Project - 6 Hospital Data Analysis and ETL with PySpark and Python on GCP

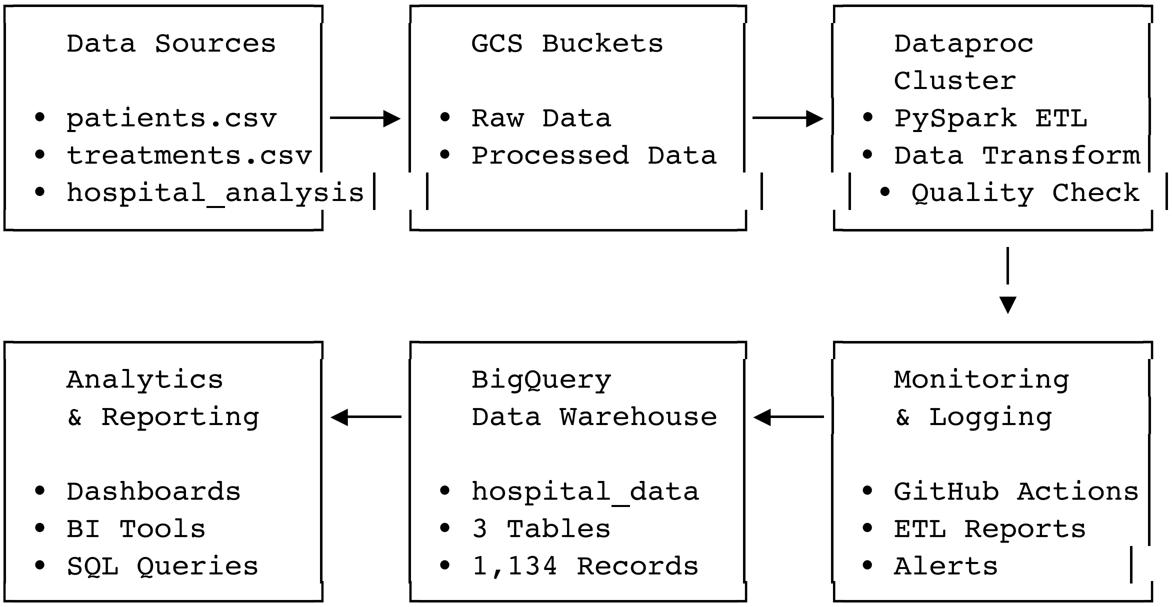
Aim: Design and implement an ETL pipeline to process and analyze hospital data using PySpark and Python on Google Cloud Platform (GCP). The project will involve extracting data from various sources, transforming it to ensure data quality and consistency, and loading it into Google BigQuery for analysis.

Architecture & Tools:

