Index

HubSpot Technical Assessment

Overall Problem Statement

Situation- A rental property management company has new source data: Listings, Calendar, Reviews, Amenities Changelog. Analysts want metrics to analyse the Revenue, Occupancy, and Amenity/review trends from the new data sources.

Task - Design an analytical model and queries to support key business questions.

Action:
The goal of this project is to:
1. Model and transform raw source data into clean, analysis-ready datasets.
2. Enable analysts to answer key business questions related to revenue, occupancy, and amenities.
3. Demonstrate solid data modeling, SQL design, and dbt-style modular development using Snowflake as the data platform.

Tools used: dbt, Snowflake, SQL, Jinja macros
Github Public Link - [HubSpot Assessment]

Environment Setup

Warehouse: Snowflake
Transformation Layer: dbt-styled SQL (Jinja templating + modular layering)
Source Data: Four CSVs uploaded into Snowflake schema:listings.csv, calendar.csv, generated_reviews.csv, amenities_changelog.csv
Seed - Each CSV was staged into Snowflake tables.

Data Model Architecture

Design Philosophy

Seed - Ingestion of Data in Snowflake
Staging Layer (stg_) — Cleans and standardizes raw CSV data.
Intermediate Layer (int_) — Enriches and combines data across sources.
Mart Layer (fct_) — Builds a final business-ready table for analytical use.
Published Layer for Analyis - Use the mart table in Snowflake to query for the analysi

Overview of the Project

1. **Project Setup**
   - Initialized dbt project; configured dbt_project.yml and profiles.yml for Snowflake connections
   - Created database and schemas for staging, intermediate, and production layers.
2. **Data Ingestion**
   - Added seed files to load CSV source data into Snowflake.
3. **Staging Layer**
   - Built stg_listings, stg_calendar, stg_generated_reviews, stg_amenities_changelog.
   - Cleaned, standardized, and cast data types for analysis readiness.
4. **Intermediate Layer**
   - Created int_reviews_with_scores, int_listing_enriched, int_calendar_enriched.
   - Aggregated reviews, combined listing details, and enriched calendar data.
5. **Production Layer / Fact Table**
   - Built fct_listing_daily_metrics at daily/listing grain.
   - Derived metrics: daily_revenue, occupancy_ratio, and review-based KPIs.
6. **Macros (Kept on adding on the go)**
   - Developed reusable Jinja macros: column_cleaner, date_cast, format_price, null_handler, safe_divide, source_ref.
   - Refactored models to leverage macros for consistency.
7. **Testing**
   - Added schema tests (not_null, unique, accepted_values) via YAML.
   - Implemented manual business logic tests (e.g., base_price_test).
8. **Analyses**
   - Added SQL scripts in analyses for key business questions: amenity revenue, neighborhood pricing, maximum stay, and amenities-based stay analysis.
   - Executed queries to verify results.
9. **Finalization**
   - Verified end-to-end workflow: seed → staging → intermediate → production → testing → analysis.
   - Documented project structure, assumptions, macros, and testing strategy.

Project Description

0. Understand the problem statement and data file

Data Sources Overview

Table	Key Columns	Description
LISTINGS	listing_id, host_id, amenities, price, review_scores_rating	Core property details
CALENDAR	listing_id, date, available, reservation_id, price	Daily availability & pricing
GENERATED_REVIEWS	listing_id, review_score, review_date	Synthetic review data
AMENITIES_CHANGELOG	listing_id, change_at, amenities	Historical amenity updates

Starting of the project

Began by understanding the problem statement and the source data provided. I reviewed all CSVs for Listings, Calendar, Reviews, and Amenities Changelog to identify key columns and data types. I then designed the dbt project structure to include staging, intermediate, and production layers to ensure a clean, modular, and maintainable workflow. I also configured the Snowflake connection and verified the environment using dbt debug

Decision on which all details are required in the final mart table

I first identified the business metrics required: daily revenue, occupancy, and review-based KPIs. Then I traced which columns from the raw and intermediate tables would support these metrics. For example, daily_price and is_booked from the calendar table feed daily_revenue, while review_score_rating and total_reviews feed review KPIs. Additional contextual columns like neighborhood, property_type, and amenities were included to support segmentation in analyses.

1. Project Setup

1. Initialized a new dbt project using dbt init hubspot_ae_assessment.
2. Configured dbt_project.yml to define folder paths for staging, intermediate, production, and analyses models.
3. Configured profiles.yml for Snowflake, including credentials, role, warehouse, database, and schema information.
4. Created Snowflake database and dedicated schemas for staging, intermediate, and production layers to maintain environment separation.
5. Verified dbt connection to Snowflake using dbt debug to ensure connectivity and permissions.
6. Structured the dbt project with folders for models, macros, analyses, snapshots, seeds, and tests for maintainability.
7. Documented the project objectives, assumptions, and workflow in the README for clarity.

