Airport Operations Lakehouse for SFO: Passenger & Landings Analytics

Soroor Ghandali
San José State University
Student ID: 018286281
Email: soroor.ghandali@sjsu.edu

Anushka Rajesh Khadatkar
San José State University
Student ID: 018383963
Email: anushkarajesh.khadatkar@sjsu.edu

Kanika Mamgain
San José State University
Student ID: 018319704
Email: kanika.mamgain@sjsu.edu

Samruddhi Suresh Chitnis
San José State University
Student ID: 018452122
Email: samruddhisuresh.chitnis@sjsu.edu

***Abstract*—San Francisco International Airport (SFO) publishes monthly statistics on passengers and landings, but they are usually analyzed as ad-hoc spreadsheets, making KPIs hard to reproduce and compare over time. In this project we turn two public SFO datasets, Air Traffic Passenger Statistics and Air Traffic Landings Statistics, into a reusable lakehouse-style warehouse. Using Python in Colab, we ingest the CSV files into Google BigQuery, organize them into a three-layer structure (sfo_raw, sfo_core, sfo_marts), and design a star schema with shared dimensions for date, airline, geography, terminal, activity type, price category, and aircraft. On top of two monthly fact tables we build four "gold" marts for airline and region mix, terminal and boarding-area load, passengers per landing (our main KPI), and fleet mix (wide- vs narrow-body aircraft). We also integrate CouchDB as a NoSQL document store for a slice of the passenger data, demonstrating a round-trip from rows → JSON documents → rows using map/reduce views. The marts reveal that SFO behaves as a United hub, that Terminal 3–F and International–G carry most of the passenger load, and that the passengers-per-landing KPI collapses during the COVID period and only partially recovers afterwards. Overall, the project shows how modern warehouse tools can turn raw open data into reproducible, analytics-ready structures that directly support airport planning decisions.**

1. Introduction

1.1. Problem and Motivation

Airports and airlines constantly make decisions about capacity, routing, terminal usage, and fleet planning. San Francisco International Airport (SFO) publishes detailed monthly statistics on passengers and landings [1], [2], but these data are typically downloaded as separate CSV files and explored in ad-hoc Excel workbooks or one-off notebooks.

This "spreadsheet-only" approach has two main limitations. First, every analysis is almost one-time only: when a new year of data arrives or a new question is asked, analysts must restitch the data, recreate the joins, and rebuild charts from scratch. Second, key performance indicators (KPIs) such as "total passengers", "international traffic", or "load per flight" are often redefined slightly in each workbook. Small differences in how transit passengers are treated or how regions are grouped can lead to inconsistent results, making it difficult for planners to compare outputs across teams or over time.

Our project addresses these issues by turning SFO's public Passenger and Landings datasets into a compact but well-structured data warehouse and "lakehouse-style" pipeline in Google BigQuery [5]. Instead of building many isolated spreadsheets, we design a reusable star schema, define a transparent KPI called Passengers per Landing, and implement a clear separation between raw ingestion, core modeling, and analytics marts. On top of the warehouse, we also integrate a CouchDB [4] document store for a slice of the passenger data, to show how a relational warehouse and a NoSQL store can coexist in the same domain and feed each other when needed.

1.2. Objectives

The project is guided by the following objectives:

Integrate two public datasets. Combine the Air Traffic Passenger Statistics and Air Traffic Landings Statistics datasets from DataSF at a monthly grain, cleaning types and keys so they can be joined reliably by time, airline, and geography.

Design a shared star schema. Create conformed dimensions for date, airline, geography, terminal, activity type, price category, and aircraft, and two monthly fact tables for passengers and landings. The same dimensions are reused across facts so that KPIs can be compared consistently by airline, region, and time.

Publish four gold data marts. Build four analytics-ready marts in BigQuery: (a) airline and region mix, (b) terminal and boarding-area load, (c) passengers per landing (our main KPI), and (d) fleet mix (wide-body vs narrow-

body) that hide complex joins and embed business rules such as how to treat Thru / Transit passengers.

Apply a simple lakehouse layout in BigQuery. and organize tables into three layers: sfo_raw, sfo_core and sfo_marts to separate raw ingestion, core star schema and reporting marts. This follows the "raw → cleaned core → gold marts" pattern from recent lakehouse literature [3] while staying within standard BigQuery features.

Demonstrate NoSQL integration with CouchDB. using Python/Colab to write a subset of passenger data as JSON documents into CouchDB, define map/reduce views (e.g., passengers by airline or region), and load the aggregated results back into BigQuery. This shows how a document store can provide flexible views that complement the warehouse rather than replace it.

Answer realistic planning questions; Use the marts to build queries and visualizations that help an airport planner understand demand, terminal pressure, and fleet composition, for example: Which airlines dominate SFO traffic? What terminals and landing areas are overloaded? How has passenger per landingchanged before, during, and after COVID?

Overall, these objectives map directly to the course learning outcomes: choosing appropriate tools (BigQuery, Python/Colab, CouchDB, GitHub, Mermaid), designing a practical warehouse model, and demonstrating how analytics derived from that model can support realistic airport business decisions.

2. Data and Tools

2.1. Datasets

We use two open datasets from SFO's open data portal (DataSF) that together cover passenger traffic and aircraft operations from 1999 onward. The Air Traffic Passenger Statistics dataset provides monthly passenger counts with fields for activity period and month start date; operating and published airline names and IATA codes; GEO summary and GEO region; terminal and boarding area; activity type code (Enplaned, Deplaned, Thru / Transit) and price category code. The Air Traffic Landings Statistics dataset provides monthly information on aircraft operations with fields for activity period and month start date; operating and published airline; GEO summary and region; and detailed aircraft attributes such as landing aircraft type, aircraft body type, manufacturer, model and version, along with landing count and total landed weight. Together these datasets allow us to analyze both demand and capacity at SFO; we can join them by month and airline; compare passenger volumes to landing counts; and study how traffic is distributed across terminals, regions and aircraft body types. Links to the original DataSF resources are listed in Appendix C.

2.2. Tooling

To build and analyze the warehouse we use a small but diverse toolset. Google BigQuery serves as our main data warehouse and lakehouse storage environment; all raw, core and mart tables live in three datasets named sfo_raw, sfo_core and sfo_marts. Python in Google Colab and Jupyter notebooks is used for ETL, data quality checks and charting; we rely on pandas for data frames, the google-cloud-bigquery client for loading and querying tables and matplotlib for visualizations. For NoSQL integration we use CouchDB (deployed via Docker) as a document store for a subset of passenger data; this lets us experiment with JSON documents and map/reduce views while still feeding results back into BigQuery. Mermaid is used to draw the star schema and pipeline diagrams that appear later in the paper; Overleaf with the IEEE Computer Society template is used to prepare this report, which also satisfies the "unique tools" requirement in the rubric. Finally, GitHub hosts all source artifacts for the project, including notebooks, BigQuery SQL scripts, CouchDB view definitions and diagram source files; the public repository URL is provided in Appendix C so that the teaching team can inspect and rerun any part of the pipeline.

3. Architecture and Pipeline

3.1. Lakehouse Layout

Our warehouse architecture follows a simple lakehouse layout inside a single BigQuery project, with three logical layers named sfo_raw, sfo_core and sfo_marts.