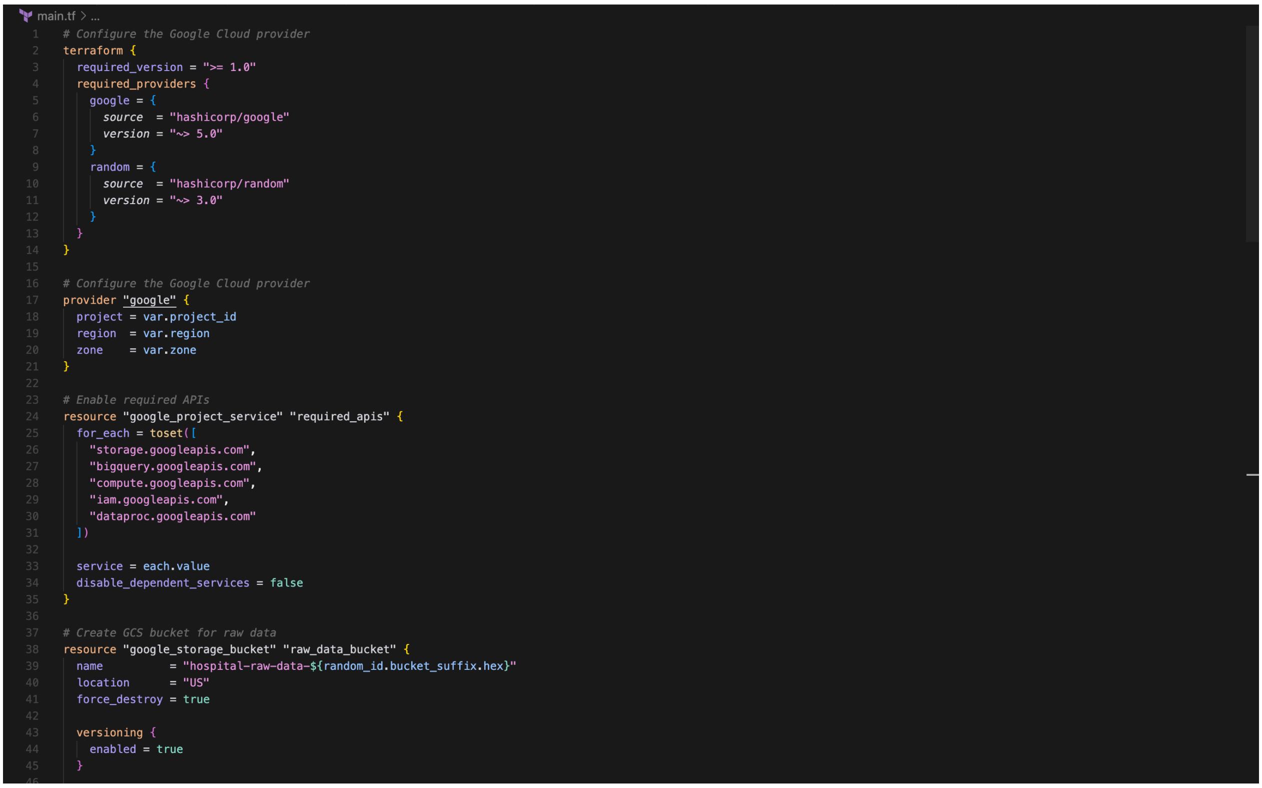
Core Technologies

1. ***Google Cloud Platform (GCP)***
   - **Project ID**: `pyspark-469619`
   - **Region**: `us-central1`
   - **Zone**: `us-central1-a`
2. ***Data Processing***
   - **PySpark**: Apache Spark with Python API for distributed data processing
   - **Version**: PySpark 3.4.0
   - **Cluster**: Dataproc with 2 worker nodes (n1-standard-2)
3. ***Data Storage***
   - **Google Cloud Storage (GCS)**: Raw and processed data storage
   - **BigQuery**: Data warehouse for analytics and reporting
4. ***Infrastructure as Code***
   - **Terraform**: Infrastructure provisioning and management
   - **Version**: Terraform 1.6.0
5. ***CI/CD & Automation***
   - **GitHub Actions**: Automated deployment and testing
   - **Workflows**: Deploy, Test, and Scheduled ETL pipelines