---

2. Data Ingestion

1. Prepared CSV source data files for listings, calendar, reviews, and amenities changelog.
2. Added seed files under the seeds folder to load source data into Snowflake.
3. Configured dbt_project.yml to define seed behavior (materialization and schema).
4. Ran dbt seed to populate raw source tables in Snowflake for staging.
5. Verified loaded data for completeness and correctness by comparing row counts and sample records.
6. Ensured consistent naming conventions and proper data types for downstream transformations.

---

3. Staging Layer

1. Built staging models: stg_listings, stg_calendar, stg_generated_reviews, stg_amenities_changelog.
2. Data cleaning transformations: trimming strings, standardizing column names, and handling nulls.
3. Casted columns to appropriate types (e.g., dates, floats) for analysis readiness.
4. Validated transformations by running sample queries and checking data integrity.
5. Ensured consistent schema across staging tables to simplify intermediate transformations. Added comments in SQL files to describe transformation logic for clarity.

---

4. Intermediate Layer

1. Created int_reviews_with_scores to compute aggregated metrics like total reviews and average review scores per listing.
2. Built int_listing_enriched to combine listing details, review aggregates, and latest amenities.
3. Developed int_calendar_enriched to enrich calendar data with availability flags, reservation status, and daily prices.
4. Applied joins and window functions to handle time-sensitive data, like amenities changelog.
5. Ensured consistency of data types and naming conventions across intermediate tables.
6. Verified intermediate outputs for correctness by comparing with raw data and manual calculations.
7. Prepared intermediate tables as inputs for production/fact tables.

---

5. Production Layer / Fact Table

1. Built fct_listing_daily_metrics at the daily/listing grain to support revenue, occupancy, and review KPIs.
2. Derived metrics such as daily_revenue, occupancy_ratio, and review-based scores.
3. Combined data from intermediate listings and calendar tables to provide a comprehensive fact table.
4. Ensured proper handling of missing values and anomalies in pricing or availability.
5. Materialized the fact table as a Snowflake table for downstream analytics.
6. Verified fact table results by running sample queries for key listings and metrics.
7. Documented the fact table design, including grain, keys, and metrics, for analyst reference.

---

6. Macros (Kept on adding on the go)

1. Developed reusable Jinja macros for common transformations:
    - column_cleaner, date_cast, format_price, null_handler, safe_divide, source_ref.
2. Refactored staging, intermediate, and production models to leverage these macros.
3. Improved consistency and reduced duplication in SQL code across models.
4. Facilitated easier maintenance and future enhancements.
5. Documented macros with usage examples in the macros folder.
6. Tested macros individually and within models to ensure correctness.

---

7. Testing

1. Added **schema tests** via YAML (not_null, unique, accepted_values, accepted_range) for key columns.
2. Created **manual/custom tests** for business logic, e.g., base_price_test.
3. Ran tests using dbt test to validate data quality and enforce constraints.
4. Verified test results and corrected data/model issues where necessary.
5. Ensured all staging, intermediate, and production models passed tests before analyses.
6. Documented test logic and expectations for future reference.

---

8. Analyses

1. Added SQL scripts under the analyses folder for business use cases:
    - Amenity revenue by month.
    - Neighborhood pricing comparison.
    - Maximum possible stay per listing.
    - Maximum stay for listings with lockbox + first aid kit.
2. Executed queries in Snowflake to validate metrics and answer business questions.
3. Documented assumptions and methodology for each analysis.
4. Provided reusable queries for analysts to explore additional dimensions.
5. Ensured analyses were aligned with the grain and metrics defined in fct_listing_daily_metrics.

SQL Ques 1

Write a query to find the total revenue and percentage of revenue by month segmented by whether or not air conditioning exists on the listing

```
SELECT
   DATE_TRUNC('month', date) AS month,
   CASE
       WHEN amenities ILIKE '%Air conditioning%' THEN 'With AC'
       ELSE 'No AC'
   END AS has_air_conditioning,
   SUM(daily_revenue) AS total_revenue,
   ROUND(
       100 * SUM(daily_revenue)
       / SUM(SUM(daily_revenue)) OVER (PARTITION BY
DATE_TRUNC('month', date)),
       2) AS pct_revenue
FROM HUBSPOT_AE_ASSESSMENT.PUBLIC_PROD.FCT_LISTING_DAILY_METRICS
GROUP BY 1, 2
ORDER BY 1, 2;
```