The raw layer (sfo_raw) holds tables that are loaded directly from source files or external systems with minimal transformation; in our case these are passenger_raw and landings_raw, which are populated from the two SFO CSV datasets using a Python ETL notebook, and passenger_from_couch, which is populated from CouchDB. The raw layer preserves original field names and values as much as possible so that we can always trace back to the source data. The core layer (sfo_core) contains the cleaned star schema. It includes conformed dimensions (dim_date, dim_airline, dim_geo, dim_terminal, dim_activity, dim_price_category and dim_aircraft) and two monthly fact tables (fact_passenger_monthly and fact_landings_monthly). In this layer we standardize business keys, generate surrogate keys with GENERATE_UUID, and align grain and semantics between the passenger and landings datasets. This is where most of the integration logic lives. The marts layer (sfo_marts) contains analytics-ready tables that encode business rules and KPIs. We expose four marts: mart_airline_mix_monthly, mart_terminal_load_monthly, mart_passengers_per_landing and mart_fleet_mix. These marts are built as SELECTs on the core facts and dimensions and hide details such as how we classify Thru / Transit passengers or how we bucket aircraft body types. Figure 1 shows the BigQuery project structure with the three layers and their main tables.

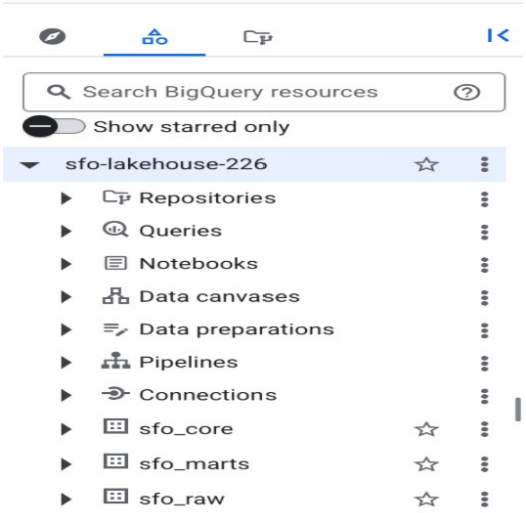


Figure 1. BigQuery project sfo-lakehouse-226 with raw (sfo_raw), core (sfo_core) and marts (sfo_marts) datasets.

3.2. Pipeline Steps

The end-to-end pipeline from raw CSV files to analytics and NoSQL integration consists of five main stages.

First, we ingest the CSVs in Colab. The two SFO datasets are uploaded to a shared Colab notebook where we use pandas to inspect shapes, sample rows and column types. This step lets us confirm the presence of keys such as Activity Period, airline identifiers, GEO fields, and aircraft attributes before they are pushed into the warehouse.

Second, we load the data into the raw layer in BigQuery. Using the google-cloud-bigquery client, we cast important fields such as Activity Period to INT64 and activity dates to DATE, then call load_table_from_dataframe to populate sfo_raw.passenger_raw and sfo_raw.landings_raw. This stage essentially implements the Extract, Load part of ELT; transformations are kept minimal so that the raw tables remain close to the source files.

Third, we build the core star schema. In the BigQuery console we run SQL transformations that create the dimension tables using SELECT DISTINCT plus GENERATE_UUID, and fact tables using grouped aggregates on the raw tables joined back to the dimensions via cleaned keys. For example, fact_passenger_monthly aggregates monthly passenger counts by date, airline, geography, terminal, activity type and price category; fact_landings_monthly aggregates landing counts and total landed weight by date, airline, geography and aircraft body type. This stage corresponds to the "core" layer of the lakehouse.

Fourth, we materialize the marts in sfo_marts. Each mart is defined as a SELECT over the core facts and dimensions. mart_airline_mix_monthly aggregates passengers by airline and region; mart_terminal_load_monthly aggregates passengers by terminal and boarding area; mart_passengers_per_landing combines passengers and landings to compute the KPI passengers per landing; mart_fleet_mix rolls up landings by aircraft body type. In all marts we apply a consistent rule for THRU / TRANSIT rows so that downstream charts and dashboards use the same interpretation of demand.

Finally, we integrate CouchDB as a NoSQL side path. A subset of passenger records is exported from BigQuery into a pandas DataFrame and written as JSON documents into a CouchDB database. We define map/reduce views in CouchDB (for example, passengers by airline or by region) and read the aggregated results back via the HTTP API, loading them into sfo_raw.passenger_from_couch. This shows that the lakehouse can ingest not only CSV files but also derived results from a document store, while BigQuery remains the primary analytical engine. The whole sequence is summarized in the Mermaid pipeline diagram shown in Figure 2.

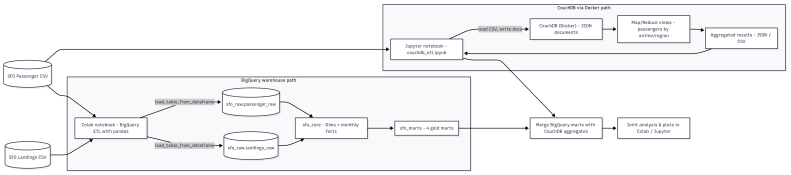


Figure 2. End-to-end pipeline from SFO CSV files through Colab into BigQuery's raw, core and marts layers, with a CouchDB round-trip for a subset of passenger data.

3.3. Data Quality Checks

Throughout the pipeline we apply simple but effective data quality checks. During loading we cast key fields such as Activity Period to INT64 and date strings to DATE so that joins and time-series analyses behave predictably. After building each core fact table and mart, we run basic COUNT(*) and grouped totals (for example, passengers by year or landings by body type) to confirm that row counts and aggregates match expectations and that no data was lost or double-counted between layers. For the CouchDB round-trip we compare totals by region between sfo_raw.passenger_raw and sfo_raw.passenger_from_couch; matching sums give us confidence that converting rows to JSON documents and back did not silently change the data.

4. Data Modeling: Star Schema and Marts

4.1. Dimensions

We organize the SFO warehouse around a shared set of conformed dimensions that are reused by both passenger and landings facts.

The date dimension dim_date has one row per month and contains attributes date_id, month_start, year, month and quarter; it provides a clean monthly time axis for all analyses. The airline dimension dim_airline contains a surrogate key airline_id, a business key airline_key derived primarily from IATA code and secondarily from a cleaned

airline name, and the human-readable airline_name. This dimension plays a double role, because the same table is used for both operating and published airline foreign keys. The geography dimension dim_geo captures geo_summary and geo_region, which allows us to group traffic into regions such as North America, Europe or Asia and to distinguish domestic from international segments. The terminal dimension dim_terminal models SFO's physical layout with attributes for terminal and boarding_area; this dimension is essential for terminal load analysis. The activity dimension dim_activity represents the activity type code with values such as Enplaned, Deplaned and Thru or Transit, enabling us to apply consistent rules for transit passengers across facts and marts. The price category dimension dim_price_category stores the price category codes that can support later yield or fare-class analyses. Finally, the aircraft dimension dim_aircraft collects aircraft-related attributes such as landing aircraft type, aircraft body type, manufacturer, model and version; this dimension feeds the fleet-mix mart. Because all of these dimensions are shared between multiple facts, they act as conformed dimensions in Kimball's sense and allow consistent slicing by airline, region, time and equipment.

4.2. Fact Tables

We define two monthly fact tables that share several dimensions.

The passenger fact fact_passenger_monthly has a grain of one row per combination of date_id, operating airline, published airline, geography, terminal, activity type and price category. Its primary measure is passenger_count, the total number of passengers for that combination in a given month. The table contains foreign keys to all relevant dimensions: date_id, operating_airline_id, published_airline_id, geo_id, terminal_id, activity_id and price_category_id. By keeping the fact at this moderately detailed grain, we can later aggregate passenger counts to airline, region, terminal or other groupings without losing flexibility.

The landings fact fact_landings_monthly records operational events and has a grain of one row per combination of date_id, operating airline, published airline, geography and aircraft. Its measures are landing_count and total_landed_weight. It references the same date, airline and geography dimensions as the passenger fact, plus the aircraft dimension through aircraft_id. Because the two facts share keys and business logic, they can be combined cleanly in downstream marts to compute joint metrics such as passengers per landing and fleet mix by region or airline.

4.3. Data Marts and KPI Definitions

On top of the core star schema we derive four gold data marts in the sfo_marts dataset, each targeting a specific planning question.