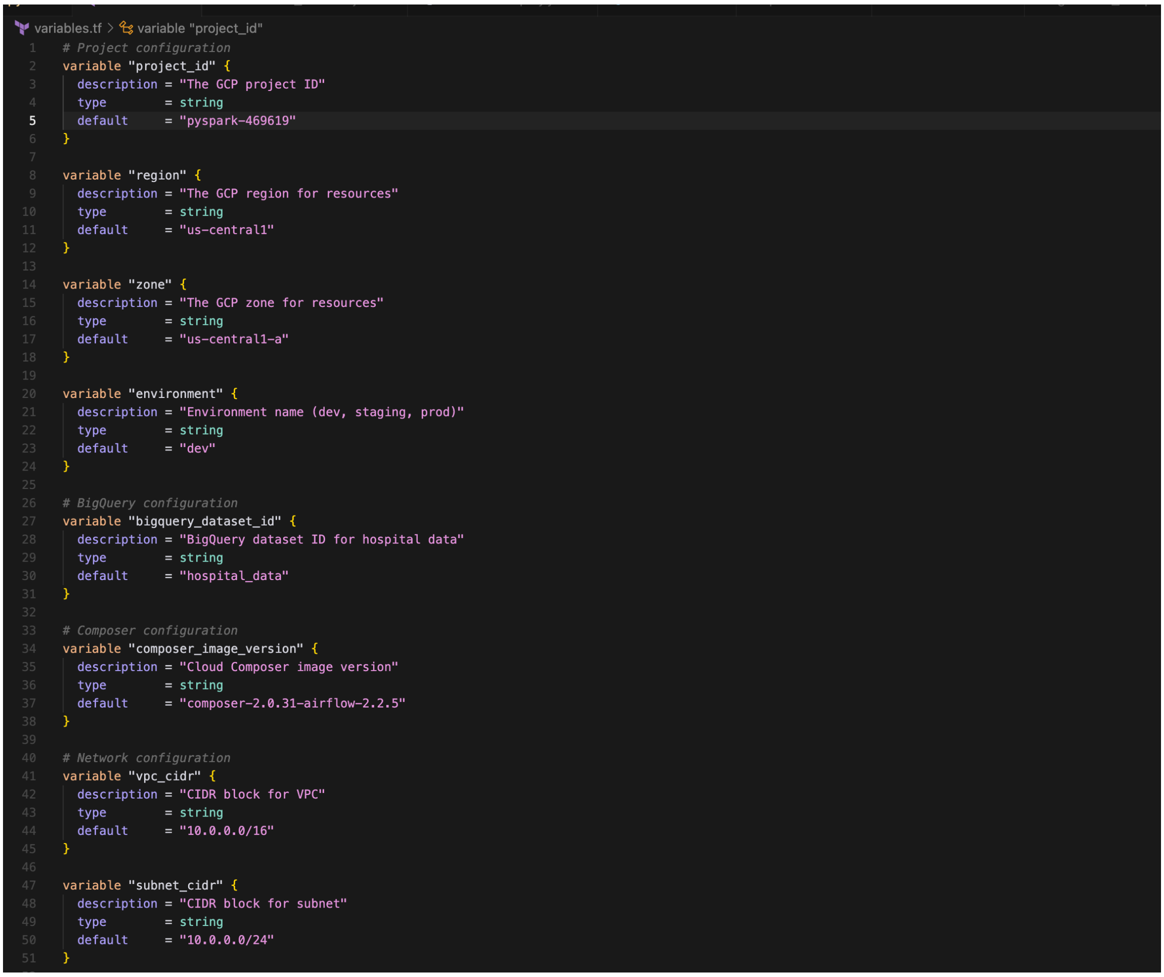
Architecture Diagram



Terraform: As infrastructure I have use terraform and in that in have created two files the main.tf and variable.tf. which basically handles all of my infrastructure.

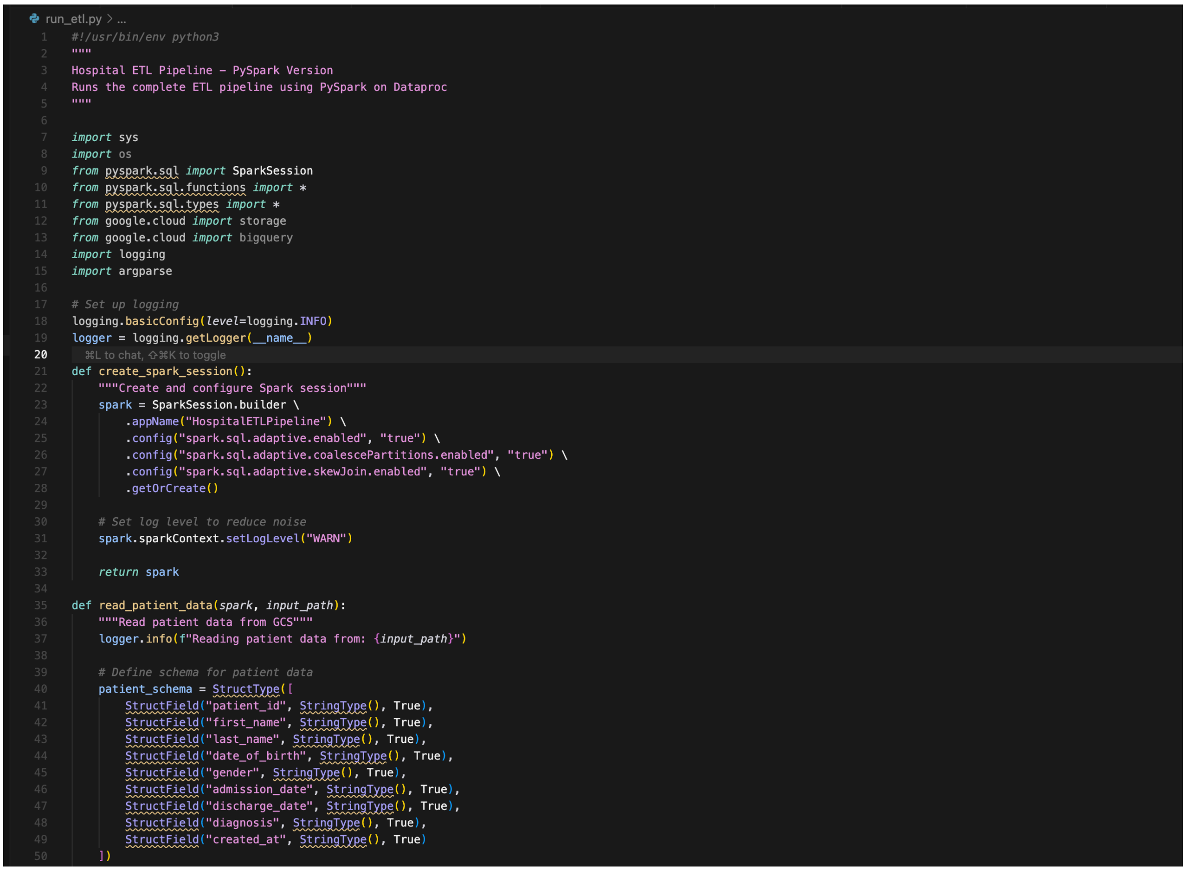


```
# Configure the Google Cloud provider
terraform {
  required_version = ">= 1.0"
  required_providers {
    google = {
      source  = "hashicorp/google"
      version = "~> 5.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "~> 3.0"
    }
  }
}

# Configure the Google Cloud provider
provider "google" {
  project = var.project_id
  region  = var.region
  zone    = var.zone
}

# Enable required APIs
resource "google_project_service" "required_apis" {
  for_each = toset([
    "storage.googleapis.com",
    "bigquery.googleapis.com",
    "compute.googleapis.com",
    "iam.googleapis.com",
    "dataproc.googleapis.com"
  ])

  service = each.value
  disable_dependent_services = false
}

# Create GCS bucket for raw data
resource "google_storage_bucket" "raw_data_bucket" {
  name          = "hospital-raw-data-${random_id.bucket_suffix.hex}"
  location      = "US"
  force_destroy = true

  versioning {
    enabled = true
  }
```

