'noble-airport-412610',
        'region': 'us-central1',
        'runner': 'DataflowRunner'
    }
}
```

3. Project-Specific Variables:

PROJECT_ID and STAGING_DATASET are defined as project-specific variables, encapsulating the Google Cloud project ID and the name of the staging dataset in BigQuery.

PROJECT_ID = ' ’ # Here we need to provide our project ID

STAGING_DATASET = " " # Here we need to provide dataset name

4. DAG Configuration:

The dag object is created using the DAG class, defining the DAG's characteristics:

dag_id: A unique identifier for the DAG.

default_args: The default arguments dictionary.

schedule_interval: The frequency with which the DAG should be executed (daily in this case).

start_date: The date and time at which the DAG should start running.

catchup: Whether or not to execute any missed DAG runs.

DAG definition

```
# DAG definition
dag = DAG(
    dag_id='Titanic_DAG',
    default_args=default_args,
    schedule_interval='@daily',
    start_date=airflow.utils.dates.days_ago(1),
    catchup=False,
    description="DAG for data ingestion and transformation"
)
```

5. Start Dummy Task:

The start task is a DummyOperator, serving as the starting point for the DAG. It acts as a placeholder, and its execution does not perform any meaningful action.

Tasks

```
# Tasks
start = DummyOperator(
    task_id="start_task_id",
    dag=dag
)
```

6. GCS to BigQuery Task:

The titanic_data_dataset task uses GCSToBigQueryOperator to transfer data from a Google Cloud Storage bucket (degauravbk) to a BigQuery table ({PROJECT_ID}:{STAGING_DATASET}.titanic_data). Key parameters include the source bucket, source object, destination table, schema definition, and write disposition.

```python
read_csv_file = GCSToBigQueryOperator(
    task_id='read_csv_file',
    bucket='degauravbk',
    source_objects=['titanic_dataset.csv'],
    destination_project_dataset_table=f'{PROJECT_ID}:{STAGING_DATASET}.titanic_data',
    write_disposition='WRITE_TRUNCATE',
    autodetect=True,
    source_format='csv',
    allow_quoted_newlines='true',
    skip_leading_rows=1,
    schema_fields=[
        {'name': 'PassengerId', 'type': 'INTEGER', 'mode': 'NULLABLE'},
        {'name': 'Survived', 'type': 'INTEGER', 'mode': 'NULLABLE'},
        {'name': 'Pclass', 'type': 'INTEGER', 'mode': 'NULLABLE'},
        {'name': 'Sex', 'type': 'STRING', 'mode': 'NULLABLE'},
        {'name': 'Age', 'type': 'FLOAT', 'mode': 'NULLABLE'},
        {'name': 'SibSp', 'type': 'INTEGER', 'mode': 'NULLABLE'},
        {'name': 'Parch', 'type': 'STRING', 'mode': 'NULLABLE'},
        {'name': 'Ticket', 'type': 'STRING', 'mode': 'NULLABLE'},
        {'name': 'Fare', 'type': 'STRING', 'mode': 'NULLABLE'},
        {'name': 'Cabin', 'type': 'STRING', 'mode': 'NULLABLE'}
    ],
    dag=dag
)
```

Raw Data Snapshot:

	A	B	C	D	E	F	G	H	I	J	K
1	Passen	Survive	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarl
3	2	1	1	female	38	1	0	PC 17599	71.2833	C85	C
11	10	1	2	female	14	1	0	237736	30.0708		C
21	20	1	3	female		0	0	2649	7.225		C
28	27	0	3	male		0	0	2631	7.225		C
32	31	0	1	male	40	0	0	PC 17601	27.7208		C
33	32	1	1	female		1	0	PC 17569	146.5208	B78	C
36	35	0	1	male	28	1	0	PC 17604	82.1708		C
38	37	1	3	male		0	0	2677	7.2292		C
41	40	1	3	female	14	1	0	2651	11.2417		C
44	43	0	3	male		0	0	349253	7.8958		C
45	44	1	2	female	3	1	2	SC/Paris 2123	41.5792		C
50	49	0	3	male		2	0	2662	21.6792		C
54	53	1	1	female	49	1	0	PC 17572	76.7292	D33	C
56	55	0	1	male	65	0	1	113509	61.9792	B30	C
59	58	0	3	male	28.5	0	0	2697	7.2292		C
62	61	0	3	male	22	0	0	2669	7.2292		C
66	65	0	1	male		0	0	PC 17605	27.7208		C
67	66	1	3	male		1	1	2661	15.2458		C
75	74	0	3	male	26	1	0	2680	14.4542		C

7. Clean Data Task:

The clean_data_task is a PythonOperator that executes the clean_data function. This function loads data from the specified BigQuery table, removes duplicate rows using Pandas, and then replaces the original table with the cleaned data.

