**Data Pipeline Project**

Table of Contents

[1. Introduction: 3](#_Toc200990122)

[2. Project Objective: 3](#_Toc200990123)

[3. Analysis Focus: 3](#_Toc200990124)

[4. Dataset Details: 3](#_Toc200990125)

[4.1. Housing Rent Dataset 4](#_Toc200990126)

[4.2. Quality of Life (QOL) Dataset 4](#_Toc200990127)

[4.3. American Community Survey Dataset 4](#_Toc200990128)

[5. Environmental Setup: 4](#_Toc200990129)

[5.1. Database and Schemas Setup 4](#_Toc200990130)

[5.2. Create Team Warehouse 4](#_Toc200990131)

[6. Role-Based Access Management – Initial Set up 5](#_Toc200990132)

[6.1. Grant Access to Team Roles 5](#_Toc200990133)

[6.2. Data Access Permissions 5](#_Toc200990134)

[6.3. Additional Grants for a specific ROLE 5](#_Toc200990135)

[Revoke Setup Role Access 5](#_Toc200990136)

[7. Final Security Hardening – Governance Model 6](#_Toc200990137)

[7.2. Grant Data Access to Team Roles 6](#_Toc200990138)

[7.3. Remove Unnecessary Privileges 6](#_Toc200990139)

[8. RAW Schema - Data Ingestion 6](#_Toc200990140)

[8.1. JSON Data Ingestion 6](#_Toc200990141)

[8.2. CSV Data Ingestion 7](#_Toc200990142)

[9: Data Preparation in the PREP Schema 7](#_Toc200990143)

[9.1. CSV Data Preparation 8](#_Toc200990144)

[9.2. JSON Data Preparation 8](#_Toc200990145)

[9.3. Income Data Transformation 8](#_Toc200990146)

[10: Data Refinement Layer 9](#_Toc200990147)

[10.1. JSON Data Refinement 9](#_Toc200990148)

[10.2. Housing Data Refinement 9](#_Toc200990149)

[10.3. Median Household Income Data 9](#_Toc200990150)

[11: Data Cleaning in the Refinement Layer 9](#_Toc200990151)

[11.1. Standardizing State Codes 9](#_Toc200990152)

[11.2. Removing Outlier Price Values 10](#_Toc200990153)

[12: Data Analysis in the Refinement Layer 10](#_Toc200990154)

[12.1. Rent Price Influencer 10](#_Toc200990155)

[Key Findings 11](#_Toc200990156)

[Visual Insight 11](#_Toc200990157)

[Conclusion 12](#_Toc200990158)

[12.2. Top 5 & Bottom 5 States by Monthly Rent Price 12](#_Toc200990159)

[Key Findings 13](#_Toc200990160)

[Visual Insight 13](#_Toc200990161)

[Conclusion 14](#_Toc200990162)

[12.3. Rent Burden 14](#_Toc200990163)

[Key Findings 14](#_Toc200990164)

[Visual Insight 14](#_Toc200990165)

[Conclusion 15](#_Toc200990166)

[12.4. Average Rent by Property Size & Type 15](#_Toc200990167)

[13. Data Delivery and Final Dashboard: 17](#_Toc200990168)

[13.1. Viewer Role Setup for Final Dashboard Access 18](#_Toc200990169)

[14. DBT (Data Build Tool) 18](#_Toc200990170)

[14.1. Project Structure 19](#_Toc200990171)

[14.2. Naming / Explaining 19](#_Toc200990172)

[14.3. Automated Documentation 20](#_Toc200990173)

[14.4. Directed Acyclic Graph (DAG) 21](#_Toc200990174)

[14.5. DevOps Integration 22](#_Toc200990175)

[14.6. Git Branch Management 22](#_Toc200990176)

[15. Retrospective Report: 23](#_Toc200990177)

[15.1. RBAC (Role-Based Access Control) Implementation 23](#_Toc200990178)

[15.2. Building an End-to-End Data Pipeline in Snowflake and DBT 25](#_Toc200990179)

Team 4: Data Pipeline Project Report

# 1. Introduction:

This project focuses on establishing robust data pipeline architecture for the Team 4 group using the Snowflake data platform and dbt.

# 2. Project Objective:

To design a secure and well-structured environment that supports seamless data ingestion, transformation, and delivery to enable insightful analysis.  
  
We collaborated closely using the **Trello Scrum Board** for agile project management, ensuring smooth teamwork and task tracking.

# 3. Analysis Focus:

The pipeline integrates multiple data sources, processes them in distinct data layers, and ensures that team members have controlled, role-based access to the data. Following best practices in data governance and data engineering, we carefully implemented roles, schemas, and permissions to support the project’s end-to-end data lifecycle.  
  
Through this setup, we enable collaborative and efficient workflows that power downstream use cases, including business intelligence dashboards and analytics.

**DBT Model Building**

In addition to SQL-based transformations, we leveraged **DBT (Data Build Tool)** for structured, version-controlled data modelling. This step ensured modular, reusable, and well-documented transformations, while supporting team collaboration in our Git repository.

4. Dataset Details:  
This pipeline is built with 3 different datasets with 3 various file formats to enable a comprehensive analysis of housing rent prices and their relationship with various socio-economic and quality-of-life factors:

## 4.1. Housing Rent Dataset

**Format:** CSV

**Source:** Kaggle

**Description:** Contains rental housing data such as Price, amenities included and the location(state)

## 4.2. Quality of Life (QOL) Dataset

**Format:** JSON

**Source:** Kaggle

**Description:** Captures quality-of-life indicators such as affordability, economy, safety, and education/health metrics.