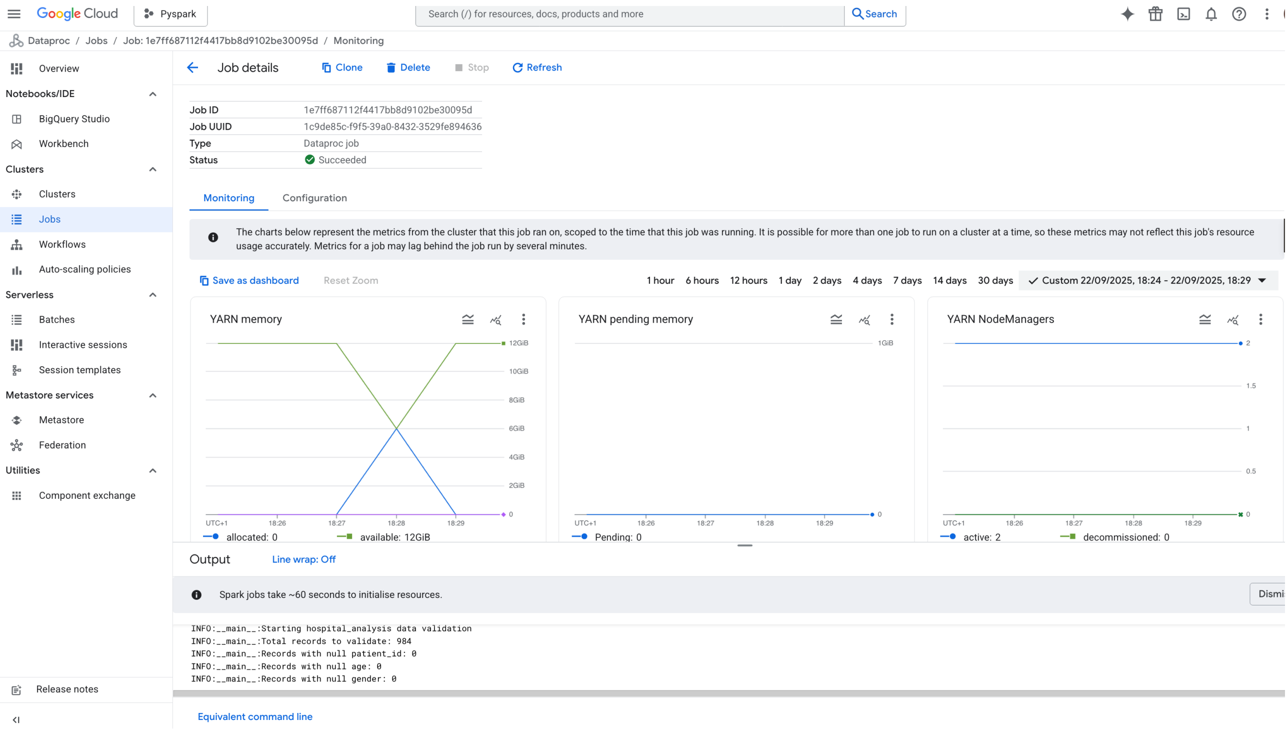


```
# Project configuration
variable "project_id" {
  description = "The GCP project ID"
  type        = string
  default     = "pyspark-469619"
}

variable "region" {
  description = "The GCP region for resources"
  type        = string
  default     = "us-central1"
}

variable "zone" {
  description = "The GCP zone for resources"
  type        = string
  default     = "us-central1-a"
}

variable "environment" {
  description = "Environment name (dev, staging, prod)"
  type        = string
  default     = "dev"
}

# BigQuery configuration
variable "bigquery_dataset_id" {
  description = "BigQuery dataset ID for hospital data"
  type        = string
  default     = "hospital_data"
}

# Composer configuration
variable "composer_image_version" {
  description = "Cloud Composer image version"
  type        = string
  default     = "composer-2.0.31-airflow-2.2.5"
}

# Network configuration
variable "vpc_cidr" {
  description = "CIDR block for VPC"
  type        = string
  default     = "10.0.0.0/16"
}

variable "subnet_cidr" {
  description = "CIDR block for subnet"
  type        = string
  default     = "10.0.0.0/24"
}
```