```python
def clean_data():
    # Load data from BigQuery into a Pandas DataFrame
    bq_client = bigquery.Client(project=PROJECT_ID)
    query = f"SELECT * FROM {PROJECT_ID}.{STAGING_DATASET}.titanic_data"
    df = bq_client.query(query).to_dataframe()

    # Remove duplicate rows
    df = df.drop_duplicates()

    # Save the cleaned DataFrame back to BigQuery
    df.to_gbq(destination_table=f'{PROJECT_ID}.{STAGING_DATASET}.titanic_data', if_exists='replace')

clean_data_task = PythonOperator(
    task_id='clean_data_task',
    python_callable=clean_data,
    dag=dag
)
```

8. Rename Columns Task:

Similar to the clean data task, the rename_columns_task PythonOperator executes the rename_columns function. This function loads data from BigQuery, renames specific columns, and updates the table in BigQuery.

```python
def rename_columns():
    # Load data from BigQuery into a Pandas DataFrame
    bq_client = bigquery.Client(project=PROJECT_ID)
    query = f"SELECT * FROM {PROJECT_ID}.{STAGING_DATASET}.titanic_data"
    df = bq_client.query(query).to_dataframe()

    # Rename columns
    df = df.rename(columns={
        'PassengerId': 'Passenger_ID',
        'Survived': 'Survival_Status',
        'Pclass': 'Ticket_Class',
        'Sex': 'Gender',
        'Age': 'Passenger_Age',
        'SibSp': 'Siblings_Spouses_Aboard',
        'Parch': 'Parents_Children_Aboard',
        'Ticket': 'Ticket_Number',
        'Fare': 'Ticket_Fare',
        'Cabin': 'Cabin_Number'
    })
    df['Ticket_Fare'] = pd.to_numeric(df['Ticket_Fare'], errors='coerce')

    # Change data types
    df['Parents_Children_Aboard'] = df['Parents_Children_Aboard'].astype(int)
    df['Ticket_Fare'] = df['Ticket_Fare'].astype(int)
    # Save the renamed DataFrame back to BigQuery
    df.to_gbq(destination_table=f'{PROJECT_ID}.{STAGING_DATASET}.titanic_data', if_exists='replace')

rename_columns_task = PythonOperator(
    task_id='rename_columns_task',
    python_callable=rename_columns,
    dag=dag
)
```

Clean And Renamed data Snapshot:

SCHEMA | DETAILS | PREVIEW | LINEAGE | DATA PROFILE | DATA QUALITY

Row	Passenger_ID	Survival_Status	Ticket_Class	Gender	Passenger_Age	Siblings_Spouse	Parents_Children_Ab
1	264	0	1	male	40.0	0	0
2	634	0	1	male	null	0	0
3	807	0	1	male	39.0	0	0
4	816	0	1	male	null	0	0
5	823	0	1	male	38.0	0	0
6	873	0	1	male	33.0	0	0
7	285	0	1	male	null	0	0
8	546	0	1	male	64.0	0	0
9	783	0	1	male	29.0	0	0
10	271	0	1	male	null	0	0
11	352	0	1	male	null	0	0
12	186	0	1	male	null	0	0
13	111	0	1	male	47.0	0	0

9. Calculate Statistics Task:

The calculate_statistics_task PythonOperator executes the calculate_statistics function. This function loads data from BigQuery, uses Pandas to calculate descriptive statistics, and stores the results in a new table in BigQuery.