SQL Ques 2

Write a query to find the average price increase for each neighborhood from July 12th 2021 to July 11th 2022.

```
WITH price_by_listing AS (
   SELECT
       Listing_id,
       Neighborhood,
       MAX(CASE WHEN date = '2021-07-12' THEN daily_price END) AS
price_start,
       MAX(CASE WHEN date = '2022-07-11' THEN daily_price END) AS
price_end
   FROM HUBSPOT_AE_ASSESSMENT.PUBLIC_PROD.FCT_LISTING_DAILY_METRICS
   GROUP BY listing_id, neighborhood
)

SELECT
   Neighborhood,
   ROUND(AVG(price_end - price_start), 2) AS avg_price_increase
FROM price_by_listing
GROUP BY neighborhood
ORDER BY neighborhood;
```

SQL Ques 3.a

Write a query to find the maximum duration one could stay in each of these listings, based on the availability and what the owner allows.

```
WITH ordered AS (
   SELECT
       Listing_id,
       Date,
       Maximum_nights,
       Is_available,
       ROW_NUMBER() OVER (PARTITION BY listing_id ORDER BY date) AS
rn
   FROM HUBSPOT_AE_ASSESSMENT.PUBLIC_PROD.FCT_LISTING_DAILY_METRICS),
-- Only keep available days and calculate "gap" to detect streaks
available AS (
   SELECT
       Listing_id,
       Date,
       Maximum_nights,
       Rn,
       ROW_NUMBER() OVER (PARTITION BY listing_id ORDER BY date) - rn
AS streak_grp
   FROM ordered
   WHERE is_available = 1),
-- Count streak lengths
streaks AS (
   SELECT
       Listing_id,
       MIN(date) AS start_date,
       MAX(date) AS end_date,
       COUNT(*) AS streak_length,
       MAX(maximum_nights) AS max_allowed
   FROM available
   GROUP BY listing_id, streak_grp),
-- Cap each streak by maximum_nights
streaks_capped AS (
   SELECT
       Listing_id,
       LEAST(streak_length, max_allowed) AS max_possible_stay
   FROM streaks),
-- Get the longest possible stay per listing
listing_max_stay AS (
   SELECT
       Listing_id,
       MAX(max_possible_stay) AS max_possible_stay
   FROM streaks_capped
   GROUP BY listing_id)
```

```
SELECT * FROM listing_max_stay
ORDER BY listing_id;
```

SQL Ques 3.b

Write a variation of the maximum duration query above for listings that have both a lockbox and a first aid kit listed in the amenities.

```
WITH filtered_listings AS (
   -- Keep only listings that have both lockbox and first aid kit in
amenities
   SELECT *
   FROM HUBSPOT_AE_ASSESSMENT.PUBLIC_PROD.FCT_LISTING_DAILY_METRICS
   WHERE LOWER(amenities) ILIKE '%lockbox%'
     AND LOWER(amenities) ILIKE '%first aid kit%'),
ordered AS (
   SELECT
       Listing_id,Date,Maximum_nights,Is_available,
       ROW_NUMBER() OVER (PARTITION BY listing_id ORDER BY date) AS
rn
   FROM filtered_listings),
-- Only keep available days and detect streaks
available AS (
   SELECT
       Listing_id,Date,Maximum_nights,Rn,
       ROW_NUMBER() OVER (PARTITION BY listing_id ORDER BY date) - rn
AS streak_grp
   FROM ordered WHERE is_available = 1),
-- Count streak lengths per listing
streaks AS (
   SELECT
       Listing_id,MIN(date) AS start_date,MAX(date) AS end_date,
       COUNT(*) AS streak_length,
       MAX(maximum_nights) AS max_allowed
   FROM available
   GROUP BY listing_id, streak_grp
),
-- Cap each streak by maximum_nights
streaks_capped AS (
   SELECT
       Listing_id,
       LEAST(streak_length, max_allowed) AS max_possible_stay
   FROM streaks),
-- Maximum stay per listing
listing_max_stay AS (
   SELECT
       Listing_id,MAX(max_possible_stay) AS max_possible_stay
   FROM streaks_capped
   GROUP BY listing_id)
```

```sql
-- Return max possible stay for all filtered listings
SELECT *
FROM listing_max_stay
ORDER BY listing_id;
```

9. Finalization

1. Verified the complete end-to-end workflow: seed → staging → intermediate → production → testing → analyses.
2. Documented the project structure, assumptions, macros, and test strategy in README.
3. Reviewed models and SQL for readability, maintainability, and best practices.
4. Confirmed all outputs matched expected business logic and data quality standards.
5. Prepared the project for submission, including clear instructions for running dbt commands and tests.

