

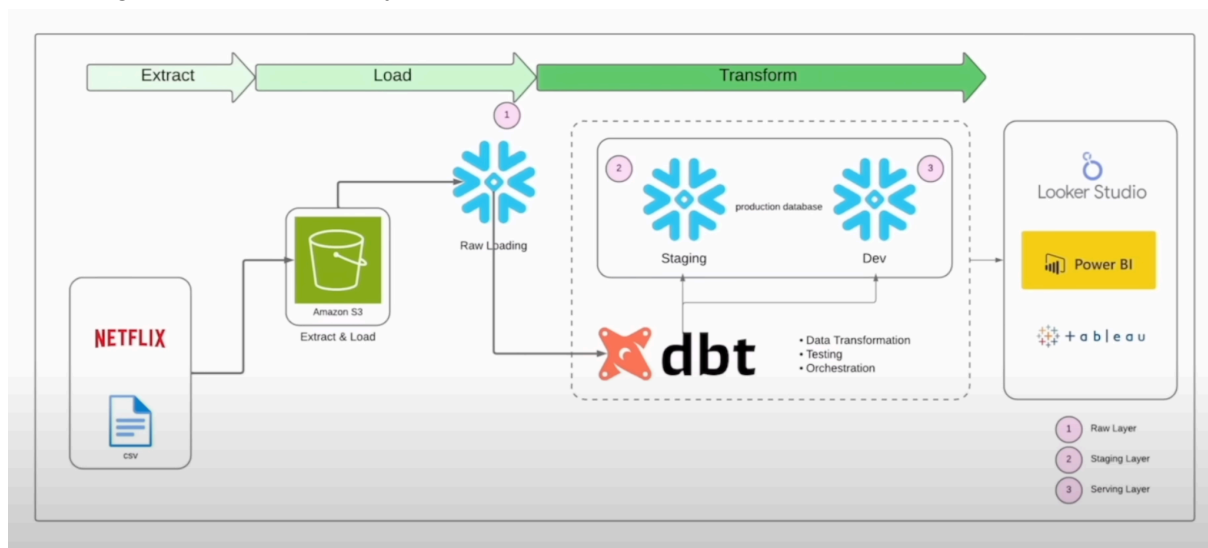
NETFLIX DATA ANALYSIS

In this project, we will analyse Netflix data and then use the dashboard to present a few analysis which we did on it.

Main tools which we will use are:

- **Dataset from Netflix**
- **AWS S3 bucket and other AWS services**
- **Snowflake Data Warehouse**
- **DBT Tool (Data Build Tool)**
- **Tableau**

This is the pipeline which we will follow to create an ETL pipeline to make a visually appealing dashboard for analysis.