## 4.3. American Community Survey Dataset

**Format:** Snowflake Marketplace dataset

**Source:** American Community Survey (ACS) 2016 data from the Marketplace

**Description:** Provides median household income and aggregate family income by state.

# **5**. **Environmental Setup**:

**Set Role for Setup**  
We began by setting the role to TRAINING\_ROLE to ensure we had the required privileges to create and manage the necessary resources.

**SQL Command:**  
USE ROLE TRAINING\_ROLE.

## 5.1. Database and Schemas Setup

We created a shared database for the team, with separate schemas for each data layer: RAW, PREP, REFINED, and DELIVERY. This ensures a clean and modular data pipeline structure.  
  
**SQL Commands:**  
CREATE DATABASE IF NOT EXISTS TEAM\_4\_DB;  
CREATE SCHEMA IF NOT EXISTS TEAM\_4\_DB.RAW;  
CREATE SCHEMA IF NOT EXISTS TEAM\_4\_DB.PREP;  
CREATE SCHEMA IF NOT EXISTS TEAM\_4\_DB.REFINED;  
CREATE SCHEMA IF NOT EXISTS TEAM\_4\_DB.DELIVERY;

## 5.2. Create Team Warehouse

We created a dedicated warehouse to perform transformations and support team workloads.

**SQL Command:**  
CREATE WAREHOUSE IF NOT EXISTS TEAM\_4\_WAREHOUSE;

# **6. Role-Based Access Management – Initial Set up**

Initially, we provided broad access to **team roles** to foster collaboration and hands-on learning.  
This setup allowed team members to:

* Create, read, and work with data across all layers.
* Build a collaborative, flexible learning environment.

## 6.1. Grant Access to Team Roles

We ensured proper access for the team roles (CHIPMUNK\_ROLE, KOALA\_ROLE, LEMMING\_ROLE, LEMUR\_ROLE) to the warehouse, database, and schemas, following the principle of least privilege.  
  
**Few of the SQL Commands:**  
GRANT USAGE ON WAREHOUSE TEAM\_4\_WAREHOUSE TO ROLE CHIPMUNK\_ROLE;  
GRANT USAGE ON DATABASE TEAM\_4\_DB TO ROLE KOALA\_ROLE;  
GRANT USAGE ON SCHEMA TEAM\_4\_DB.RAW TO ROLE LEMMING\_ROLE;  
and so on for all roles and schemas.

## 6.2. Data Access Permissions

We applied SELECT-only grants on RAW, PREP, and REFINED schemas for relevant team roles to ensure data integrity and controlled access. Additionally, future tables and views were also covered.

**Few of the SQL Commands:**  
GRANT SELECT ON ALL TABLES IN SCHEMA TEAM\_4\_DB.RAW TO ROLE LEMMING\_ROLE;  
GRANT SELECT ON FUTURE TABLES IN SCHEMA TEAM\_4\_DB.PREP TO ROLE CHIPMUNK\_ROLE;  
GRANT SELECT ON FUTURE VIEWS IN SCHEMA TEAM\_4\_DB.REFINED TO ROLE LEMUR\_ROLE;

## 6.3. Additional Grants for a specific ROLE

The KOALA\_ROLE was granted broader permission to create tables, views, and schemas within the RAW, PREP, and REFINED schemas to support more advanced data engineering tasks.

## Revoke Setup Role Access

Finally, we revoked all privileges from TRAINING\_ROLE to ensure security and compliance.

**SQL Command:**  
REVOKE ALL PRIVILEGES ON DATABASE TEAM\_4\_DB FROM ROLE TRAINING\_ROLE

This setup ensures a collaborative environment for the Team 4 data pipeline project.

# **7. Final Security Hardening – Governance Model**

For the final project phase, all privileges were updated to match real-world security and governance best practices:  
  
7.1. Create and Configure Team Role  
Created a **dedicated team role** (**TEAM\_4\_USER\_ROLE**) that owns all project data. This ensures a clear separation of data ownership and access control.  
  
The **ownership** of all database objects was **transferred** to **TEAM\_4\_USER\_ROLE**, ensuring clear control and governance:

## 7.2. Grant Data Access to Team Roles

**USAGE** and **SELECT** access to project roles for read-only access and collaboration:

**Few SQL Commands:**

USE ROLE TEAM\_4\_USER\_ROLE;

GRANT USAGE ON DATABASE TEAM\_4\_DB TO ROLE KOALA\_ROLE;

GRANT USAGE ON SCHEMA TEAM\_4\_DB.RAW TO ROLE KOALA\_ROLE;

GRANT SELECT ON ALL TABLES IN SCHEMA TEAM\_4\_DB.RAW TO ROLE KOALA\_ROLE;

## 7.3. Remove Unnecessary Privileges

To follow the **least privilege** principle, a **revoked** all future object privileges that were previously granted:  
  
REVOKE ALL PRIVILEGES ON FUTURE TABLES IN DATABASE TEAM\_4\_DB FROM ROLE KOALA\_ROLE;

# 8. RAW Schema - Data Ingestion

After setting up the roles, schemas, and permissions, we proceeded to load our raw data into Snowflake.

## 8.1. JSON Data Ingestion

We switched to the RAW schema and created a table to store JSON data.  
  
-- Switch to RAW schema  
USE SCHEMA TEAM\_4\_DB.RAW;

SQL Commands:  
-- Create table to store JSON data  
CREATE TABLE IF NOT EXISTS QOL\_JSON\_RAW (  
 json\_data VARIANT  
);

## 8.2. CSV Data Ingestion

We prepared the housing CSV data by creating a structured table and ingested income data from the Snowflake Marketplace.