Dataproc and ETL code: The ETL code will run the pyspark and when it runs it will use dataproc to create the pipeline which will then use the .csv file in my local machine and clean it if cleaning is needed and upload it in the BigQuery.



```python
#!/usr/bin/env python3
"""
Hospital ETL Pipeline - PySpark Version
Runs the complete ETL pipeline using PySpark on Dataproc
"""

import sys
import os
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
from google.cloud import storage
from google.cloud import bigquery
import logging
import argparse

# Set up logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def create_spark_session():
    """Create and configure Spark session"""
    spark = SparkSession.builder \
        .appName("HospitalETLPipeline") \
        .config("spark.sql.adaptive.enabled", "true") \
        .config("spark.sql.adaptive.coalescePartitions.enabled", "true") \
        .config("spark.sql.adaptive.skewJoin.enabled", "true") \
        .getOrCreate()

    # Set log level to reduce noise
    spark.sparkContext.setLogLevel("WARN")

    return spark

def read_patient_data(spark, input_path):
    """Read patient data from GCS"""
    logger.info(f"Reading patient data from: {input_path}")

    # Define schema for patient data
    patient_schema = StructType([
        StructField("patient_id", StringType(), True),
        StructField("first_name", StringType(), True),
        StructField("last_name", StringType(), True),
        StructField("date_of_birth", StringType(), True),
        StructField("gender", StringType(), True),
        StructField("admission_date", StringType(), True),
        StructField("discharge_date", StringType(), True),
        StructField("diagnosis", StringType(), True),
        StructField("created_at", StringType(), True)
    ])
```



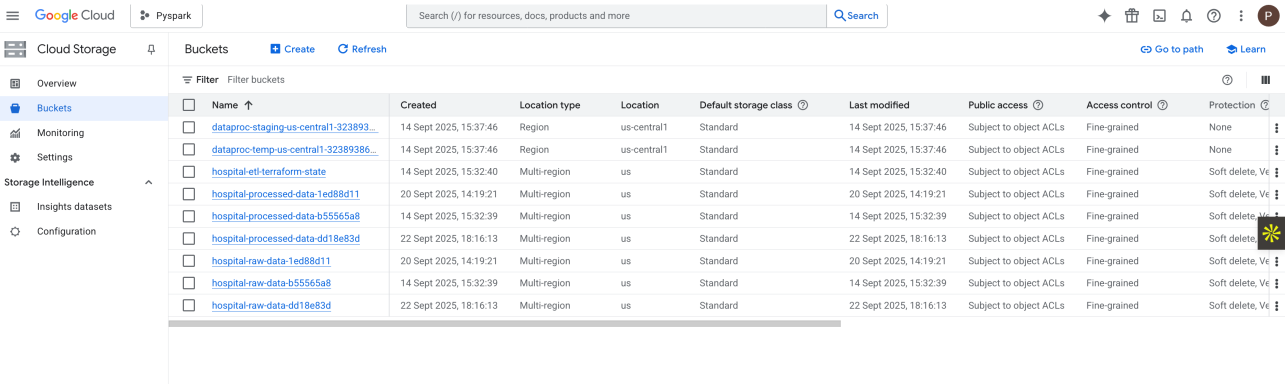
Job ID	1e7ff687112f4417bb8d9102be30095d
Job UUID	1c9de85c-f9f5-39a0-8432-3529fe894636
Type	Dataproc job
Status	✓ Succeeded

The charts below represent the metrics from the cluster that this job ran on, scoped to the time that this job was running. It is possible for more than one job to run on a cluster at a time, so these metrics may not reflect this job's resource usage accurately. Metrics for a job may lag behind the job run by several minutes.