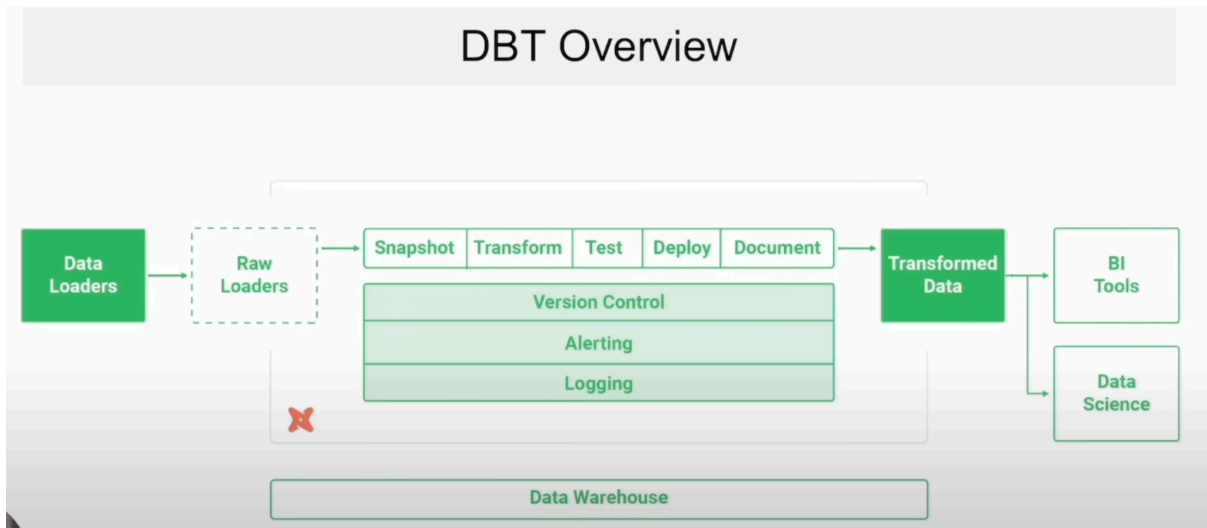


Little bit about DBT (Data Build Tool):

DBT is a transformation tool which allows us to write modular SQL queries that help us transform raw data into analytics-ready data models.

In the Data Analytics domain, transformation means that we are applying Business logic on top of the data to get meaningful observations from the data and in turn increase the profit margins.

DBT focuses on the Transformation part of the ELT (Extract, Load, Transform) pipeline.



DBT helps to transform the data using SQL queries, it can also add new columns to the tables. DBT works with data warehouses.

In Summary:

DBT = SQL + Git + Automation + Documentation + Testing

Why to use DBT:

- Reusability
- Modularity
- Dependency Management
- Development Workflow
- Testing Framework

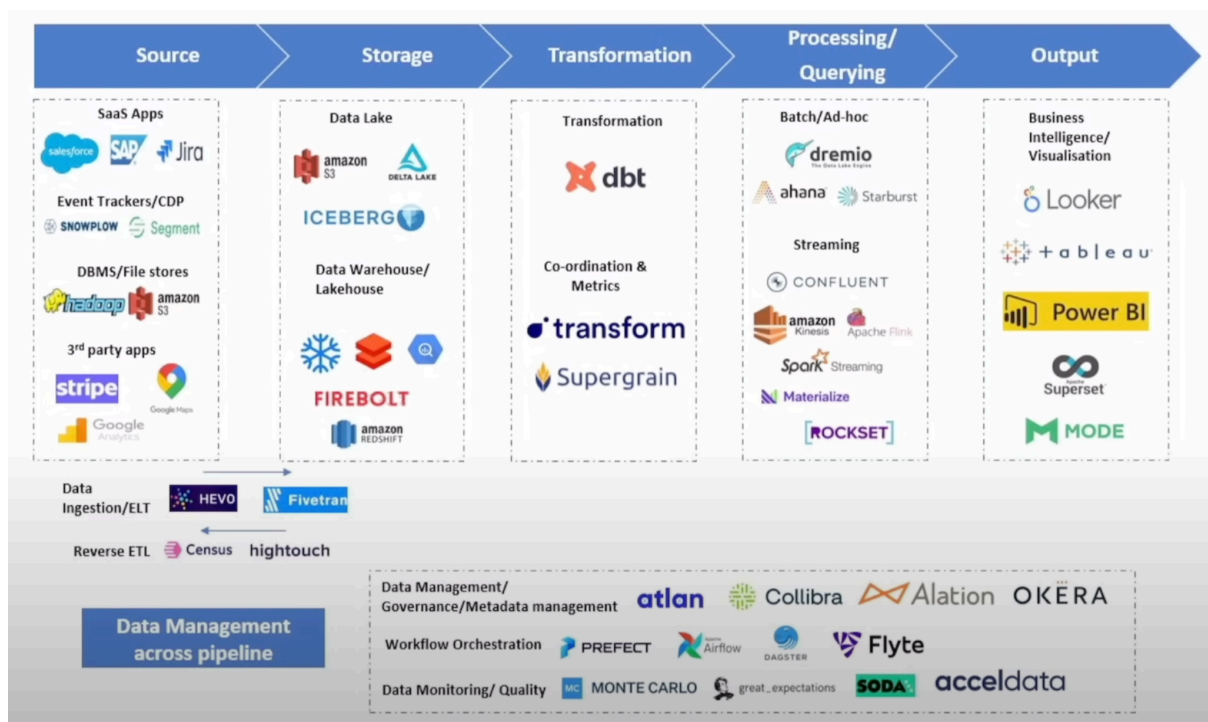
Data Warehouse Vs Data Lake Vs Data Lakehouse