```
def calculate_statistics():
    # Load data from BigQuery into a Pandas DataFrame
    bq_client = bigquery.Client(project=PROJECT_ID)
    query = f"SELECT * FROM {PROJECT_ID}.{STAGING_DATASET}.titanic_data"
    df = bq_client.query(query).to_dataframe()

    # Calculate statistics
    statistics = df.describe()

    # Save the statistics to BigQuery
    statistics.to_gbq(destination_table=f'{PROJECT_ID}.{STAGING_DATASET}.titanic_statistics', if_exists='replace')

statistics_task = PythonOperator(
    task_id='calculate_statistics_task',
    python_callable=calculate_statistics,
    dag=dag
)
```

Calculate statistics Output Snapshot:

SCHEMA | DETAILS | PREVIEW | LINEAGE | DATA PROFILE

Row	Passenger_ID	Survival_Status	Ticket_Class	Passenger_Age	Siblings_Spouse
1	1.0	0.0	1.0	0.42	0.0
2	223.5	0.0	2.0	20.125	0.0
3	446.0	0.0	3.0	28.0	0.0
4	668.5	1.0	3.0	38.0	1.0
5	891.0	1.0	3.0	80.0	8.0
6	891.0	891.0	891.0	714.0	891.0
7	446.0	0.38383838...	2.30864197...	29.6991176...	0.52300785...
8	257.353842...	0.48659245...	0.83607124...	14.5264973...	1.10274343...

10. Data Visualization Task:

The visualization_query_task utilizes BigQueryExecuteQueryOperator to execute a complex SQL query. The query aggregates data for various visualizations, such as passenger counts by gender and survival by class. The results are stored in a dedicated table for visualization purposes.

Task for data visualization

```
visualization_query_task = BigQueryExecuteQueryOperator(
    task_id='visualization_query_task',
    sql='''WITH PassengerCounts AS (
        SELECT
          Gender,
          COUNT(*) AS TotalPassengers
        FROM
          `noble-airport-412610.shashank.titanic_data`
        GROUP BY
          Gender
      ),

      SurvivalByClass AS (
        SELECT
          Ticket_Class,
          Survival_Status,
          COUNT(*) AS PassengerCount
        FROM
          `noble-airport-412610.shashank.titanic_data`
        GROUP BY
          Ticket_Class, Survival_Status
      ),

      AgeDistribution AS (
        SELECT
          Passenger_Age,
          COUNT(*) AS PassengerCount
        FROM
          `noble-airport-412610.shashank.titanic_data`
        WHERE
          Passenger_Age IS NOT NULL
        GROUP BY
          Passenger_Age
      ),
```

```
      SurvivalByGender AS (
        SELECT
          Gender,
          Survival_Status,
          COUNT(*) AS PassengerCount
        FROM
          `noble-airport-412610.shashank.titanic_data`
        GROUP BY
          Gender, Survival_Status
      )

      SELECT
        'Gender' AS ChartType,
        Gender AS Category,
        TotalPassengers AS Value
      FROM
        PassengerCounts

      UNION ALL

      SELECT
        'SurvivalByClass' AS ChartType,
        CONCAT('Class ', CAST(Ticket_Class AS STRING)) AS Category,
        PassengerCount AS Value
      FROM
        SurvivalByClass

      UNION ALL

      SELECT
        'AgeDistribution' AS ChartType,
        CASE
          WHEN Passenger_Age < 10 THEN '0-9'
          WHEN Passenger_Age BETWEEN 10 AND 19 THEN '10-19'
          WHEN Passenger_Age BETWEEN 20 AND 29 THEN '20-29'
```

```
        WHEN Passenger_Age BETWEEN 40 AND 49 THEN '40-49'
        WHEN Passenger_Age BETWEEN 50 AND 59 THEN '50-59'
        WHEN Passenger_Age >= 60 THEN '60+'
        ELSE 'Unknown'
      END AS Category,
      PassengerCount AS Value
    FROM
      AgeDistribution