SQL Commands:  
-- Create table for housing CSV data and loaded data using **SNOWSQL**:

COPY INTO TEAM\_4\_DB.RAW.housing\_CSV\_RAW

(id, price, type, sqfeet, beds, baths, cats\_allowed, dogs\_allowed, smoking\_allowed, wheelchair\_

access, electric\_vehicle\_charge, comes\_furnished, parking\_options, state)

FROM @~/housing\_cleaned.csv.gz

FILE\_FORMAT = (TYPE = 'CSV' FIELD\_OPTIONALLY\_ENCLOSED\_BY='"' SKIP\_HEADER=1)

ON\_ERROR = 'CONTINUE';  
  
8.3. Snowflake Marketplace Data Ingestion

CREATE OR REPLACE TABLE TEAM\_4\_DB.RAW.MARKETPLACE\_INCOME\_RAW AS  
SELECT  
 \*  
FROM AMERICAN\_COMMUNITY\_SURVEY\_2016.PUBLIC."acs-2016-5-e-income\_USA\_ALL\_STATES";

This completes the foundational setup and data ingestion for our project. Moving forward, we will refine, transform, and analyze the data in the subsequent layers to derive actionable insights for our stakeholders.

9: Data Preparation in the PREP Schema

After loading the raw data into the RAW schema, we moved to the PREP schema to do minimal changesand structure the data for downstream use cases.

## 9.1. CSV Data Preparation

We also performed a simple data preview from the housing CSV data:

SELECT

id,

price,

type,

sqfeet,

beds,

baths,

cats\_allowed,

dogs\_allowed,

smoking\_allowed,

wheelchair\_access,

electric\_vehicle\_charge,

comes\_furnished,

parking\_options,

state

FROM TEAM\_4\_DB.RAW.housing\_CSV\_RAW;

## 9.2. JSON Data Preparation

The relevant fields are extracted from the JSON variant column and structured them as columns in the new QOL\_JSON\_PREP table:  
  
CREATE OR REPLACE TABLE TEAM\_4\_DB.PREP.QOL\_JSON\_PREP AS

SELECT

json\_data:state::STRING AS state,

json\_data:QualityOfLifeTotalScore::NUMBER AS Qol\_TotalScore,

json\_data:QualityOfLifeQualityOfLife::NUMBER AS Qol\_Life,

json\_data:QualityOfLifeAffordability::NUMBER AS Qol\_Affordability,

json\_data:QualityOfLifeEconomy::NUMBER AS Qol\_Economy,

json\_data:QualityOfLifeEducationAndHealth::NUMBER AS Qol\_Education\_and\_Health,

json\_data:QualityOfLifeSafety::NUMBER AS Qol\_Safety

FROM TEAM\_4\_DB.RAW.QOL\_JSON\_RAW;

## 9.3. Income Data Transformation

We created a structured table for median household income data from the Snowflake Marketplace dataset:

CREATE OR REPLACE TABLE TEAM\_4\_DB.PREP.median\_household\_income\_PREP AS

SELECT

STATE,

AREANAME,

B19013\_001 AS MEDIAN\_HOUSEHOLD\_INCOME,

B19127\_001 AS AGGREGATE\_FAMILY\_INCOME

FROM TEAM\_4\_DB.RAW.MARKETPLACE\_INCOME\_RAW;

# 10: Data Refinement Layer

With the data prepared in the PREP schema, we moved to the REFINED schema, which serves as the **foundation for final analysis and reporting**.  
The REFINED layer’s main purpose is to:

* Clean data
* Join with other datasets
* Aggregate data

## ****10.1. JSON Data Refinement****

Since the JSON data in QOL\_JSON\_PREP was already structured, we simply transferred it to the refined schema:

CREATE OR REPLACE TABLE TEAM\_4\_DB.REFINED.QOL\_JSON\_REFINED AS

SELECT \*

FROM TEAM\_4\_DB.PREP.QOL\_JSON\_PREP;

## 10.2. Housing Data Refinement

CREATE OR REPLACE TABLE TEAM\_4\_DB.REFINED.housing\_CSV\_REFINED AS

SELECT \*

FROM TEAM\_4\_DB.PREP.housing\_CSV\_PREP;

## 10.3. Median Household Income Data

CREATE OR REPLACE TABLE TEAM\_4\_DB.REFINED.MEDIAN\_HOUSEHOLD\_INCOME\_REFINED AS

SELECT

STATE,

AREANAME,

MEDIAN\_HOUSEHOLD\_INCOME,

AGGREGATE\_FAMILY\_INCOME

FROM TEAM\_4\_DB.PREP.median\_household\_income\_PREP;

# 11: Data Cleaning in the Refinement Layer

The REFINED layer’s role is to provide clean and reliable data for downstream analysis. We performed the following data cleaning tasks:

## 11.1. Standardizing State Codes

We ensured that all state codes are in uppercase for consistency:

UPDATE TEAM\_4\_DB.REFINED.HOUSING\_CSV\_REFINED

SET STATE = UPPER(STATE);

## 11.2. Removing Outlier Price Values

To ensure realistic rent data, we removed records with potentially invalid price values:

* **Too high (length ≥ 6)**:

DELETE FROM TEAM\_4\_DB.REFINED.HOUSING\_CSV\_REFINED

WHERE LENGTH(CAST(PRICE AS STRING)) >= 6;

* **Too low (length ≤ 1)**:

DELETE FROM TEAM\_4\_DB.REFINED.HOUSING\_CSV\_REFINED

WHERE LENGTH(CAST(PRICE AS STRING)) <= 1;

This data cleaning ensures that only realistic, consistent, and reliable housing price data is available for subsequent analysis.