Comparison Table

Feature	Data Warehouse	Data Lake	Data Lakehouse
Data Type	Structured only	Structured, Semi-structured, Unstructured	All types (Structured + Raw)
Storage Format	Tables (SQL-like schemas)	Files (Parquet, JSON, CSV, Images, etc.)	Files + Table abstraction (Delta, Iceberg)
Cost	High (compute + storage)	Low (cheap object storage)	Moderate (balances compute/storage)
Performance	Fast for SQL queries	Slow unless engineered	Optimized for both analytics and ML
Flexibility	Low (schema-on-write)	High (schema-on-read)	Medium (hybrid of both)
Use Case	BI, analytics, reporting	ML, raw data analysis, batch processing	Unified BI + ML + real-time analytics
Examples	Snowflake, Redshift, BigQuery	S3, ADLS, Hadoop	Databricks, Delta Lake, Apache Iceberg
Data Governance	Strong (ACID, RBAC)	Weak (file permissions)	Improved governance & transaction support

Up until this point we have been using legacy tools but now we have new tools and for each of the steps, the tools are mentioned below. These are the modern tools which help us solve modern business problems and perform analysis on them.

Modern Data Stack:



Data link: <https://grouplens.org/datasets/movielens/>

We have downloaded the 20M dataset from the above link. Now the next step is to upload this data on Snowflake.

There are 3 ways to upload data on Snowflake:

1. Go to Snowflake, directly create a table and upload the files, Snowflake does everything on its own.
2. Upload the data on Amazon S3 bucket and then connect this bucket to Snowflake.
3. Use the DBT tool to load the data directly using a seed concept.

For this project, we will use the second approach, we will upload the data on a S3 bucket and then fetch the data from there.

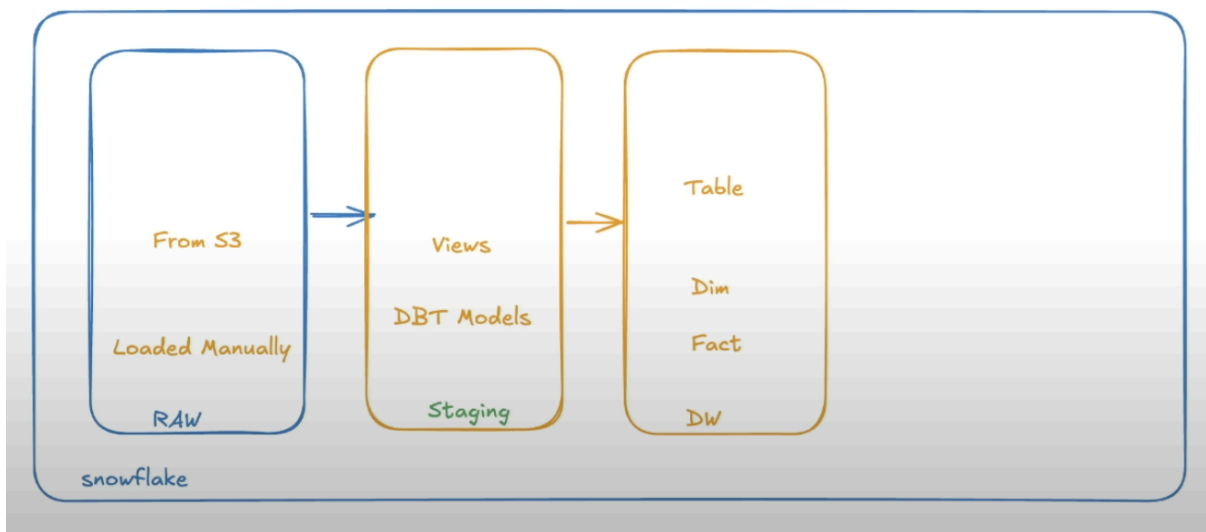
First we will extract the data from the downloaded zip file and then create a new S3 bucket on AWS and upload all the relevant files on the bucket except the README file.