    UNION ALL

    SELECT
      'SurvivalByGender' AS ChartType,
      CONCAT(Gender, ' - ', Survival_Status) AS Category,
      PassengerCount AS Value
    FROM
      SurvivalByGender
  ''',
  use_legacy_sql=False,
  destination_dataset_table=f'{PROJECT_ID}.{STAGING_DATASET}.visualization_of_Survival',
  write_disposition='WRITE_TRUNCATE',
  dag=dag
)
```

Visualization of Survial Output Snapshot:

SCHEMA | DETAILS | PREVIEW | LINEAGE | DATA PROFILE | D/

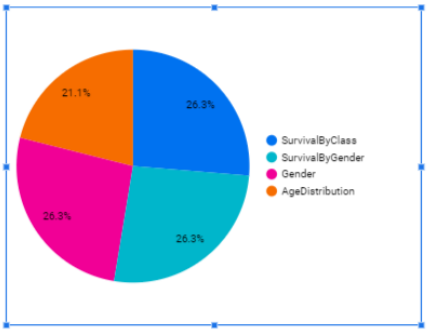
Row	ChartType	Category	Value
1	AgeDistribution	0-9	1
2	AgeDistribution	0-9	1
3	AgeDistribution	0-9	1
4	AgeDistribution	60+	1
5	AgeDistribution	60+	1
6	AgeDistribution	60+	1
7	AgeDistribution	60+	1
8	AgeDistribution	10-19	1
9	AgeDistribution	10-19	1
10	AgeDistribution	20-29	1
11	AgeDistribution	20-29	1
12	AgeDistribution	20-29	1
13	AgeDistribution	30-39	1
14	AgeDistribution	30-39	1

Visualization output:

visualization_of_Survival

	ChartType	Value
1.	SurvivalByClass	891
2.	SurvivalByGender	891
3.	Gender	891
4.	AgeDistribution	714

1 - 4 / 4



11. data_validation_query Function:

Describes the purpose of the function, which is to generate a SQL query for data validation in BigQuery.

The query checks the count of rows in the specified dataset where the Passenger_ID is greater than 0.

_data_validation_pass Function:

Describes the function to be called when data validation passes.

Prints a message indicating that data validation passed.

data_validation_pass:

Creates a PythonOperator task for successful data validation.

Task ID is 'data_validation_pass'.

Calls the _data_validation_pass function and associates it with the specified DAG.

_data_validation_fail Function:

Describes the function to be called when data validation fails.

Prints a message indicating that data validation failed and raises a ValueError.

data_validation_fail:

Creates a PythonOperator task for failed data validation.

Task ID is 'data_validation_fail'.

Calls the _data_validation_fail function and associates it with the specified DAG.

Define a function to generate a BigQuery SQL query for data validation

```python
def _data_validation_query():
    return f"""
        SELECT COUNT(*) as total_rows
        FROM `{PROJECT_ID}.{STAGING_DATASET}.titanic_data`
        WHERE Passenger_ID > 0  -- Adjust the condition as needed
    """

query = _data_validation_query()

bigquery_validation_task = BigQueryCheckOperator(
    task_id='bigquery_validation_task',
    sql=query,
    use_legacy_sql=False,
    dag=dag
)



def _data_validation_pass():
    print("Data validation passed!")

data_validation_pass = PythonOperator(
    task_id='data_validation_pass',
    python_callable=_data_validation_pass,
    dag=dag,
)

def _data_validation_fail():
    print("Data validation failed!")
    raise ValueError("Data validation failed")

data_validation_fail = PythonOperator(
    task_id='data_validation_fail',
    python_callable=_data_validation_fail,
    dag=dag,
)
```

12. End Dummy Task:

The end task is a DummyOperator serving as the endpoint for the DAG. Like the start task, it acts as a placeholder and doesn't perform any significant action.

```python
end = DummyOperator(
    task_id="end_task_id",
    dag=dag
)
```

13. Task Dependencies:

The set_upstream and set_downstream methods define dependencies between tasks. These ensure that tasks are executed in the desired sequence, creating a logical flow for the data processing pipeline.

Define task dependencies

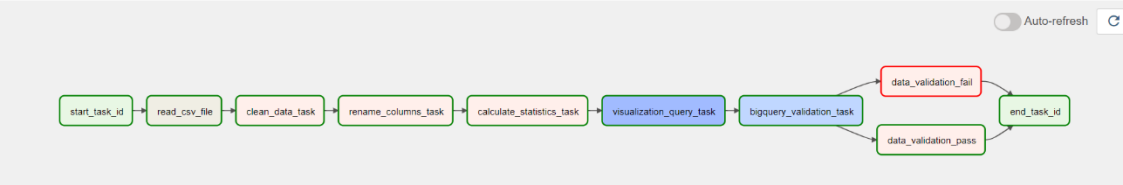
```python
start >> read_csv_file >> clean_data_task >> rename_columns_task >> statistics_task >> visualization_query_task >> bigquery_validation_task