# 12: Data Analysis in the Refinement Layer

In this session, we dive into the intricate relationships between rent prices, income, and quality-of-life indicators across U.S. states. Our analysis explores the following:

## 12.1. Rent Price Influencer

We followed a structured approach to understand rent prices and their relationship with quality of life (QOL) factors and income. This involved conducting correlation analysis using the CORR() function in Snowflake SQL.

**SQL Command:**

WITH rent\_by\_state AS (

SELECT

C.STATE,

MEDIAN(C.PRICE) AS MEDIAN\_RENT

FROM TEAM\_4\_DB.REFINED.HOUSING\_CSV\_REFINED C

GROUP BY C.STATE

),

income\_rent\_qol AS (

SELECT

M.STATE,

M.MEDIAN\_HOUSEHOLD\_INCOME,

R.MEDIAN\_RENT,

-Q.Qol\_Affordability AS Qol\_Affordability\_Reversed,

-Q.Qol\_Economy AS Qol\_Economy\_Reversed,

-Q.Qol\_Education\_And\_Health AS Qol\_Education\_And\_Health\_Reversed,

-Q.Qol\_Safety AS Qol\_Safety\_Reversed

FROM rent\_by\_state R

INNER JOIN TEAM\_4\_DB.REFINED.MEDIAN\_HOUSEHOLD\_INCOME\_REFINED M

ON R.STATE = M.STATE

INNER JOIN TEAM\_4\_DB.REFINED.QOL\_JSON\_REFINED Q

ON M.AREANAME = Q.STATE

),

correlations AS (

SELECT

CORR(MEDIAN\_HOUSEHOLD\_INCOME, MEDIAN\_RENT) AS correlation\_income\_rent,

CORR(Qol\_Affordability\_Reversed, MEDIAN\_RENT) AS correlation\_affordability\_rent,

CORR(Qol\_Economy\_Reversed, MEDIAN\_RENT) AS correlation\_economy\_rent,

CORR(Qol\_Education\_And\_Health\_Reversed, MEDIAN\_RENT) AS correlation\_education\_health\_rent,

CORR(Qol\_Safety\_Reversed, MEDIAN\_RENT) AS correlation\_safety\_rent

FROM income\_rent\_qol

)

SELECT 'Income-Rent' AS metric, ROUND(correlation\_income\_rent, 2) AS correlation FROM correlations

UNION ALL

SELECT 'Affordability-Rent', ROUND(correlation\_affordability\_rent, 2) FROM correlations

UNION ALL

SELECT 'Economy-Rent', ROUND(correlation\_economy\_rent, 2) FROM correlations

UNION ALL

SELECT 'Education-Health-Rent', ROUND(correlation\_education\_health\_rent, 2) FROM correlations

UNION ALL

SELECT 'Safety-Rent', ROUND(correlation\_safety\_rent, 2) FROM correlations;

## Key Findings

The results showed a fascinating interplay between rent prices and various QOL factors:

* **Education\_Health-Rent**: 0.73 (Strong positive correlation)
* **Income-Rent**: 0.68 (Strong positive correlation)
* **Safety-Rent**: 0.34 (Moderate positive correlation)
* **Economy-Rent**: 0.24 (Weak positive correlation)
* **Affordability-Rent**: -0.69 (Strong negative correlation)

## Visual Insight

Bar chart to highlight these relationships visually. States with higher **income** and **education and health scores** tended to have higher rents, suggesting that better economic and quality of life conditions can support higher housing costs. Conversely, **affordability** was strongly negatively correlated with rent, indicating that as affordability sharply declines, rent increases. **Safety** and **economy** metrics showed weaker positive correlations, suggesting more nuanced or complex local dynamics affecting rent levels.

A blue rectangular object with text

AI-generated content may be incorrect.

**Figure 1:** Rent Price Influencers - Correlation of QOL Metrics with Median Rent

## Conclusion

This analysis underscores that while income and education/health drive rent higher, affordability remains inversely linked to rent, reflecting its direct impact on household budgets. Safety and economic metrics also show moderate effects, highlighting the complex interplay of social and economic factors in rent pricing.

## 12.2. Top 5 & Bottom 5 States by Monthly Rent Price

In this analysis, we examine the states with the highest and lowest monthly rent prices. Understanding these outliers can provide insight into the broader housing market and its relationship with factors such as geography, demand, and local economic conditions.

**SQL Command:**

(SELECT i.areaname, ROUND(MEDIAN(price)) AS median\_rent\_price, 'Top 5' AS Rent\_Group

FROM refined\_housing h

INNER JOIN refined\_income i  ON h.state = i.state

GROUP BY i.areaname

ORDER BY median\_rent\_price DESC

LIMIT 5)

UNION ALL

(SELECT i.areaname, ROUND(MEDIAN(price)) AS median\_rent\_price, 'Bottom 5' AS Rent\_Group

FROM refined\_housing h

INNER JOIN refined\_income i ON h.state = i.state

GROUP BY i.areaname

ORDER BY median\_rent\_price ASC

LIMIT 5)