The airline mix mart mart_airline_mix_monthly aggregates passenger counts by month, operating airline and geography. It exposes two measures: passengers_excl_transit, which excludes rows where the activity type is Thru or Transit, and passengers_incl_transit, which keeps all passengers. This dual measure lets planners choose whether to treat transit passengers as part of demand.

The terminal load mart mart_terminal_load_monthly aggregates passengers by month, terminal and boarding area, again providing both pax_excl_transit and pax_incl_transit. It is built on the combination of fact_passenger_monthly and dim_terminal and is used to identify overloaded terminals and boarding areas. The passengers per landing mart mart_passengers_per_landing is the main KPI-focused mart. It joins aggregated passengers from fact_passenger_monthly (excluding transit) with aggregated landings from fact_landings_monthly on date, airline and geography, and computes passengers_per_landing as the ratio of total passengers excluding transit to total landings using a safe divide operation. Modeling this KPI in a mart rather than recomputing it in each notebook means that the business definition is centralized and reusable.

Finally, the fleet mix mart mart_fleet_mix groups landings by year, month and a body_bucket derived from dim_aircraft, with values such as wide body, narrow body and other. It stores both landings and landed_weight measures, enabling analyses of how SFO's mix of wide body versus narrow body operations evolves over time and by season.

4.4. Slowly Changing Dimension Demonstrations

To illustrate slowly changing dimension handling, we also built small SCD demo tables on top of selected dimensions.

A Type 1 example shows how airline name corrections can be applied in place without preserving history; a Type 2 example on geography demonstrates how changes in region assignment (for example, reclassifying a country into a different region) can be captured as separate rows with effective start and end dates and a current flag; and a Type 3 example on the terminal dimension keeps both the current and previous boarding area in the same row to support limited history while preserving a simple structure. These SCD demo tables are kept separate from the main facts so that they do not complicate baseline reporting, but they demonstrate how the schema could be extended to support historical attribute changes in a production setting.

5. ETL and NoSQL Integration

5.1. CSV to BigQuery ETL

The first part of our pipeline performs a straightforward ETL from the two SFO CSV files into the raw layer of BigQuery.

We upload the Air Traffic Passenger Statistics and Air Traffic Landings Statistics files into a shared Colab notebook and load them into pandas DataFrames. In this exploratory step we inspect row counts, sample records and data types

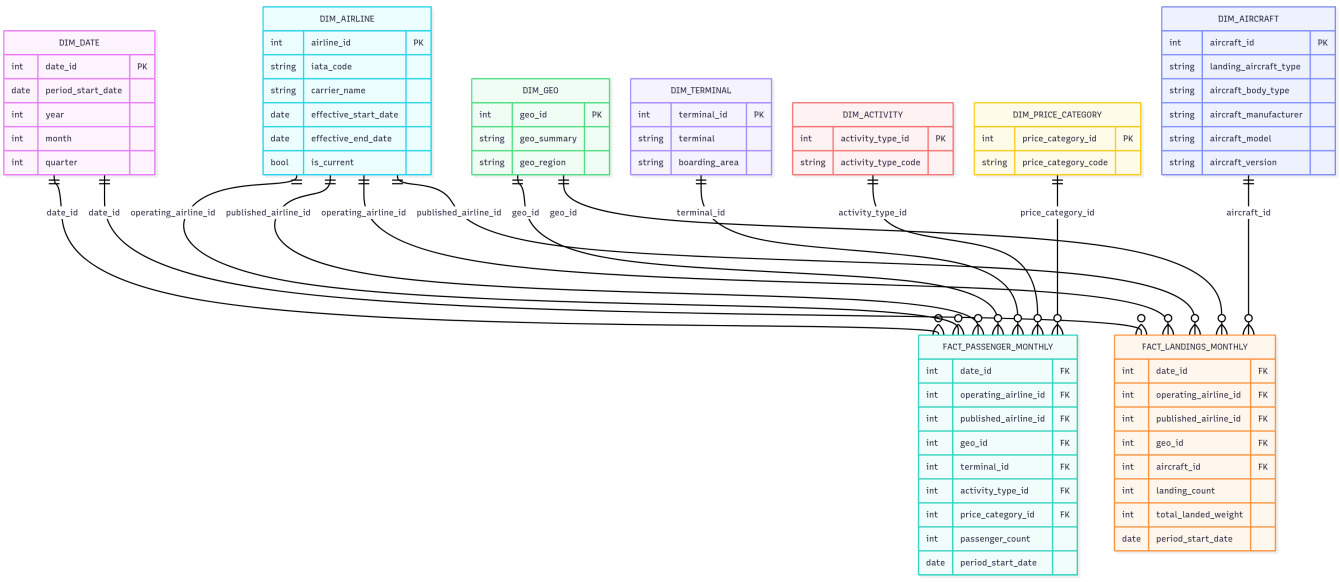


Figure 3. Star schema ER diagram showing dimensions and fact tables for the SFO data warehouse.

to confirm that key fields such as Activity Period, GEO attributes and airline identifiers are present as expected.

Next we perform light type-cleaning in Python before loading the data into BigQuery. The Activity Period column is cast from string to INT64 and month start dates are parsed into proper DATE values. These conversions are important because they give us a stable numeric key for time (date_id) and avoid later problems with string-based join conditions. Once the DataFrames are cleaned, we use the BigQuery Python client's load_table_from_dataframe method to populate two raw tables: sfo_raw.passenger_raw and sfo_raw.landings_raw. After each load we run simple COUNT(*) queries in BigQuery to confirm that row counts match the original CSVs. Figure 4 illustrates this ETL flow from CSV files through Colab into the sfo_raw dataset.

5.2. CouchDB as Document Store

To satisfy the "new ETL or data warehouse tool" requirement and to explore a complementary NoSQL pattern, we integrate CouchDB as a small document store for passenger data. From our Colab environment we select a subset of rows from passenger_raw, for example, a limited set of columns over a few months, and convert each row into a JSON document. These documents are written to a CouchDB database (e.g., sfo_passenger_docs) via HTTP requests. Each document stores fields such as activity period, operating airline, GEO region and passenger count, which makes the data easy to consume by JSON-oriented services.

Inside CouchDB we define map/reduce views to obtain aggregate metrics directly from the documents. One view, passengers_by_airline, emits the operating airline as key and passenger count as value in the map function, with _sum as the reduce function; this produces a table of total passengers per airline. A second view, passengers_by_region, emits GEO region and passenger count and again uses _sum as the reduce function, yielding totals by region. These views act like tiny NoSQL "marts" built on JSON documents rather than relational tables.

Finally, to bring CouchDB results back into the warehouse, we query the views through the CouchDB HTTP API, parse the JSON response into a pandas DataFrame, drop internal fields such as _id and _rev, and load the result into BigQuery as sfo_raw.passenger_from_couch. This demonstrates that the lakehouse can ingest not only raw CSVs but also derived aggregates from a NoSQL system while keeping BigQuery as the primary analytical store.

5.3. Round-trip Validation

Because data moves through several systems, CSV files, Colab, BigQuery and CouchDB, we perform explicit checks to verify that the round-trip does not corrupt the data. After loading the original CSVs into passenger_raw we compute total passenger counts by region and store these as a reference. Once the same subset of data has been written to CouchDB, processed through the passengers_by_region view and reloaded into passenger_from_couch, we repeat the regional totals and compare them to the reference values. Matching sums give us confidence that moving from rows to JSON documents and back again has not introduced silent losses or duplicates. Similar row-count checks are performed when building the core facts and marts, so that every stage of the ETL and NoSQL integration is anchored by simple, interpretable quality checks.