bigquery_validation_task >> data_validation_pass
bigquery_validation_task >> data_validation_fail

data_validation_pass >> end
data_validation_fail >> end
```

Conclusion:

To sum up, the Airflow DAG effectively orchestrates a sequence of tasks dedicated to loading, cleaning, column renaming, statistical calculations, and data preparation for visualization using the Titanic dataset. Each task utilizes designated operators or Python functions to fulfill its role, and the overall DAG configuration ensures a smooth and

automated data pipeline. The modular design provides flexibility and scalability, facilitating straightforward maintenance and extension of the workflow as required.

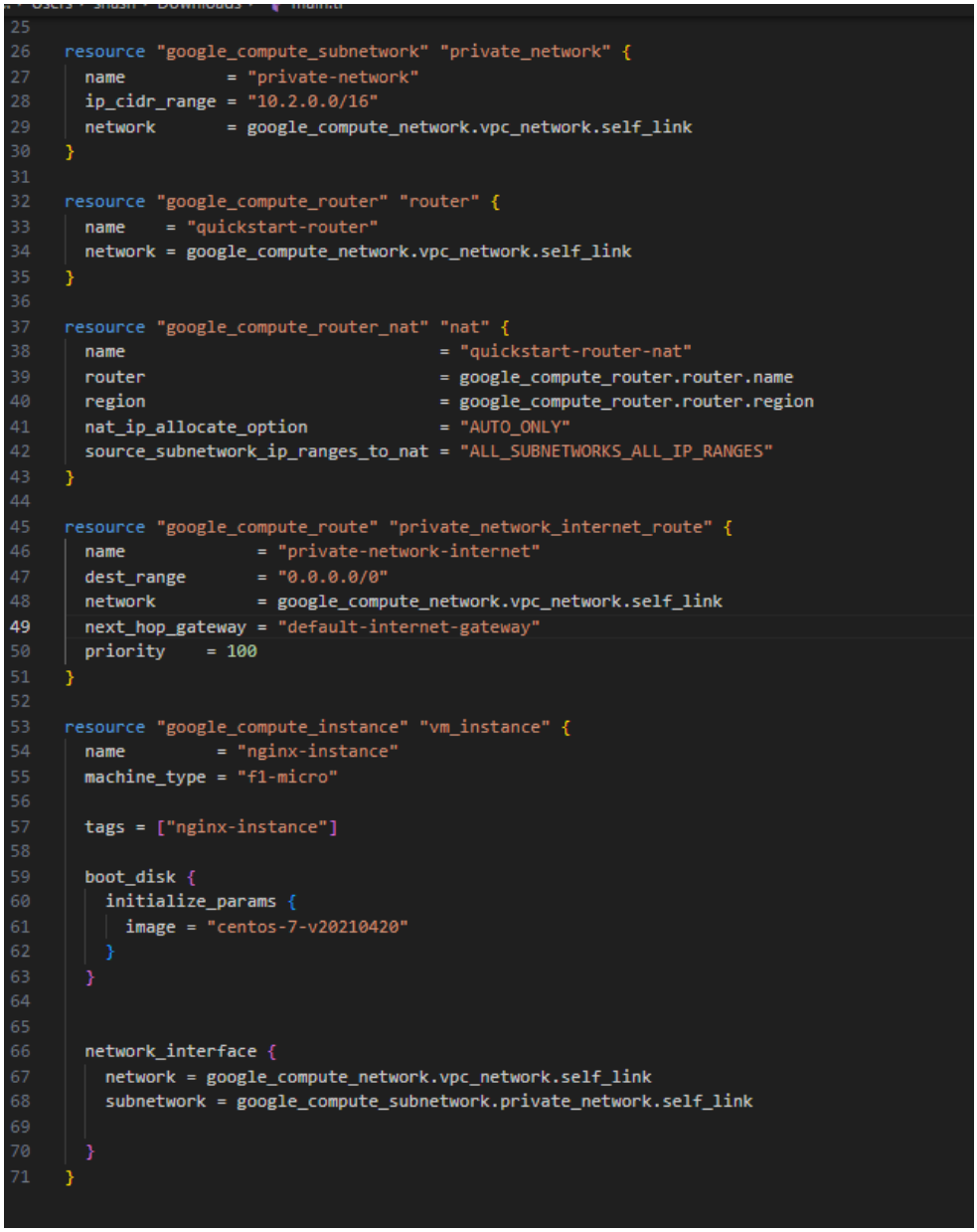
Data pipeline Snapshot:



7.2 Terraform Infrastructure Code

C: > Users > shash > Downloads > main.tf