ORDER BY median\_rent\_price DESC

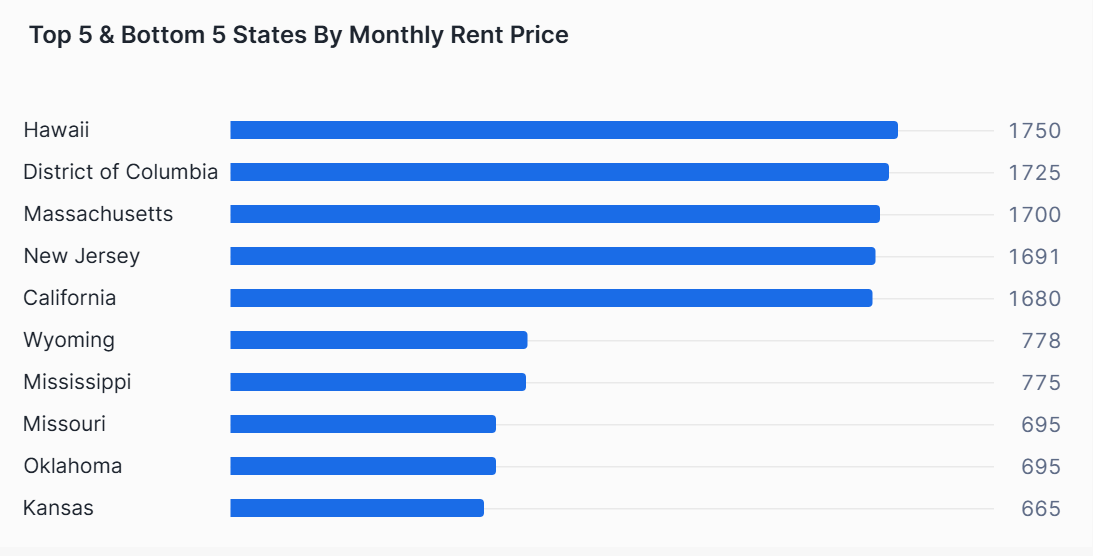
## Key Findings

The results clearly illustrate a stark divide between the states with the highest and lowest rent prices:

* Top 5 States: The states in the Top 5 rent group, including Hawaii, District of Columbia, and Massachusetts, show rent prices that are significantly higher than the national average. This suggests these regions may have higher living standards, stronger economic conditions, and greater demand for housing, etc.
* Bottom 5 States: Conversely, states in the Bottom 5 rent group, such as Wyoming, Mississippi, and Kansas, have lower median rent prices. These areas likely face different economic dynamics, including lower demand for rental properties, less economic growth, or smaller populations, etc..

## Visual Insight

A bar chart would visually depict the stark contrast between the Top 5 and Bottom 5 states.



**Figure 2:** Monthly Rent Analysis

## Conclusion

The analysis of the Top 5 and Bottom 5 states by monthly rent price reveals clear geographical and economic patterns. These findings reinforce the complex interplay between economic factors, demand, and geography in determining rent prices across the U.S.

## 12.3. Rent Burden

In this analysis, we examine the **Rent Burden across different states**. Rent Burden is calculated as the ratio of Median Rent Price to Median Household Income. A higher Rent Burden indicates that a larger portion of household income is going towards paying for rent, which can be an indicator of affordability issues within a given region. Understanding Rent Burden helps to assess the economic pressure on households and the affordability of housing across the United States.

**SQL Command:**

SELECT i.areaname, ROUND(MEDIAN(h.price)\* 12) AS median\_rent\_price, i.median\_household\_income,  ROUND((MEDIAN(h.price) \* 12) / i.median\_household\_income,3) AS rent\_burden

FROM refined\_housing h

INNER JOIN refined\_income i  ON h.state = i.state

GROUP BY i.areaname, i.median\_household\_income

ORDER BY rent\_burden DESC

## Key Findings

* **Highest** Rent Burden States:

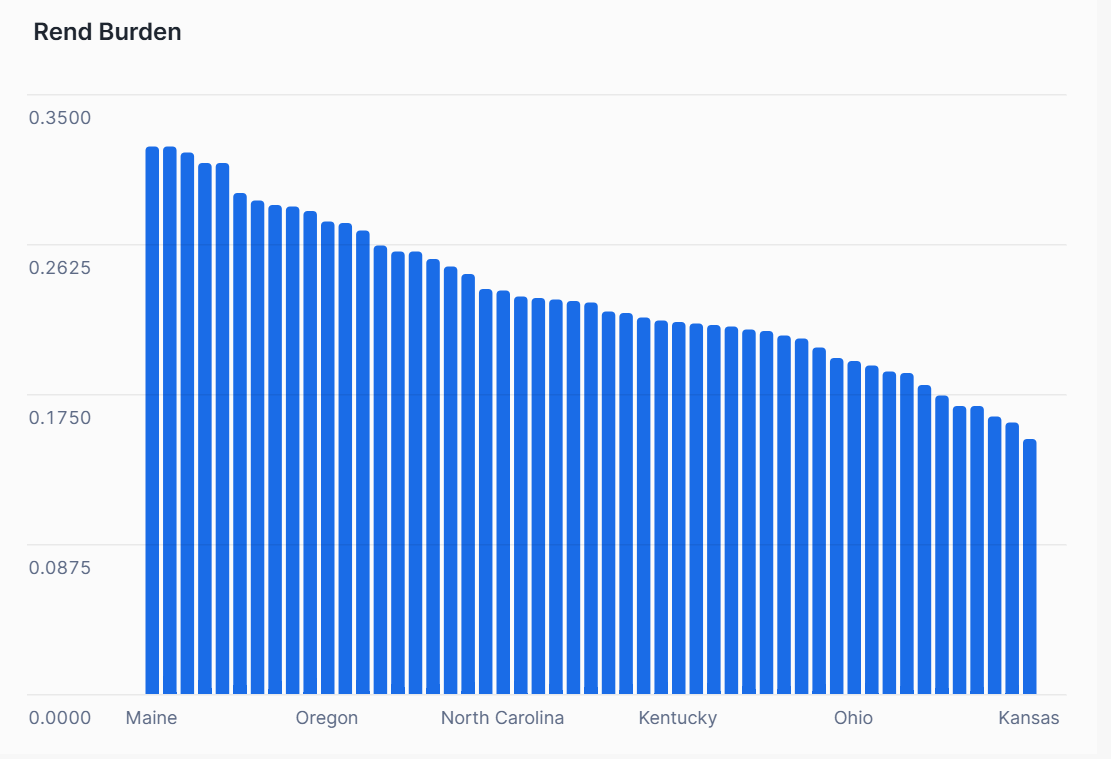
The states with the highest Rent Burden are Maine (0.319), Rhode Island (0.319), and California (0.316). These states have the highest rent-to-income ratios, suggesting that residents in these states are spending a larger portion of their income on rent. This reflects the high cost of living and rent in these states, particularly in areas with strong economic activity but high housing demand.