DBT models are SQL statements which transform the data. Each and every model in Snowflake produces a table or view in the data warehouse and it can reference other models, creating a dependency graph.

A view in Snowflake is a view on top of the data source which will have the latest records in it, in the backend, a normal SQL is run.

For example, if you have a movies dataset inside your project folder, and you create a view, then based on the columns selected, the size of view will change but if you create a copy table then it will have same size as that of the original table and if we make changes to the original table, we cannot see the changes in the created table but we can see the changes inside a view.

This is a basic structure we will be using, till now we have created basic views in Snowflake using DBT Models and Snowflake.



There are two ways to refer to a table in DBT, 'ref' and 'src' we use 'ref' to refer to a table or model which is created by using a DBT model and we use 'src' when we are referencing raw data to create views or tables.

We made some changes to the dbt_project.yml file:

```
32  models:
33    netflixproject:
34      +materialized: view
35      dim:
36        +materialized: table
37
```

This simply means that, each and every file under the netflixproject will be treated as a view but if the file exists inside the dim folder, it will be treated as a table.

Using the dbt models, we can create tables with modifications performed on them, for clean analysis.

We cleaned our project setup a bit.

We removed the SRC_tables from the RAW folder because we created views in the DEV Schema and we can access the table view from there and we can also create new tables in the dim folder for analysis.

To run a command in DBT, we use this:

→ dbt run --model

We use the above code to run all the SQL scripts which we have written inside different files.

→ `du run -model <file-name-w/o-extension>`

We use this code to run an individual file, so that we don't have to process loads of data each time and when we run the code, we give only the name of the file and not the file extension.