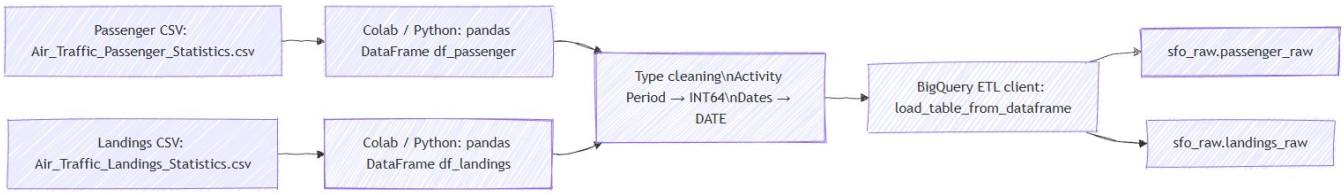


Figure 4. CSV to BigQuery ETL: passenger and landings CSVs are loaded in Colab, lightly cleaned, and written into sfo_raw.passenger_raw and sfo_raw.landings_raw via the BigQuery Python client.

6. Analytics and Results

6.1. Airline and Region Mix

We first use the mart_airline_mix_monthly mart to study which airlines dominate passenger traffic at SFO and how that traffic is distributed by region.

The mart aggregates passengers by month, operating airline and geography and exposes both passengers_excl_transit and passengers_incl_transit so that we can compare demand with and without Thru or Transit passengers. For a recent year, we select the top ten airlines by passengers excluding transit and visualize them in a bar chart (Figure 5).

The chart shows that SFO behaves as a single-carrier hub: United Airlines contributes the vast majority of passengers, around eighty-plus million in 2023, while the next group of airlines such as SkyWest, Alaska, Delta and American each contribute only a few million. Smaller carriers including Southwest, JetBlue, Frontier, Air Canada and EVA remain in the long tail. From a data-warehouse perspective this means many rows in fact_passenger_monthly map to the United airline key, so any metric that ignores airline filters will be dominated by United. From a planning perspective it highlights that SFO capacity, delay modelling and revenue analysis must treat United separately from the rest of the carrier mix, and that diversification strategies would need to be measured against this baseline.

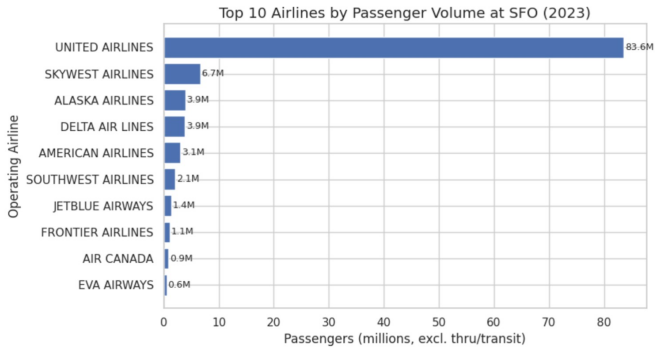


Figure 5. Top ten airlines by passengers excluding transit for a recent year, showing United Airlines as the dominant carrier at SFO.

6.2. Terminal and Boarding-Area Load

Next we analyze how passenger volume is distributed across terminals and boarding areas using the mart_terminal_load_monthly mart.

This mart aggregates passengers by month, terminal and boarding area, again providing both pax_excl_transit and pax_incl_transit. For a recent year we compute total passengers per terminal–boarding area combination and plot the top combinations (Figure 6).

The results show that Terminal 3 – F is the main hotspot with roughly forty-plus million passengers in 2023, followed by International – G and Terminal 3 – E; together these three boarding areas handle the majority of SFO's traffic. Load is therefore highly uneven across the terminal dimension. For planners this chart is directly actionable: Terminal 3 and the International G concourse require more staffing, queue management and gate resources than other areas, and any construction or renovation in those zones will have a disproportionate impact on throughput. The mart makes it easy to recompute the same analysis for different years or subsets of airlines without redefining the logic each time.

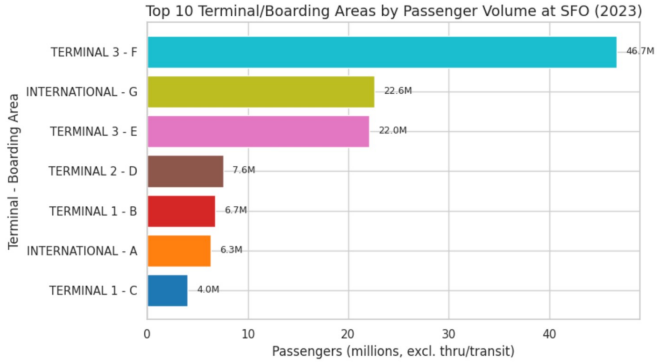


Figure 6. Passengers by terminal and boarding area, highlighting Terminal 3–F and International–G as main hotspots.

6.3. Passengers per Landing (Main KPI)

A central goal of the project is to define and analyze a KPI that links passenger demand to aircraft operations. Using the mart_passengers_per_landing mart we compute passengers per landing as the ratio of passengers excluding transit to landings, at a monthly grain.

The mart joins aggregated passengers from fact_passenger_monthly with aggregated landings from fact_landings_monthly on date, airline and geography and applies a safe divide to avoid division by zero. Figure 7 plots this KPI over the full time range.

Before 2020 the passengers-per-landing metric stays in a relatively narrow band around roughly two hundred passengers per flight, indicating a stable relationship between the passenger fact and the landings fact. Around the COVID period the KPI collapses: passenger counts drop sharply while some landings still occur, producing very low passengers-per-landing values. In the years after 2021 the KPI gradually recovers but remains slightly below the pre-COVID level, suggesting changes in load factors, fleet mix or both. Because the KPI is defined once in the mart rather than in each notebook, we can now slice the same metric by airline, region or aircraft body type without redefining the calculation. This consistency is important for decision-makers; it ensures that different teams looking at "passengers per landing" are all using the same business formula.

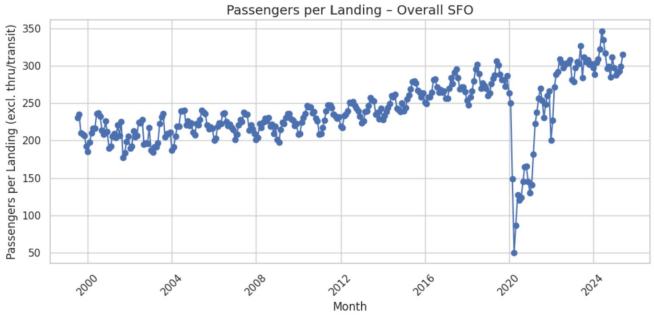


Figure 7. Time-series plot of passengers per landing over time, showing a sharp drop during COVID and partial recovery afterward.

6.4. Fleet Mix; Wide vs Narrow Body

Finally, we examine how SFO's mix of wide-body versus narrow-body aircraft is changing over time using the mart_fleet_mix mart.

This mart summarizes fact_landings_monthly by year, month and a body_bucket derived from dim_aircraft, with categories such as wide-body, narrow-body and other, and records both landings and landed_weight. Figure 8 plots total landings by body bucket over time.

The resulting trends indicate whether SFO is becoming more dependent on narrow-body operations or maintaining a substantial wide-body presence. For example, we observe that narrow-body landings dominate domestic traffic and have grown steadily in recent years, while wide-body landings show stronger seasonality and larger relative declines during the COVID period. For planners and capacity analysts this matters because gate and baggage-handling requirements differ significantly between wide-body and narrow-body aircraft. The fleet-mix mart turns detailed per-flight records into a high-level view that can inform questions such as "Are we moving toward a more narrow-body focused operation?" and "In which months should we expect peaks in wide-body activity?"