* **Lowest** Rent Burden States:

On the other end, Kansas (0.149), Wyoming (0.158), and Illinois (0.162) have the lowest Rent Burden. This suggests that housing is more affordable in these states relative to the median income, and residents are spending a smaller portion of their income on rent.

## Visual Insight

A bar chart could effectively illustrate the Rent Burden across various states, highlighting those with the highest and lowest values.



**Figure 3:** Rent Burden

## Conclusion

This analysis underscores the varying Rent Burden across states, reflecting the affordability of housing in different regions. Understanding these disparities is crucial for policymakers and residents alike, as it highlights the direct impact of housing affordability on household budgets.

## 12.4. Average Rent by Property Size & Type

To understand how property **size and type** influence rental pricing, we grouped all listings by **square footage** and **property type** after excluding extreme or invalid values. This approach ensured more accurate and meaningful comparisons across different housing categories.

**SQL Command:**

WITH cleaned\_data AS (

SELECT

type,

housing\_state,

CASE

WHEN sqfeet < 500 THEN 'Small (<500 sqft)'

WHEN sqfeet BETWEEN 500 AND 1000 THEN 'Medium (500–1000 sqft)'

WHEN sqfeet > 1000 THEN 'Large (>1000 sqft)'

END AS size\_group,

price,

sqfeet

FROM TEAM\_4\_DB.DELIVERY.JOINED\_VIEW

WHERE price BETWEEN 100 AND 25000

AND sqfeet BETWEEN 100 AND 8000

AND price IS NOT NULL

AND sqfeet IS NOT NULL

),

avg\_rent\_by\_size\_type AS (

SELECT

type,

size\_group,

ROUND(AVG(price), 2) AS avg\_price,

ROUND(AVG(sqfeet), 0) AS avg\_sqfeet,

COUNT(\*) AS listing\_count

FROM cleaned\_data

GROUP BY type, size\_group

)

SELECT \*

FROM avg\_rent\_by\_size\_type

ORDER BY size\_group, type;  
  
**Key Findings**

The visualization demonstrates a **clear upward trend in rent with increasing property size** for all three property types. Key insights include:

* **Condos** consistently command the highest rent across all size groups, particularly for large units (>$1900), suggesting a premium valuation likely tied to location or amenities.
* **Apartments** show a gradual price increase with size, from approximately **$850 (small)** to **$1500 (large)**.
* **Duplexes** follow a similar trend but sit in the mid-range across all size brackets, offering a balance of space and cost.

Interestingly, **small condos (<500 sqft)** have a higher average rent (~$1300) than **medium apartments**, highlighting the premium placed on compact, likely urban-located units.

**Visual Insight**

Figure 4 highlights the relationship between unit size and average rent across property types. While larger spaces generally cost more, the rate of price increase varies by type. Condos, often associated with premium amenities and central locations, experience the steepest rent curve. Apartments and duplexes show more gradual transitions, with duplexes possibly offering better value per square foot in larger size groups.  
A graph of different colored squares

AI-generated content may be incorrect.  
**Figure 4:** Average Rent Analysis

**Conclusion**

This analysis confirms that both size and type are key drivers of rent, but their impact is nuanced. Condos lead in pricing regardless of size, apartments offer affordability with consistency, and duplexes present a mid-range option attractive to larger households. Understanding this dynamic can guide housing decisions for both renters and developers targeting segmented rental markets.

# 13. Data Delivery and Final Dashboard:

**Viewer role** (TEAM\_4\_VIEWER\_ROLE) ensures **final data delivery** is done in a **secure, read-only** manner.

## 13.1. Viewer Role Setup for Final Dashboard Access

To enable read-only access to final data in the DELIVERY schema, a new role TEAM\_4\_VIEWER\_ROLE was created:  
  
USE ROLE TRAINING\_ROLE;

CREATE ROLE TEAM\_4\_VIEWER\_ROLE;

GRANT ROLE TEAM\_4\_VIEWER\_ROLE TO USER KOALA;   
  
Grant Minimal Read Access  
USE ROLE TEAM\_4\_USER\_ROLE;

GRANT USAGE ON DATABASE TEAM\_4\_DB TO ROLE TEAM\_4\_VIEWER\_ROLE;

GRANT USAGE ON SCHEMA TEAM\_4\_DB.DELIVERY TO ROLE TEAM\_4\_VIEWER\_ROLE;

GRANT USAGE ON WAREHOUSE TEAM\_4\_WAREHOUSE TO ROLE TEAM\_4\_VIEWER\_ROLE;

GRANT SELECT ON ALL VIEWS IN SCHEMA TEAM\_4\_DB.DELIVERY TO ROLE TEAM\_4\_VIEWER\_ROLE;  
  
13.2. Final Dashboard  
This role grants data consumers access to **only the finalized data models and dashboards**, ensuring the integrity and security of the underlying data by maintaining strict control over data modifications.  
This role allows stakeholders, such as analysts and decision-makers—to access the **Housing\_Rent\_Analysis dashboard**  and other final models without risking any modification of the underlying data.