```
variable "project_id" {
    type = string
}
locals {
  project_id = var.project_id
}
provider "google" {
  project = local.project_id
  region  = "us-central1"
  zone    = "us-central1-b"
}
resource "google_project_service" "compute_service" {
  project = local.project_id
  service = "compute.googleapis.com"
}

resource "google_compute_network" "vpc_network" {
  name                    = "terraform-network"
  auto_create_subnetworks = false
  delete_default_routes_on_create = true
  depends_on = [
    google_project_service.compute_service
  ]
}
```



```
25
26  resource "google_compute_subnetwork" "private_network" {
27    name          = "private-network"
28    ip_cidr_range = "10.2.0.0/16"
29    network       = google_compute_network.vpc_network.self_link
30  }
31
32  resource "google_compute_router" "router" {
33    name    = "quickstart-router"
34    network = google_compute_network.vpc_network.self_link
35  }
36
37  resource "google_compute_router_nat" "nat" {
38    name                               = "quickstart-router-nat"
39    router                             = google_compute_router.router.name
40    region                             = google_compute_router.router.region
41    nat_ip_allocate_option             = "AUTO_ONLY"
42    source_subnetwork_ip_ranges_to_nat = "ALL_SUBNETWORKS_ALL_IP_RANGES"
43  }
44
45  resource "google_compute_route" "private_network_internet_route" {
46    name             = "private-network-internet"
47    dest_range       = "0.0.0.0/0"
48    network          = google_compute_network.vpc_network.self_link
49    next_hop_gateway = "default-internet-gateway"
50    priority    = 100
51  }
52
53  resource "google_compute_instance" "vm_instance" {
54    name         = "nginx-instance"
55    machine_type = "f1-micro"
56
57    tags = ["nginx-instance"]
58
59    boot_disk {
60      initialize_params {
61        image = "centos-7-v20210420"
62      }
63    }
64
65
66    network_interface {
67      network = google_compute_network.vpc_network.self_link
68      subnetwork = google_compute_subnetwork.private_network.self_link
69
70    }
71  }
```

Explanation:

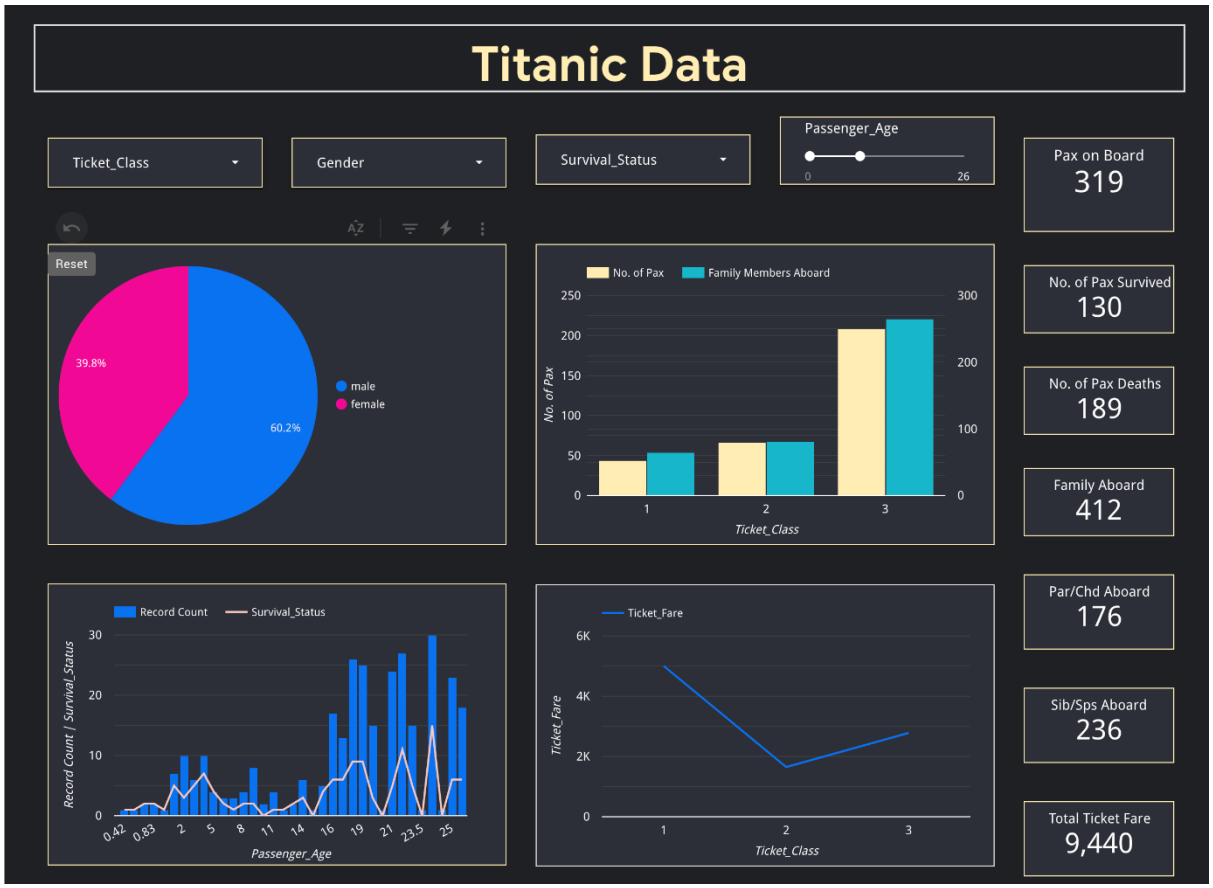
The above Terraform configuration defines infrastructure on Google Cloud Platform (GCP) for a simple virtual private cloud (VPC) setup. Here's an overview of the code:

- **Variable and Locals:**