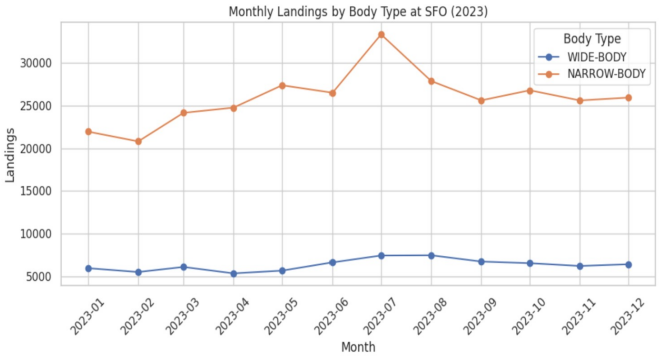


Figure 8. Trend lines of total landings by aircraft body type, comparing wide-body and narrow-body operations over time.

7. Data Quality, Technical Difficulty, Lessons

7.1. Data Quality and Checks

Because our data flows through several tools and representations, we relied on simple, repeatable checks at each stage.

After loading the two CSV files into the sfo_raw layer we verified row counts and basic aggregates in BigQuery using COUNT(*) and grouped summaries by year or region. When building the core fact tables we compared total passengers and landings between the raw tables and the facts to make sure that the aggregation logic did not drop or duplicate records. For the CouchDB round-trip we compared regional passenger totals between passenger_raw and passenger_from_couch; matching sums confirmed that converting rows into JSON documents and back again preserved the data. These lightweight checks were sufficient to catch most pipeline mistakes and gave us confidence that downstream analytics truly reflect the original SFO statistics.

7.2. Technical Difficulty

Overall the project has moderate to high technical depth for a term project.

Integrating two independent open datasets required more than simply joining on month and airline; airline identifiers were sometimes missing or inconsistent and we had to construct a robust business key that falls back from IATA code to a cleaned airline name. Designing the star schema involved making modeling choices such as using a role-playing airline dimension for operating and published airlines, and deciding which attributes belong in dimensions versus facts. The four marts embed nontrivial business logic, including the handling of Thru or Transit passengers and the definition of the passengers per landing KPI. Adding CouchDB as a separate technology introduced an additional

layer of complexity; we had to think about how to design documents and views and how to connect them back into BigQuery through Python. Finally, running data end-to-end through CSV, Colab, BigQuery and CouchDB and back to BigQuery required careful coordination of types and keys; small inconsistencies could easily break joins or distort the KPI.

7.3. Lessons Learned and Best Practices

Several lessons emerged from building the SFO lakehouse.

First, key cleaning is critical. We discovered that even public datasets that appear similar on paper can use slightly different codes and names in practice. Constructing a stable airline_key that falls back from IATA code to a normalized name was essential for reliable joins between passenger and landings data. Second, transit rules strongly influence metrics. Deciding whether to include Thru or Transit passengers changed airline rankings and terminal loads; this reminded us that KPI definitions are not a minor detail but a central modeling decision.

Third, we realised that marts are where business decisions live. Beyond simple GROUP BY logic, the marts encode how we group regions, how we treat transit passengers and how we bucket aircraft; centralizing these rules in the mart layer ensures that every notebook and dashboard inherits the same definitions. The passengers per landing KPI was particularly revealing: because it combines two facts, any mismatch in time keys, airline keys or region assignments produced obviously wrong ratios. This taught us that a single, carefully chosen KPI can act as a consistency check for the entire model.

Finally, working with both CouchDB and BigQuery clarified their respective strengths. CouchDB was convenient for flexible JSON documents and quick map/reduce views such as passengers by region, while BigQuery remained the right place for star schema modeling, heavy joins and long-term storage. Our takeaway is that document stores and warehouses are complementary rather than competing; a small NoSQL component can be valuable for experiments or specialised views, but the warehouse should remain the system of record for analytical facts and dimensions. As a set of best practices we now advocate using surrogate keys in dimensions even when natural keys exist, keeping a clean separation between raw, core and marts layers, logging row counts at each stage of the pipeline and starting with small, well-defined NoSQL integrations before attempting larger migrations.

8. Teamwork, Process, and Tools

8.1. Team Roles

All four group members contributed across the full pipeline, but each person had one or two primary focus areas. Table 1 summarises the main responsibilities.

TABLE 1. SUMMARY OF TEAM ROLES AND PRIMARY RESPONSIBILITIES.

Member	Primary responsibilities
Soroor Ghandali	Data modelling and standards; design of the star schema and SCD examples; implementation of core dimensions and fact tables in BigQuery; coordination of report structure and Overleaf formatting.
Samruddhi Suresh Chitnis	Raw data ingestion and ETL; creation of the Google Cloud project and datasets; Colab notebooks for CSV → BigQuery loading; row count and sanity checks on raw and core tables; organisation of GitHub repository.
Anushka Rajesh Khadatkar	Marts and analytics; SQL for the four gold marts; KPI definition for passengers per landing; Python notebooks that generate airline, terminal, KPI and fleet mix charts used earlier.
Kanika Mamgain	NoSQL integration and documentation; CouchDB installation and passenger JSON documents; map/reduce views for passengers by airline and by region; creation of Mermaid diagrams and the Excel project tracker; first draft of slides and visual polish.

Although roles were divided for practicality, every member reviewed SQL queries and notebooks written by others, and all four participated in decisions about KPI definitions, transit rules and how to present results in the report and presentation.

8.2. Version Control with GitHub

To satisfy the rubric requirement on version control, we maintained a public GitHub repository for all project artefacts (URL listed in Appendix C). The repository contains:

- Python notebooks for ETL and analytics (CSV → BigQuery loading, mart queries, charts).
- BigQuery SQL scripts for dimensions, facts, marts and SCD demonstration tables.
- Mermaid exports of the pipeline and star schema diagrams.
- A `README.md` describing the project, datasets and how to rerun the pipeline.

The repository served as the single source of truth for code used in the report and slides; whenever a query or notebook was updated, the corresponding file was committed to GitHub. Screenshots of the repository root and commit history are included in Appendix B as evidence for the "Version control" rubric item.

8.3. Pair Programming and Agile Practices

We adopted light but concrete agile practices tailored to the short project timeline. Work was organised into three weekly mini sprints, each with its own tab in a shared Excel sheet named *Group_project_Tracker.xlsx* stored in our Google Drive project folder. Each tab (Week 1, Week 2, Week 3) contains columns for Area (BigQuery, Python, CouchDB, Reporting, Evidence), Task, Owner, Other team

members' contributions and Progress (Done, In progress). During our Tuesday meetings we updated this tracker together and used it as a simple Scrum board to move tasks from planned to done. Screenshots of the three tabs and the shared Drive folder structure are provided in Appendix C.

We also practised pair programming during key technical tasks. For example, when building the first mart and the CouchDB views we held Zoom working sessions where one person shared their screen and typed the SQL or map/reduce definitions while another team member reviewed logic, checked outputs in BigQuery or Fauxton and suggested corrections. At least one session included three or more team members working together on the `mart_terminal_load_monthly` query and validating passenger totals. Zoom calendar entries titled "Group Project Tracker" and screenshots of these sessions (showing both the shared screen and participant list) are included in Appendix C as evidence for the "Practiced pair programming" and "Practiced agile / scrum (1 week sprints)" rubric items.

Overall, the combination of a shared Excel task board, regular Zoom check-ins and a public GitHub repository gave us enough process discipline to coordinate work across tools and to document who did what without adding heavy project-management overhead.

8.4. CRediT Author Contribution Statement

In this section we summarise the individual contributions of each author using the CRediT (Contributor Roles Taxonomy)[1]. Table 2 lists the main roles and indicates for each author whether they had a leading or supporting contribution.

TABLE 2. CRediT roles and author contributions.