A screenshot of a graph

AI-generated content may be incorrect.  
**Figure 5:** Housing Rent Analysis Dashboard

# 14. DBT (Data Build Tool)

In this project, DBT was integrated with snowflake to transform the data pipeline into a modular, collaborative, and version-controlled project. DBT enables us to use SQL-based models to transform raw data into analytics-ready datasets in a structured and reproducible way. It also provides a DAG (Directed Acyclic Graph) to visualize dependencies and ensure a clear flow from raw data to final insights.

## 14.1. Project Structure

The project is structured into **RAW, PREP, and REFINED** layers:

**RAW**: Ingested raw data from CSV, JSON, and Marketplace sources.

**PREP**: Lightly transformed data, ensuring schema consistency.

**REFINED**: Final refined datasets, fully cleaned and structured for analysis.

**Analysis Layer**: Final models to perform the business analysis

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 5:** DBT Project Structure and Models

This figure shows the modular structure of our DBT project, including the RAW, PREP, REFINED layers and the final analysis model. The YAML schema files define tests and documentation for data quality and governance.

## 14.2. Naming / Explaining

**In DBT, we utilized the {{ ref() }} and {{ source() }} macros to manage data lineage and dependencies:**

* **{{ ref('model\_name') }}**: Links **refined or prep models** to their upstream data transformations (e.g., {{ ref('refined\_qol\_data') }}).
* **{{ source('schema', 'table') }}**: References **raw data tables** directly (e.g., {{ source('raw', 'HOUSING\_CSV\_RAW') }}).

This modular approach ensures that **raw data** feeds into **prep models** via {{ source() }}, and **prep models** feed into **refined models** via {{ ref() }}.  
A screenshot of a computer

AI-generated content may be incorrect.A list of words on a white background

AI-generated content may be incorrect.

**Figure 6:** RAW **Figure 7:** PREP

## 14.3. Automated Documentation

DBT automatically generates comprehensive documentation for our models, including descriptions of datasets, column-level details, and dependencies. By running dbt docs generate, we created an interactive website that visualizes the data pipeline’s structure (via a DAG) and clearly explains each transformation step, making it easy for stakeholders to explore and understand our data models.

A screenshot of a web page

AI-generated content may be incorrect.

**Figure 8:** DB used in DBT

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 9**: DBT Documentation Project Overview

## 14.4. Directed Acyclic Graph (DAG)

A **Directed Acyclic Graph (DAG)** to visualize dependencies and data lineage. This helps to understand how raw data flowed into final analysis outputs.

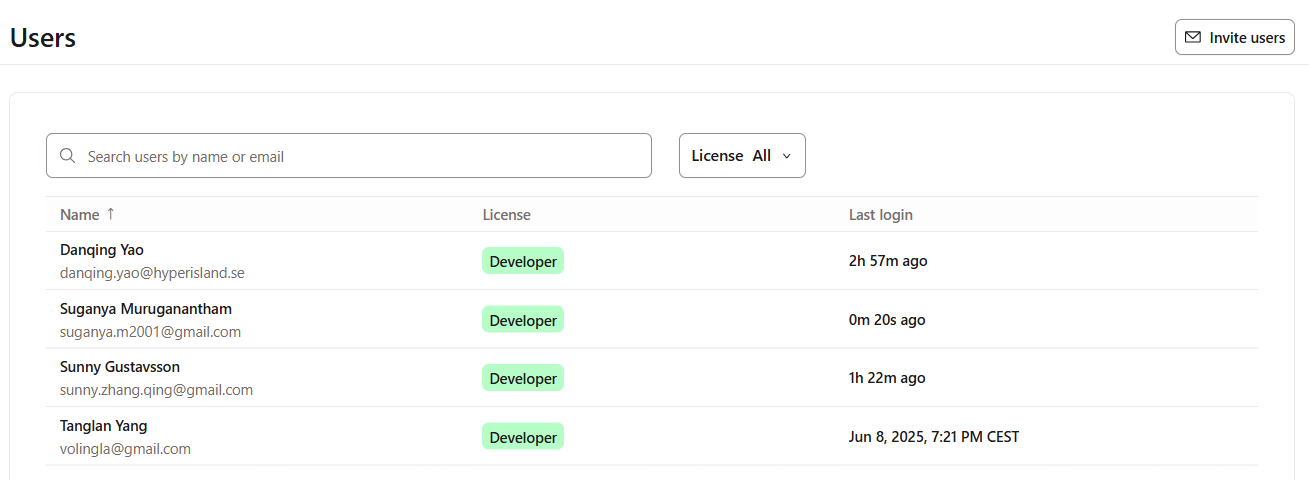
A screenshot of a computer

AI-generated content may be incorrect.