  - **variable "project_id"**: Declares a variable for the GCP project ID, specifying its type as a string.
  - **locals**: Defines a local variable named **project_id** to store the value of the **project_id** variable.
- **Google Cloud Provider Configuration:**
  - **provider "google"**: Configures the Google Cloud provider with the project ID, region ("us-central1"), and zone ("us-central1-b").

- **Google Project Service:**
  - **resource "google_project_service" "compute_service"**: Enables the Compute Engine API for the specified GCP project.
- **Google Compute Network:**
  - **resource "google_compute_network" "vpc_network"**: Creates a custom VPC network named "terraform-network" with specific configurations. Auto creation of subnetworks is disabled.
- **Google Compute Subnetwork:**
  - **resource "google_compute_subnetwork" "private_network"**: Creates a custom subnetwork named "private-network" with a specified IP CIDR range and associates it with the previously created VPC network.
- **Google Compute Router:**
  - **resource "google_compute_router" "router"**: Defines a router named "quickstart-router" associated with the VPC network.
- **Google Compute Router NAT:**
  - **resource "google_compute_router_nat" "nat"**: Configures Network Address Translation (NAT) for the router to allow instances in the private subnetwork to access the internet.
- **Google Compute Route:**
  - **resource "google_compute_route" "private_network_internet_route"**: Creates a route to direct internet-bound traffic from the private network through the default internet gateway.
- **Google Compute Instance:**
  - **resource "google_compute_instance" "vm_instance"**: Defines a virtual machine instance named "nginx-instance" with a specified machine type ("f1-micro"), tags, and a boot disk with a CentOS 7 image. The instance is connected to the custom VPC network and subnetwork.

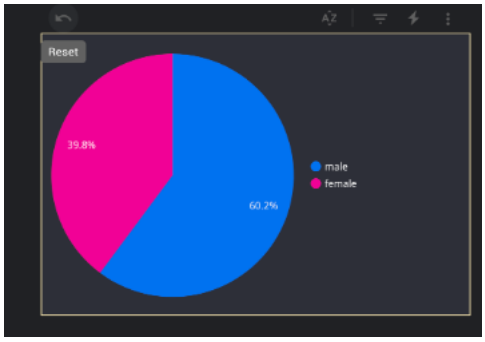
In summary, this Terraform script sets up a custom VPC with a private subnetwork, a router with NAT for internet access, a route for internet-bound traffic, and a virtual machine instance within the private subnetwork. The configuration is focused on creating a basic network infrastructure on Google Cloud Platform.

8.1 Dashboard Overview:



8.2 Explanation of Charts:

1st Graph :



This graph displays the data on male and female percentage on the Titanic ship.