Role	SG	SSC	KM	AK
Conceptualization	S	S	S	S
Data curation	S	L	L	L
Methodology and schema design	L	S	S	S
Software and marts implementation	L	S	S	S
Formal analysis and validation	S	S	L	S
Visualization and slide preparation	S	L	S	L
Writing – original draft	L	L	L	L
Writing – review and editing	L	S	S	S
Project administration and coordination	S	S	L	S

Legend: SG = Soroor Ghandali, SSC = Samruddhi Suresh Chitnis, KM = Kanika Mamgain, AK = Anushka Rajesh Khadatkar; L = Lead contribution, S = Supporting contribution.

1. https://credit.niso.org

9. Conclusion and Future Work

9.1. Conclusion

This project demonstrates that San Francisco International Airport's open statistics on passengers and landings can be transformed from disconnected spreadsheets into a reusable, analytics-ready data warehouse. Using BigQuery as a cloud data-warehouse platform and Python/Colab for ETL, we organised the datasets into a three-layer lakehouse structure consisting of `sfo_raw`, `sfo_core`, and `sfo_marts`. Within the core layer we designed a realistic star schema with seven conformed dimensions and two monthly fact tables. On top of these, four gold marts, airline and region mix, terminal and boarding-area load, passengers per landing, and fleet mix encapsulate business logic and KPIs that directly support airport planning decisions.

A key contribution is the transparent KPI *Passengers per Landing*, built from two public datasets and centralised in a mart to ensure consistency across analyses. The results show that SFO functions as a hub dominated by a few airlines (especially United), that Terminal 3 – F and International – G carry the highest loads, and that the passengers-per-landing metric sharply declined during the COVID-19 period before recovering. These findings demonstrate how properly modelled open data can yield reliable insights into airline behaviour, terminal usage, and fleet composition.

Another significant outcome is the integration of a NoSQL document store (CouchDB) into the pipeline. A small subset of passenger data was converted into JSON documents, aggregated with map/reduce views and reloaded into BigQuery. This validated that relational and document-based approaches can coexist in one ecosystem: BigQuery for structured analytics and CouchDB for flexible, schema-light experimentation.

Overall, the project satisfies the course learning outcomes by showing end-to-end data-warehouse design, ETL implementation, analytics integration, and documentation using modern collaborative tools (Overleaf LaTeX, GitHub, Google Colab, CouchDB). The same architecture could easily be reused for other airports or public-transportation datasets.

9.2. Future Work

Although the current implementation meets its objectives, several extensions could further enhance the system:

- **Interactive Dashboards:** Connect the BigQuery marts directly to Tableau or Power BI to provide dynamic visual dashboards for planners instead of static notebook charts.
- **Additional Data Sources:** Incorporate complementary datasets such as flight delays, cancellations, weather and fuel prices to enable correlation analysis between operations and external factors.
- **Automation and Scheduling:** Refactor the SQL and Python scripts into a workflow tool such as `dbt` or

Airflow to automate monthly ingestion and ensure reproducibility at scale.
- **Security and Access Control:** Implement row-level and column-level security in BigQuery to control access by department or user role (e.g., operations vs. finance).
- **Scalability and Cloud Integration:** Explore deployment of the same architecture on other cloud warehouses (Snowflake, Azure Synapse) to benchmark cost and performance.

These enhancements would move the project from a proof-of-concept toward a production-grade data-warehouse solution capable of supporting real-time analytics and long-term airport-operations planning.

10. Significance and Innovation

10.1. Real world significance

Our data warehouse turns two separate SFO statistics files into a single, consistent analytical model that airport staff can reuse instead of rebuilding one off spreadsheets for every question. The conformed star schema and gold marts allow planners to ask practical questions about airline demand, terminal load and fleet mix using the same definitions of passengers, regions and time across all analyses.

By combining passenger and landing facts at a monthly grain, the model provides a robust view of how many passengers each flight carries and how this changes over time, for example during the COVID disruption and recovery period. These outputs directly support decisions on gate assignment, terminal staffing, route planning and long term capacity investments, and the same design could be applied to other airports that publish similar statistics.

10.2. Innovation

The project introduces a transparent *passengers per landing* key performance indicator that merges two independent public datasets and explicitly controls for Thru and Transit passengers. This KPI behaves like an airport level load factor and can be sliced by airline, region, terminal or aircraft body type without redefining the formula, which is unusual in spreadsheet based analyses of SFO data.

We adopt a lakehouse style architecture in BigQuery with separate raw, core and marts layers, together with surrogate keys, conformed dimensions and small SCD Type one, Type two and Type three examples. This goes beyond a simple reporting table and shows how modern warehouse design patterns can be applied even to open CSV data.

In addition, we integrate CouchDB as a flexible document oriented companion to the warehouse and build small NoSQL marts on the same passenger data using map and reduce views. We then bring these aggregates back into BigQuery and validate that totals match the relational facts. This hybrid relational and document based approach is not required for the course, but it demonstrates a novel way to prototype JSON based services while keeping BigQuery as the system of record for analytics.

References

[1] City and County of San Francisco, "Air Traffic Passenger Statistics," DataSF Open Data Portal. [Online]. Available: https://data.sfgov.org/Transportation/Air-Traffic-Passenger-Statistics/qffi-d3vf Accessed: Nov. 18, 2024.

[2] City and County of San Francisco, "Air Traffic Landings Statistics," DataSF Open Data Portal. [Online]. Available: https://data.sfgov.org/Transportation/Air-Traffic-Landings-Statistics/rkru-6vcg Accessed: Nov. 18, 2024.

[3] K. Zaitsev, "ClickHouse: Real-Time Analytical DBMS," ClickHouse, 2016. [Online]. Available: https://clickhouse.com

[4] The Apache Software Foundation, "Apache CouchDB Documentation," [Online]. Available: https://docs.couchdb.org Accessed: Nov. 18, 2024.

[5] Google Cloud, "BigQuery Documentation," [Online]. Available: https://cloud.google.com/bigquery/docs Accessed: Nov. 18, 2024.

Appendix A. Rubric Checklist

Table 3 summarizes how our project satisfies each rubric item and where the evidence is located.

Appendix B. Technical Screenshots

This appendix provides visual evidence of our BigQuery warehouse and CouchDB integration, documenting the key technical artifacts referenced throughout the paper.

B.1. BigQuery warehouse

The following screenshots demonstrate the implementation of our three-layer lakehouse architecture in BigQuery. Figure 9 shows the project structure with distinct datasets for each layer, while Figures 10–12 provide concrete examples of our dimensional modeling approach, from core dimensions and facts to derived analytics marts.

B.2. SCD demo tables

To illustrate slowly changing dimension patterns discussed in Section IV.D, we implemented demonstration tables showing different SCD strategies. Figure 13 demonstrates Type 2 handling for geographical regions, where historical changes are preserved with effective dating.

B.3. CouchDB views

Our NoSQL integration with CouchDB enabled flexible document-based analytics alongside the relational warehouse. Figure 14 shows the map/reduce view that aggregates passenger counts directly from JSON documents, demonstrating the complementary role of document stores in our lakehouse architecture.