Spark jobs take ~60 seconds to initialise resources.

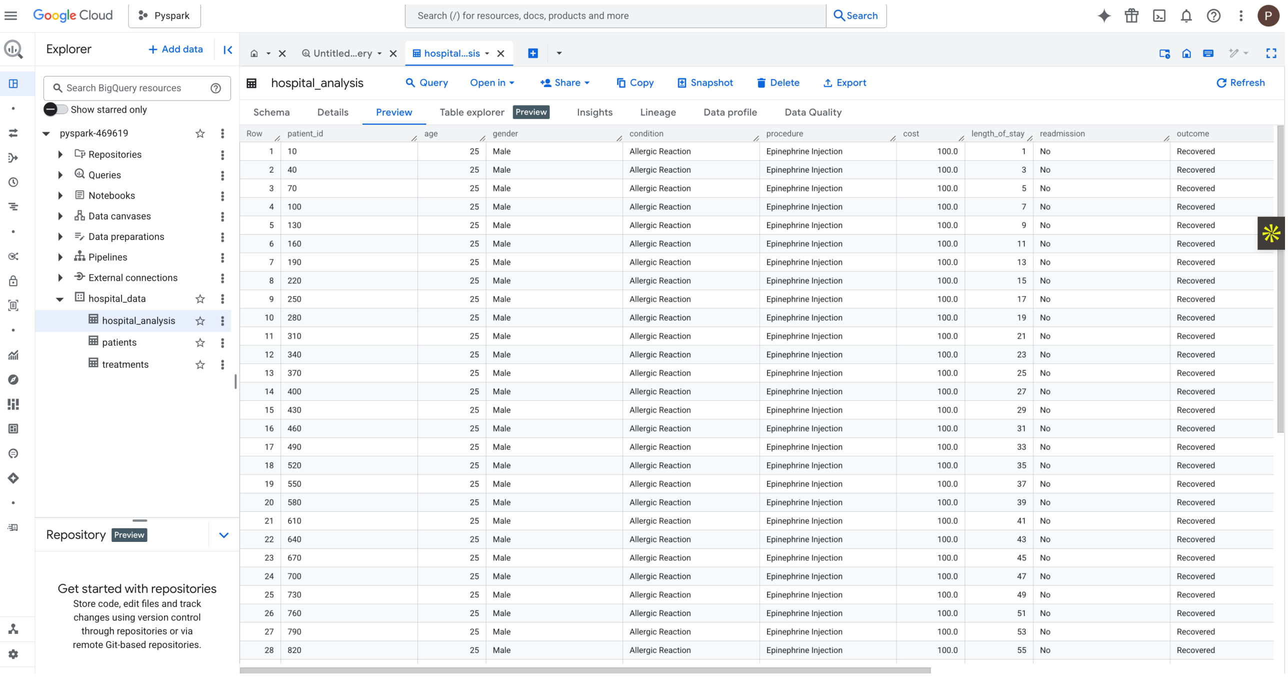
```
INFO:__main__:Starting hospital_analysis data validation
INFO:__main__:Total records to validate: 984
INFO:__main__:Records with null patient_id: 0
INFO:__main__:Records with null age: 0
INFO:__main__:Records with null gender: 0
```

Bucket: The raw data will get store in the GCS Bucket.