2025-07-07 3:54pm

2025-07-07 4:08pm

Databases

Worksheets



Search objects



MOVIELENS

DEV

Tables

DIM_MOVIES

Views

SRC_GENOME_SCORE

SRC_GENOME_TAGS

SRC_LINKS

SRC_MOVIES

SRC_RATINGS

SRC_TAGS

INFORMATION_SCHEMA

PUBLIC

RAW

Tables

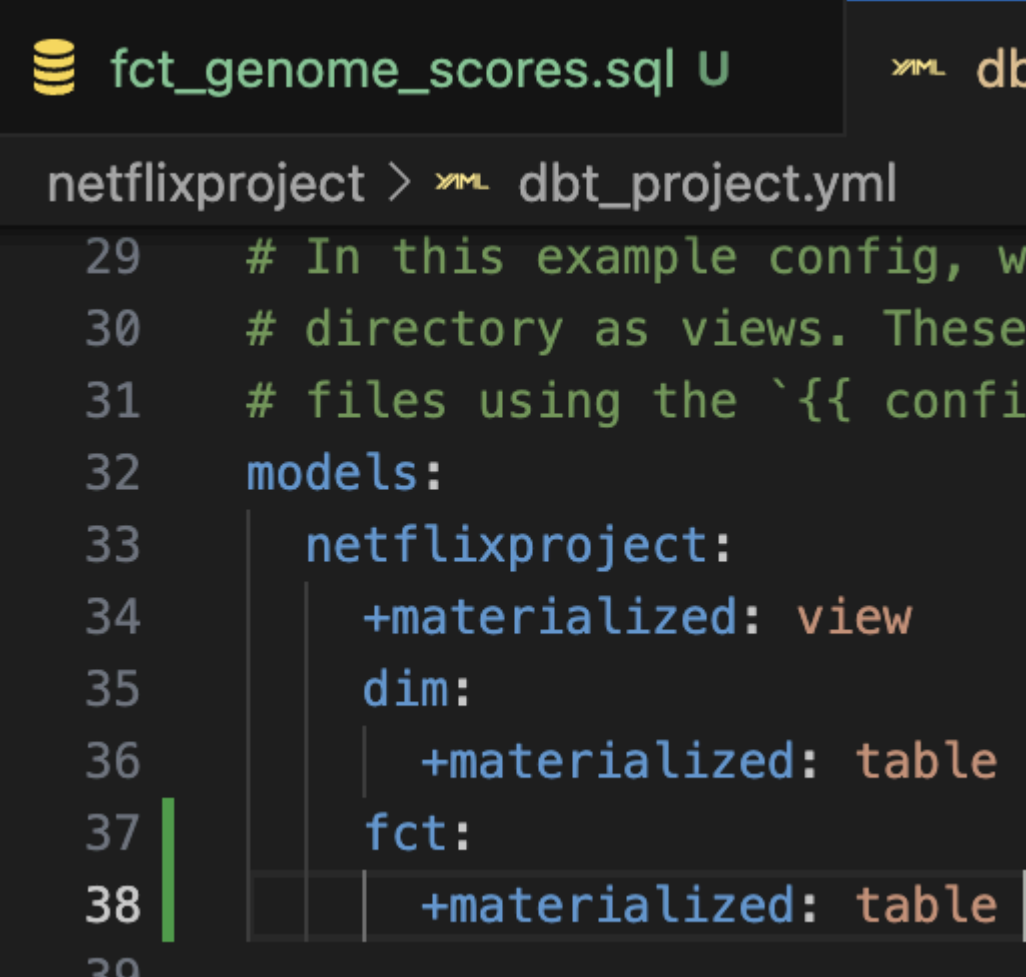
RAW_GENOME_SCORES

RAW_GENOME_TAGS

RAW_LINKS



Now we did not want the fct_genome_scores to be a view therefore we added the below to the models section in the dbt_project.yml file so that the output will be stored as a table.

A screenshot of a code editor with a dark background. The top of the editor shows a tab labeled 'fct_genome_scores.sql U' with a database icon on the left and 'dbt' on the right. Below the tab, the text 'netflixproject > dbt_project.yml' is visible. The main content is a YAML configuration file. Lines 29-31 are comments: '# In this example config, w', '# directory as views. These', and '# files using the `{{ confi'. Line 32 is 'models:'. Line 33 is ' netflixproject:'. Line 34 is ' +materialized: view'. Line 35 is ' dim:'. Line 36 is ' +materialized: table'. Line 37 is ' fct:'. Line 38 is ' +materialized: table'. Line 39 is partially visible at the bottom. A green vertical cursor is on line 37.

```
29  # In this example config, w
30  # directory as views. These
31  # files using the `{{ confi
32  models:
33    netflixproject:
34      +materialized: view
35    dim:
36      +materialized: table
37    fct:
38      +materialized: table
39
```

There are 5 different types of materialized models in DBT for data transformation, here is a brief summary of them:

Summary Table				
Materialization	Physical Object	Rebuild Behavior	Use Case	
view	View	Recomputed on query	Lightweight transformations	
table	Table	Full rebuild every run	Final reporting data	
incremental	Table	Adds new data only	Large fact tables, append-only logs	
ephemeral	None (CTE only)	Inlined during compile	Reusable subqueries	
snapshot	Table	Tracks changes over time	Slowly changing dimensions	

To run only the select statement in the incremental model, that is to get the updated table each time for doing an analysis, we run the following command.

→ `dbt run --select <file-name-w/o-extension>`

We are now running an ephemeral materialized model in DBT, this config does not create a new table or fact table, this will only create a table in the backend which we can access if we have any particular use case for that, else it will not show up in the front end anywhere.

Now we are going to create a seed using DBT.

Seeds in DBT are the CSV files that we can load in our data warehouse. They are very useful in static tasks like referencing data for country codes, ZIP codes or product categories.

Seeds are useful when we want to quickly create a table without writing SQL code for creating a new table and materializing it or referencing it with any other tables. These tables will not change in future.

To run a seed command in DBT, we simply write `dbt seed`.

As we have mapped inside our `dbt_project.yml` file that each file inside the seed folder is a seed therefore, dbt automatically knows which files to run and it will create a quick table for our reference in the tables section of DBT, under the selected schema.

Now we will configure sources in DBT.

Sources in our raw data represent the raw data which is loaded in our warehouse. This gives us better control on referencing data because we can use our own naming conventions.