TABLE 3. MAPPING OF RUBRIC ITEMS TO EVIDENCE

Rubric item	How we satisfy it	Evidence
Presentation skills	10–12 minute in-class presentation with timed agenda and dedicated Q&A slide	Slides, video link (App. C)
Code walkthrough	Explain ETL notebooks and key SQL for marts	Slides, GitHub repo (App. C)
Demo	Live BigQuery demo: preview marts and run KPI query	BigQuery screenshots (App. B)
Version control	Public GitHub repository with notebooks, SQL, diagrams	GitHub URL (App. C)
Significance to real world	Planning questions on airline demand, terminal load, fleet mix	Sec. 10
Lessons learned	Discussion of modeling and pipeline lessons	Sec. 7
Innovation	Lakehouse layers, passengers_per_landing KPI, CouchDB integration	Sec. 10
Teamwork	Shared tasks across four members	Sec. 8, Excel tracker (App. C)
Technical difficulty	Two datasets, star schema, four marts, SCD demos, NoSQL	Sec. 7
Pair programming	Zoom session with shared SQL screen and two active participants	Zoom screenshots (App. C)
Agile / scrum	Three-week Excel task board (Week 1–3) with status updates	Excel screenshots (App. C)
Slides	IEEE-style slide deck	Slides (App. C)
Report	6–11 page IEEE paper with appendices	This document
Used unique tools	Overleaf, Mermaid diagrams, CouchDB NoSQL store	Sec. 2, App. B
New ETL / DW tool	CouchDB + Python ETL round-trip into BigQuery	Sec. 5, App. B
Analytics decision support	Marts and charts used to answer planning questions	Sec. 6

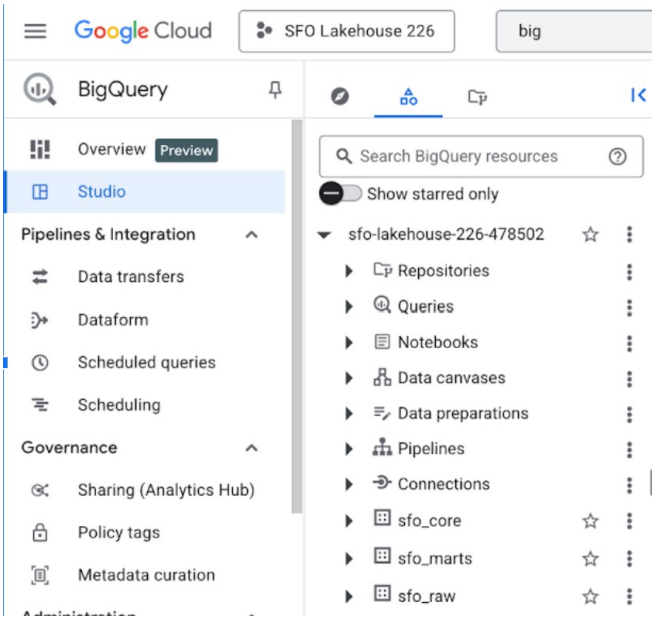


Figure 9. BigQuery project `sfo-lakehouse-226` with raw (`sfo_raw`), core (`sfo_core`), and marts (`sfo_marts`) datasets.

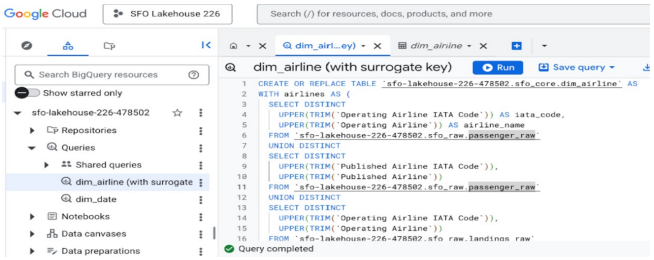


Figure 10. Dimension `dim_airline` showing surrogate key `airline_id` and business key `airline_key`.

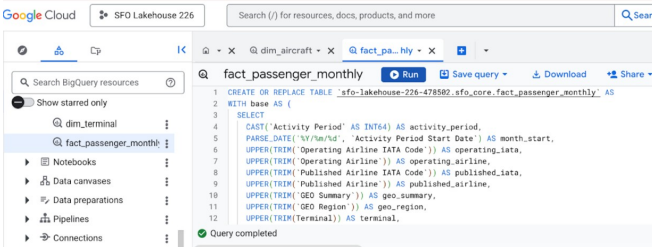


Figure 11. Fact table `fact_passenger_monthly` at monthly grain with foreign keys to conformed dimensions.

Appendix C. Process Evidence

C.1. Three-week task tracker

To manage our agile workflow, we maintained a shared Excel task board organized by weeks, tracking progress from project setup through final deliverables. Figures 15–17 show the evolution of our sprint planning across the three-week project timeline.

C.2. Zoom meetings and pair programming

To manage our agile workflow, we maintained a shared Excel task board organized by weeks, tracking progress from project setup through final deliverables. Figures 18–19–20 show the evolution of our sprint planning across the three-week project timeline.

This appendix lists the external resources referenced in the project.

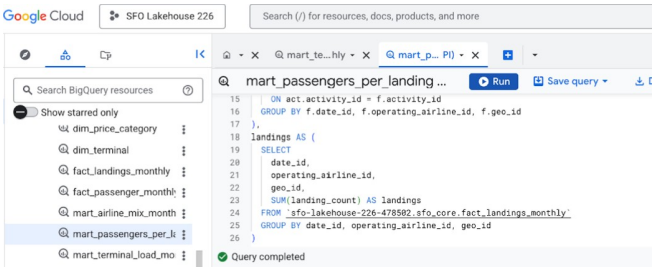


Figure 12. Mart `mart_passengers_per_landing` with the passengers_per_landing KPI.

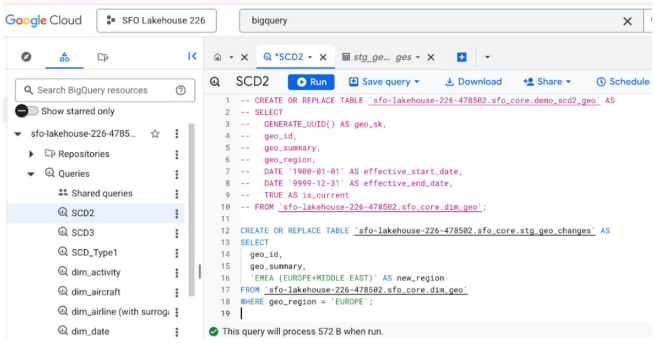


Figure 13. SCD Type 2 demo table for geography, tracking historical changes in `geo_region`.

C.3. Code and artifacts

- **GitHub repository** (code, SQL, diagrams): https://github.com/soroorgh93/sfo-airport-lakehouse
- Python notebooks for ETL and analytics: https://github.com/soroorgh93/sfo-airport-lakehouse/blob/main/ETL-couchdb.py

C.4. Datasets

- Air Traffic Passenger Statistics (DataSF): https://data.sfgov.org/Transportation/Air-Traffic-Passenger-Statistics/rkru-6vcg
- Air Traffic Landings Statistics (DataSF): https://data.sfgov.org/Transportation/Air-Traffic-Landings-Statistics/fpux-q53t

C.5. Documentation and tools

- Google BigQuery documentation: https://cloud.google.com/bigquery/docs
- Apache CouchDB documentation: https://docs.couchdb.org
- Mermaid diagramming tool: https://mermaid.js.org
- Overleaf (LaTeX collaboration): https://www.overleaf.com

C.6. Presentation

- Slides: https://docs.google.com/presentation/d/1hB2MxpPW4vOvrR9i7PhJr-8fHyenPF_A/edit?

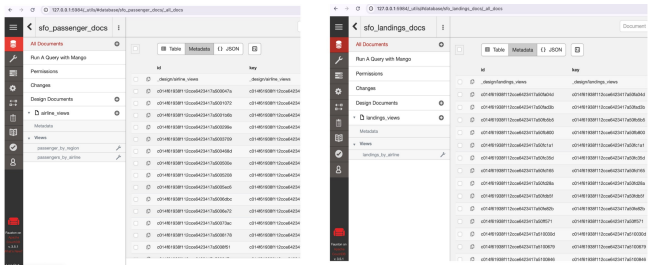


Figure 14. CouchDB `passengers_by_airline` view aggregating passenger counts directly from JSON documents.