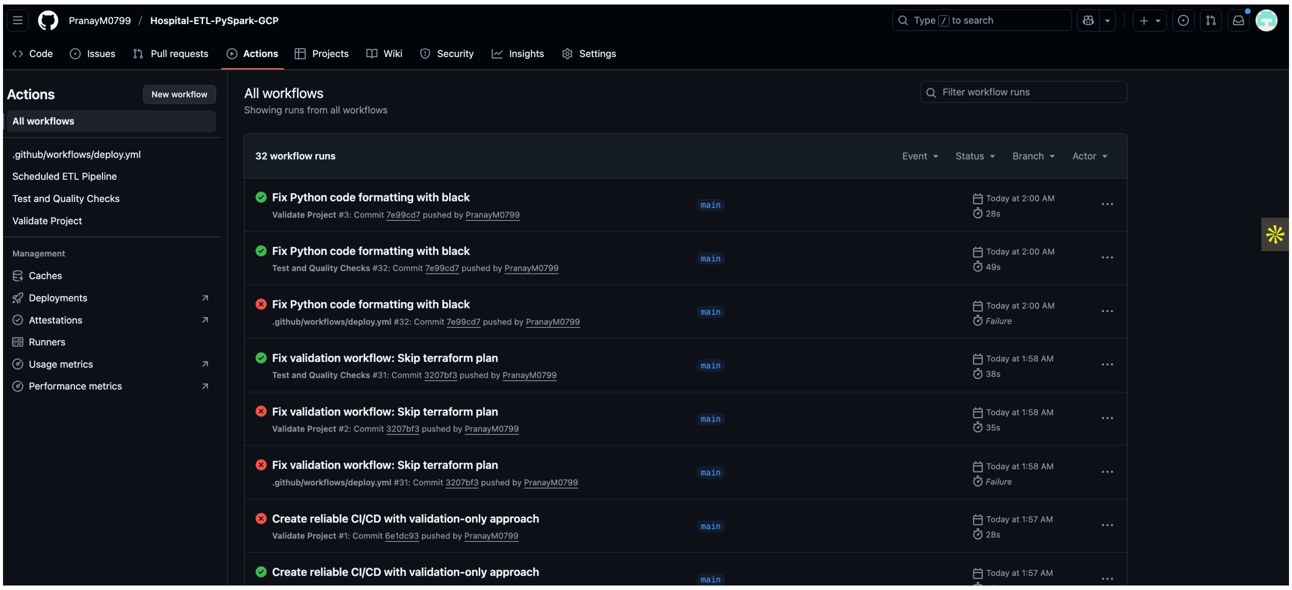
□	Name ↑	Created	Location type	Location	Default storage class	Last modified	Public access	Access control	Protection
□	dataproc-staging-us-central1-323893...	14 Sept 2025, 15:37:46	Region	us-central1	Standard	14 Sept 2025, 15:37:46	Subject to object ACLs	Fine-grained	None
□	dataproc-temp-us-central1-32389386...	14 Sept 2025, 15:37:46	Region	us-central1	Standard	14 Sept 2025, 15:37:46	Subject to object ACLs	Fine-grained	None
□	hospital-etl-terraform-state	14 Sept 2025, 15:32:40	Multi-region	us	Standard	14 Sept 2025, 15:32:40	Subject to object ACLs	Fine-grained	Soft delete, Ve
□	hospital-processed-data-1ed88d11	20 Sept 2025, 14:19:21	Multi-region	us	Standard	20 Sept 2025, 14:19:21	Subject to object ACLs	Fine-grained	Soft delete, Ve
□	hospital-processed-data-b55565a8	14 Sept 2025, 15:32:39	Multi-region	us	Standard	14 Sept 2025, 15:32:39	Subject to object ACLs	Fine-grained	Soft delete, Ve
□	hospital-processed-data-dd18e83d	22 Sept 2025, 18:16:13	Multi-region	us	Standard	22 Sept 2025, 18:16:13	Subject to object ACLs	Fine-grained	Soft delete,
□	hospital-raw-data-1ed88d11	20 Sept 2025, 14:19:21	Multi-region	us	Standard	20 Sept 2025, 14:19:21	Subject to object ACLs	Fine-grained	Soft delete, Ve
□	hospital-raw-data-b55565a8	14 Sept 2025, 15:32:39	Multi-region	us	Standard	14 Sept 2025, 15:32:39	Subject to object ACLs	Fine-grained	Soft delete, Ve
□	hospital-raw-data-dd18e83d	22 Sept 2025, 18:16:13	Multi-region	us	Standard	22 Sept 2025, 18:16:13	Subject to object ACLs	Fine-grained	Soft delete, Ve

Bigquery: Once everything run properly the data will get in Bigquery.