This dashboard is created in a dynamic way.

In this dynamic dashboard we have filters like

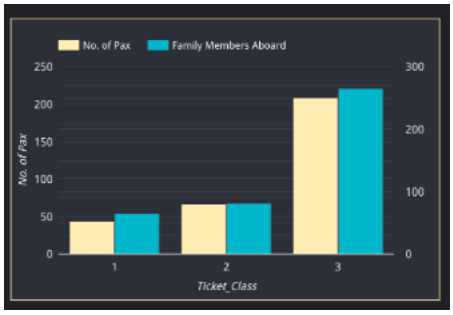
Ticket_Price

Survival_Status

Passenger_Age slider

When cursor is hovered on any colour in the chart it pops up the information of either number of male passengers or female passengers, based on the filters on the dynamic dashboard i.e Ticket_Price ,Survival_Status,Passenger_Age

2nd graph:



This graph displays the statistics on number of Total passengers with the inclusion of the family members in the titanic ship.

Here in the X axis we have No of passenger and in the y axis we have ticket class.

When cursor hovered on the Ticket _class 1 on the no. of. Pax it pops the info of the total individual passengers.

When cursor hovered on the Ticket _class 1 on the family Members Aboard it pops the information of the family members on boarded.

Similarly we can see the readings for ticket class 2 and 3 for the No of pax and family members aboard.

In this dynamic dashboard we have filters like

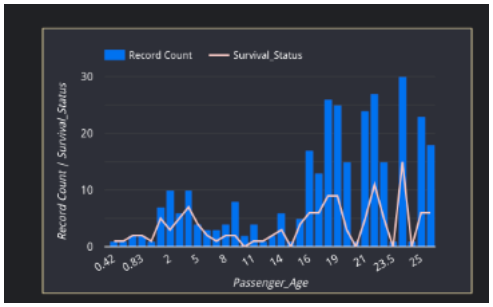
Ticket_Price

Survival_Status

Passenger_Age slider

Where graphs change accordingly to the above filters selected.

3rd graph :



In this graph we have record count / survival_status in the X axis and passenger age slider in the Y axis .

When we hover on this candle sticks graphs it gives the count of the passengers based on the passenger age selected

In this dynamic dashboard we have filters like

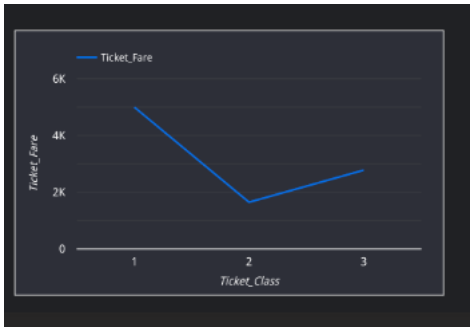
Ticket_Price

Survival_Status

Passenger_Age slider

Where graphs change accordingly to the above filters selected.

4th graph:



In this graph we have Ticket_fare in the x axis and Ticket class in the y axis.

This graph tells the info of the ticket price of the titanic ship based on the Ticket classes like 1st , 2nd and 3rd .

In this dynamic dashboard we have filters like

Ticket_Price

Survival_Status

Passenger_Age slider

Where graphs change accordingly to the above filters selected.

Conclusion:

In conclusion, this project entails the development of a comprehensive data ingestion pipeline utilizing Apache Airflow, Google Cloud Platform (GCP), and Terraform. The pipeline efficiently manages and processes our data. Additionally, efforts were made to create a dashboard aimed at addressing challenges and enhancing our understanding of the data in a more intuitive manner.