**Figure 10:** End-to-End Data Transformation Lineage

## 14.5. DevOps Integration

By integrating DBT with Git for version control, it helps to practice the devops integration, ensuring that changes in transformation logic could be tested, reviewed, and safely deployed. This **collaborative, transparent approach** improved both our workflow and the reliability of our data pipeline.



**Figure 11:** User Access and Collaboration in the Project

## 14.6. Git Branch Management

Effective **branch management** is critical for collaborating on data models and analytics projects, ensuring that work can be done in parallel and merged safely.  
A screenshot of a calendar

AI-generated content may be incorrect.

**Figure 12:** Git Branch Management **Key points:**  
1. The main **branch** is the production-ready branch containing the finalized, tested models.

2. The suganyam2002-housing **branch** was created for focused development of housing analytics transformations.

The interface clearly shows the **current working branch** (highlighted as “current branch”), enabling developers to quickly confirm their environment and reduce errors.  
**Usage and Workflow:**

* Developers switch to feature branches (like suganyam2001-housing) to make changes or add new models.
* Once updates are complete and tested, changes are merged back to main to ensure a stable, unified project.
* This workflow ensures **clean, traceable development**, and **safe collaboration** on the project.

A screenshot of a chat

AI-generated content may be incorrect.  
**Figure 13:** Privilege in Project DB

**DBT Schema Access Grants some SQL Commands for Snowflake:**GRANT USAGE ON DATABASE KOALA\_ANALYTICS TO ROLE LEMUR\_ROLE;  
GRANT USAGE ON WAREHOUSE KOALA\_TRANSFORMING TO ROLE CHIPMUNK\_ROLE;

GRANT USAGE ON SCHEMA KOALA\_ANALYTICS.dbt\_smuruganantham TO ROLE CHIPMUNK\_ROLE;

GRANT SELECT ON ALL VIEWS IN SCHEMA KOALA\_ANALYTICS.dbt\_smuruganantham TO ROLE LEMMING\_ROLE;  
  
**Summary:**  
DBT played a crucial role in our project by enabling structured, version-controlled data transformations and collaborative development. Its ability to visualize data dependencies through DAG and integrate seamlessly with Git has made our pipeline robust, transparent, and easy to maintain. DBT’s automated documentation and testing further ensured the quality and governance of our data models — aligning perfectly with modern Development Operations practices.

# 15. Retrospective Report:

## 15.1. RBAC (Role-Based Access Control) Implementation

We implemented Role-Based Access Control (RBAC) to manage data access in our database within Snowflake. Our goal was to ensure that different users and teams had the right level of access, balancing collaboration and security.

**What worked well**

1. We created dedicated roles for the Team
2. We used **future grants** (SELECT - FUTURE TABLE, USAGE - FUTURE SCHEMA, etc.), ensuring that **new tables or views** created in the future inherit these permissions automatically – saving time and reducing manual errors.
3. We aligned access based on **principle of least privilege** – giving only the required level of access to each role, minimizing security risks.

**Challenges**

1. Initially, the Training Role had very broad privileges – including the ability to grant themselves higher access.
2. Team members could potentially **escalate their own privileges** by using the **future grants** if not properly managed.
3. There was some confusion about how **future grants** worked and which roles needed privileges on the database and schemas.

**What did we learn**

1. RBAC in Snowflake is powerful but can be tricky to get right without careful planning.
2. It’s important to **audit existing privileges** and **remove excessive permissions** to avoid unnecessary risks.
3. Using **future grants** strategically can **save time** and **reduce maintenance work**, but you have to carefully control who gets those grants to avoid accidental privilege escalation.
4. Aligning privileges to **team-specific roles** fosters better collaboration **without compromising security**.

**Lessons for next time**

1. Start with least privilege and add as needed – instead of starting with broad access and trying to reduce it later.
2. Document roles and permissions carefully so that everyone understands what access they have and why.
3. **Regularly audit roles and grants** to ensure they align with actual needs.
4. Use **team-specific roles** to maintain control and avoid privilege sprawl.

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 14:** Broader Privileges A screenshot of a computer

AI-generated content may be incorrect.

**Figure 15:** Minimal Privileges

## 15.2. Building an End-to-End Data Pipeline in Snowflake and DBT

In this project, we aimed to build a complete data pipeline from raw data ingestion to dashboard-ready views. We approached this using two different tools:

* Directly in Snowflake, using manual SQL scripts for each layer (RAW, PREP, REFINED, DELIVERY).
* In DBT integrated with Snowflake, leveraging its automation, version control, and data lineage tracking.

**What Worked Well**

1. We learned and implemented the entire pipeline – from loading raw data to preparing data for dashboards, gaining a solid understanding of real-world data flows.
2. Using two separate databases – one for raw data (TEAM\_4\_DB) and one for transformed data (KOALA\_ANALYTICS) – improved data governance and followed best practices.

**Challenges**

1. Using two databases and duplicating the same transformations in both (manually in Snowflake and then again in DBT) felt redundant and sometimes inefficient.
2. It took time and some trial-and-error to understand how DBT connects to Snowflake and fully uses its power.
3. Managing the same transformations in two places was initially overwhelming, especially across multiple layers (RAW, PREP, REFINED, DELIVERY).

**Key Learnings**We learned that combining DBT and Snowflake is a modern data engineering best practice. DBT’s ability to automate transformations, manage data lineage, and provide version control complements Snowflake’s data storage and processing power.

**Lessons for Next Time**

Where possible, avoid duplicating transformation logic in both Snowflake and DBT. Instead, let DBT handle transformations beyond the raw data layer, using its built-in orchestration and documentation features.