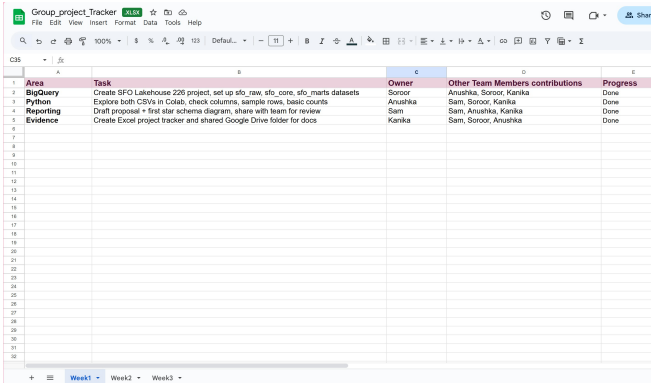
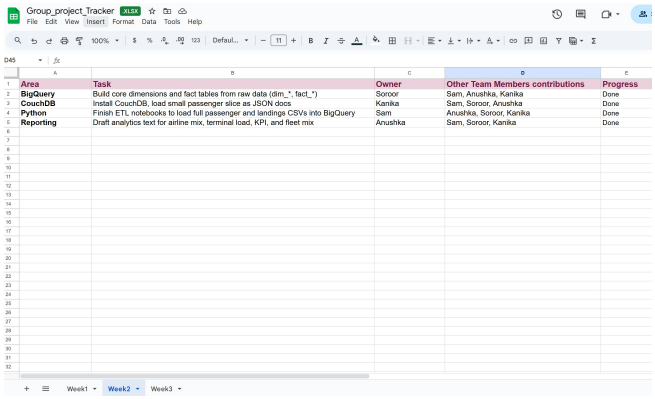


Figure 15. Week 1 Excel task board covering project setup, initial data exploration, and drafting the proposal.

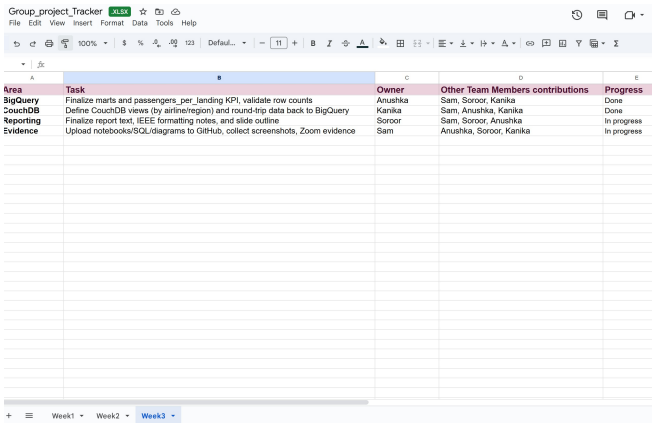
usp=sharing&ouid=106888351366857396505&rtpof=true&sd=true

- Elevator pitch video : https://drive.google.com/file/d/1WRLcrAObG01MwhmCkfTynMj6Ri75xCLC/view?usp=sharing



Area	Task	Owner	Other Team Members contributions	Progress
BigQuery	Build core dimensions and fact tables from raw data (dim_*, fact_*)	Soroor	Sam, Anushka, Kanika	Done
CouchDB	Install CouchDB, load small passenger slice as JSON docs	Kanika	Sam, Soroor, Anushka	Done
Python	Finish ETL notebooks to load full passenger and landings CSVs into BigQuery	Sam	Anushka, Soroor, Kanika	Done
Reporting	Draft analytics text for airline mix, terminal load, KPI, and fleet mix	Anushka	Sam, Soroor, Kanika	Done

Figure 16. Week 2 Excel task board: building dimensions, fact tables, CouchDB setup, and analytics draft.



Area	Task	Owner	Other Team Members contributions	Progress
BigQuery	Finalize marts and passengers_per_landing KPI, validate row counts	Anushka	Sam, Soroor, Kanika	Done
CouchDB	Define CouchDB views (by airline/region) and round-trip data back to BigQuery	Kanika	Sam, Anushka, Kanika	Done
Reporting	Finalize report text, IEEE formatting notes, and slide outline	Soroor	Sam, Soroor, Anushka	In progress
Evidence	Upload notebooks/SQL/diagrams to GitHub, collect screenshots, Zoom evidence	Sam	Anushka, Soroor, Kanika	In progress

Figure 17. Week 3 Excel task board: finalizing marts, report, slides, and evidence collection.

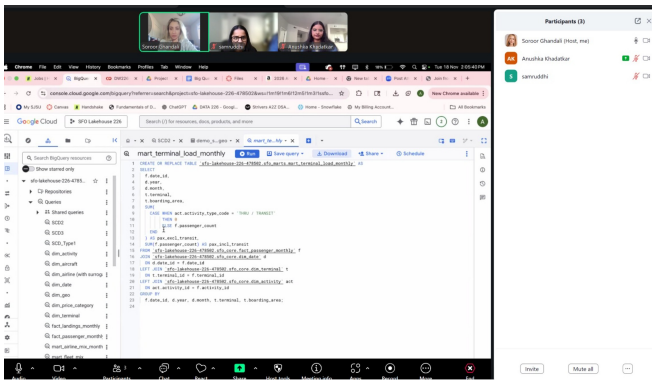


Figure 18. Recurring Zoom meeting for the *Group Project Tracker* on Tuesdays, used as our sprint check-in.

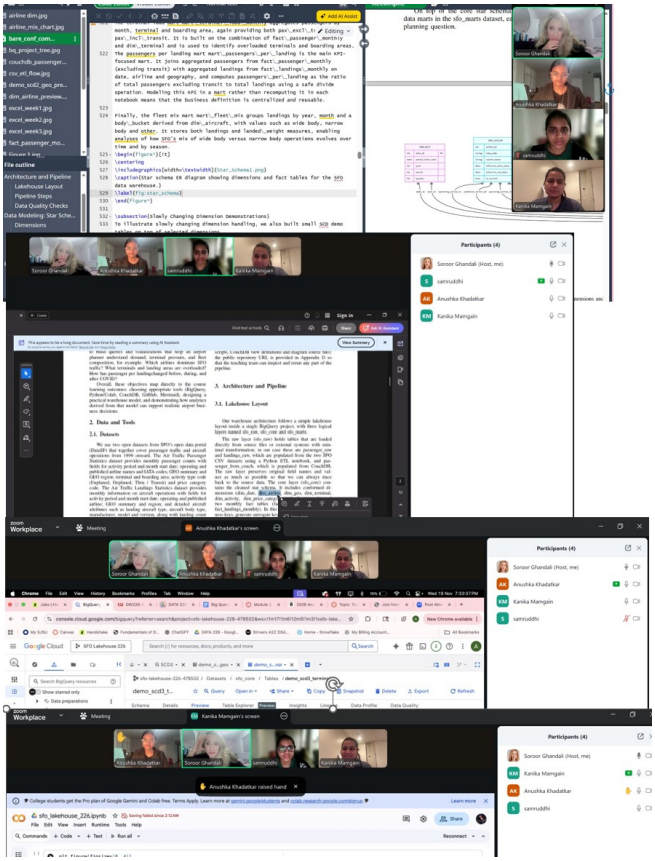


Figure 19. Additional Zoom working session in the final week for joint debugging and report integration.

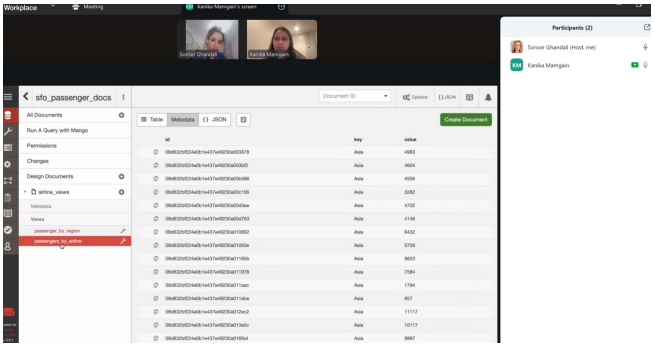


Figure 20. Pair-programming session on CouchDB views, with shared Fauxton screen and two active participants.