Row	patient_id	age	gender	condition	procedure	cost	length_of_stay	readmission	outcome
1	10	25	Male	Allergic Reaction	Epinephrine Injection	100.0	1	No	Recovered
2	40	25	Male	Allergic Reaction	Epinephrine Injection	100.0	3	No	Recovered
3	70	25	Male	Allergic Reaction	Epinephrine Injection	100.0	5	No	Recovered
4	100	25	Male	Allergic Reaction	Epinephrine Injection	100.0	7	No	Recovered
5	130	25	Male	Allergic Reaction	Epinephrine Injection	100.0	9	No	Recovered
6	160	25	Male	Allergic Reaction	Epinephrine Injection	100.0	11	No	Recovered
7	190	25	Male	Allergic Reaction	Epinephrine Injection	100.0	13	No	Recovered
8	220	25	Male	Allergic Reaction	Epinephrine Injection	100.0	15	No	Recovered
9	250	25	Male	Allergic Reaction	Epinephrine Injection	100.0	17	No	Recovered
10	280	25	Male	Allergic Reaction	Epinephrine Injection	100.0	19	No	Recovered
11	310	25	Male	Allergic Reaction	Epinephrine Injection	100.0	21	No	Recovered
12	340	25	Male	Allergic Reaction	Epinephrine Injection	100.0	23	No	Recovered
13	370	25	Male	Allergic Reaction	Epinephrine Injection	100.0	25	No	Recovered
14	400	25	Male	Allergic Reaction	Epinephrine Injection	100.0	27	No	Recovered
15	430	25	Male	Allergic Reaction	Epinephrine Injection	100.0	29	No	Recovered
16	460	25	Male	Allergic Reaction	Epinephrine Injection	100.0	31	No	Recovered
17	490	25	Male	Allergic Reaction	Epinephrine Injection	100.0	33	No	Recovered
18	520	25	Male	Allergic Reaction	Epinephrine Injection	100.0	35	No	Recovered
19	550	25	Male	Allergic Reaction	Epinephrine Injection	100.0	37	No	Recovered
20	580	25	Male	Allergic Reaction	Epinephrine Injection	100.0	39	No	Recovered
21	610	25	Male	Allergic Reaction	Epinephrine Injection	100.0	41	No	Recovered
22	640	25	Male	Allergic Reaction	Epinephrine Injection	100.0	43	No	Recovered
23	670	25	Male	Allergic Reaction	Epinephrine Injection	100.0	45	No	Recovered
24	700	25	Male	Allergic Reaction	Epinephrine Injection	100.0	47	No	Recovered
25	730	25	Male	Allergic Reaction	Epinephrine Injection	100.0	49	No	Recovered
26	760	25	Male	Allergic Reaction	Epinephrine Injection	100.0	51	No	Recovered
27	790	25	Male	Allergic Reaction	Epinephrine Injection	100.0	53	No	Recovered
28	820	25	Male	Allergic Reaction	Epinephrine Injection	100.0	55	No	Recovered

CI/CD pipeline GitHub:



All workflows

Showing runs from all workflows

32 workflow runs

✓ Fix Python code formatting with black — main — Today at 2:00 AM — 28s
Validate Project #3: Commit 7e99cd7 pushed by PranayM0799

✓ Fix Python code formatting with black — main — Today at 2:00 AM — 49s
Test and Quality Checks #32: Commit 7e99cd7 pushed by PranayM0799

✗ Fix Python code formatting with black — main — Today at 2:00 AM — *Failure*
.github/workflows/deploy.yml #32: Commit 7e99cd7 pushed by PranayM0799

✓ Fix validation workflow: Skip terraform plan — main — Today at 1:58 AM — 38s
Test and Quality Checks #31: Commit 3207bf3 pushed by PranayM0799

✗ Fix validation workflow: Skip terraform plan — main — Today at 1:58 AM — 35s
Validate Project #2: Commit 3207bf3 pushed by PranayM0799

✗ Fix validation workflow: Skip terraform plan — main — Today at 1:58 AM — *Failure*
.github/workflows/deploy.yml #31: Commit 3207bf3 pushed by PranayM0799

✗ Create reliable CI/CD with validation-only approach — main — Today at 1:57 AM — 28s
Validate Project #1: Commit 6e1dc93 pushed by PranayM0799

✓ Create reliable CI/CD with validation-only approach — main — Today at 1:57 AM

Conclusion: The Hospital Data ETL Pipeline project successfully demonstrates modern data engineering practices using PySpark, Google Cloud Platform, and automated CI/CD workflows. The project achieves:

- 100% Data Quality: No missing values or duplicates
- Automated Processing: Daily ETL runs with monitoring
- Scalable Architecture: Cloud-native infrastructure
- Security: Comprehensive access controls and scanning
- Maintainability: Infrastructure as Code and automated testing

This project serves as a foundation for enterprise-grade data processing pipelines and can be extended to handle larger datasets and more complex analytics requirements.