Additional Resources/Codes

Stage Tables

Stage Table	What it stores in simple terms
stg_listings	All the details about the properties: ID, host info, type of room, price, amenities, reviews info. Columns are cleaned and data types fixed.
stg_calendar	Daily availability for each listing. Tells you if a listing is available or reserved, what the daily price is, min/max nights, etc.
stg_generated_reviews	Raw reviews for each listing: who reviewed what, review score, and review date.
stg_amenities_changelog	Historical record of amenities updates per listing (e.g., when a new feature like “Air Conditioning” or “Lockbox” was added).

Intermediate Tables

Intermediate Table	What it stores in simple terms
int_reviews_with_scores	Aggregates review information per listing: total number of reviews and average rating. Makes it easier to calculate metrics.
int_listing_enriched	Combines listing details, aggregated reviews, and the latest amenities. Basically a full profile of each property.
int_calendar_enriched	Enriches the calendar with flags for availability and reservation status, along with cleaned daily prices. Makes it easy to analyze booking patterns.

Mart Tables

Mart Table	What it stores in simple terms
fct_listing_daily_metrics	Each row represents a single listing for a single day. Contains: • Daily revenue (how much the property made that day) • Occupancy (booked or available) • Price info • Review scores • Amenities info • Derived metrics like occupancy ratio.

Macros Used

Macro	Purpose
column_cleaner(column_name)	Standardizes column names (lowercase, trimmed)
date_cast(column_name)	Converts to date
format_price(column_name)	Converts string price to float
null_handler(column_name, default_value)	Replace NULLs with defaults
safe_divide(numerator, denominator)	Prevent division by zero
source_ref(model_name)	Reference another dbt model

Test Cases in the Schema_test.yml

Column	Test	Use Case
listing_id	not_null	Ensure each row has a valid listing identifier; no missing keys.
date	not_null	Ensure each record has a valid date; prevents gaps in time-series analysis.
is_available	accepted_values [0,1]	Validates that availability is encoded correctly; prevents invalid flags.
is_booked	accepted_values [0,1]	Validates that booking status is correctly recorded; prevents invalid flags.

```
CREATE WAREHOUSE IF NOT EXISTS COMPUTE_HS
  WITH WAREHOUSE_SIZE = 'XSMALL'
  AUTO_SUSPEND = 60
  AUTO_RESUME = TRUE
  INITIALLY_SUSPENDED = TRUE;
```

Creates a compute warehouse (Snowflake’s term for a “virtual cluster” that runs queries) called COMPUTE_HS.The warehouse will be very small (XSMALL), meaning it has minimal compute resources. Small warehouses are cheaper but slower for large queries.

AUTO_SUSPEND = 60
If no queries run for 60 seconds, Snowflake will automatically pause the warehouse to save costs.

AUTO_RESUME = TRUE
When a new query is submitted, Snowflake will automatically start the warehouse again if it’s paused.

INITIALLY_SUSPENDED = TRUE

When the warehouse is first created, it will start in a suspended/paused state, so it doesn’t consume resources until needed.

Summary

Pointers	Usecase
Purpose of the project	To transform raw rental property data into analysis-ready datasets that allow business analysts to measure revenue, occupancy, and review/amenity trends.
Tools used and why?	Snowflake for storage and computation due to scalability, dbt for modular ETL development, and SQL/Jinja for transformations.
Structure of dbt project	I used separate folders for staging (cleaning raw data), intermediate (enrichments), production (fact tables), macros, analyses, and tests.
Seeding	I added them as dbt seeds, which loaded them into Snowflake staging tables, verified row counts, and ensured consistent column names/types.
Usefulness of multiple layer architecture	Multiple layers help separate concerns: staging handles cleaning, intermediate handles enrichments/aggregations, and production builds business-ready fact tables.
Data Quality checks	I added schema tests (not_null, unique, accepted_values) and manual tests for business logic, then ran dbt test to ensure data quality.
Transformations	I analyzed business requirements and source data; for example, casting prices to float, cleaning strings, calculating occupancy ratios, and capping maximum stay by owner rules.
Use of Macros	I created reusable Jinja macros for common transformations like cleaning columns, casting dates, handling nulls, and safe divisions to ensure consistency.
Design of fact table	I chose a daily/listing grain because most metrics (revenue, occupancy, reviews) are daily per listing, enabling time-series and segmented analyses.
Verification of Analysis	I executed queries against fct_listing_daily_metrics, cross-checked calculations with raw data, and confirmed that outputs aligned with business logic expectations.

Complete Folder structure of dbt