Data Warehousing project report:  
  
Tutify Data Warehouse Project Report

1. Title Page

Project Title: Tutify Educational Data Warehouse  
Submitted By: Muhammad Mashhood uddin and Hayat Ullah

Video demo: <https://www.youtube.com/watch?v=S2CFyDV7EPs&ab_channel=MUHAMMADMASHHOOD>

2. Abstract

The Tutify Data Warehouse is designed to enable centralized analytics for a virtual tutoring platform. The warehouse consolidates payment and teacher payout data, providing insights for business KPIs like revenue tracking, teacher performance, and student behavior. This report outlines the architecture, ETL pipeline, data modeling decisions, and analytical use cases that form the basis of the implemented data mart.

Tutify Academy lacks a unified system to analyze student revenue and teacher compensation trends across academic subjects, dates, and payment channels. Without a centralized analytical model, it is difficult to answer key questions such as revenue by subject, teacher payout efficiency, and region-wise income.

3. Introduction

Tutify is a virtual tutoring academy that handles multi-dimensional relationships between teachers, students, and subjects. To support analytical reporting and performance insights, we built a cloud-based sales-focused data mart using Snowflake and Apache Airflow.

Link to know more about Tutify: <https://tutify.org/>

***This data warehouse pipeline supports:***

* Centralized analytics on student payments and teacher payouts.
* Monitoring metrics like revenue, top subjects, teacher payouts, etc.
* A flexible schema supporting future integration of scheduling, performance, and content engagement data.

4. Business Process Overview

The pipeline is modelled on two core business processes:

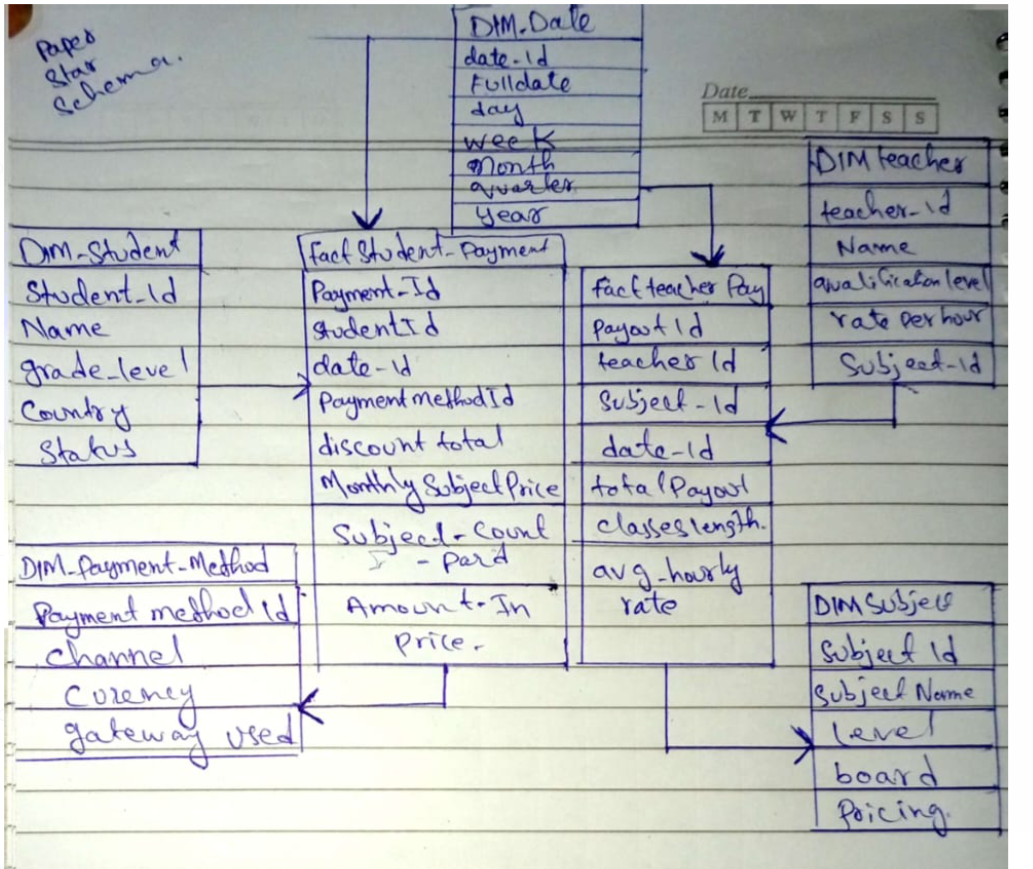
1. Student Payments: Students pay per subject per month based on pre-defined rates.
2. Teacher Payouts: Teachers are paid based on hours taught, per subject and level.

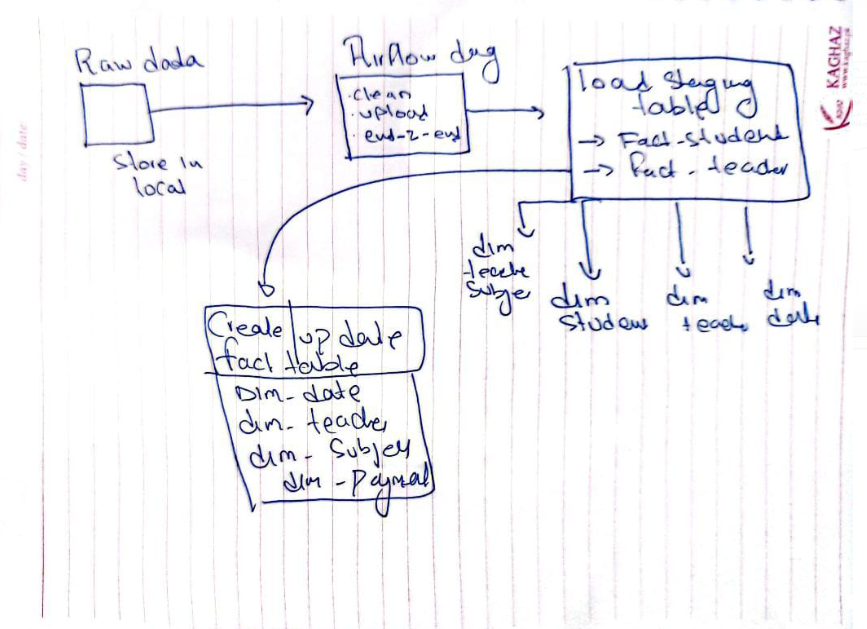
These processes are supported by dimensional data such as date, subjects, teachers, students, and payment methods.

5. Grain of the Data Mart

The fact table grain is one row per student payment per subject per date, and one row per teacher payout per subject per date. This granularity allows flexible aggregation for monthly reports, subject-level analysis, and teacher performance tracking

# 6. Star Schema Design





**Business Insights Enabled**

* Monthly revenue and teacher cost trends
* Revenue by grade, subject, board, and country
* Payouts by teacher qualification or subject
* Cost-to-income ratio analysis per subject
* Impact of payment channels or currency on revenue

# Data Generation and Validation Process

For the Tutify educational platform, all required data was synthetically generated using Python scripts. The goal was to create realistic yet controlled datasets that reflect actual business processes while enabling thorough testing of ETL logic, schema design, and data quality checks.

**Data Generation Overview:**

Data was generated for all key dimension and fact tables in the star schema, including:

DIM\_STUDENT, DIM\_TEACHER, DIM\_SUBJECT, DIM\_DATE, DIM\_PAYMENT\_METHOD, DIM\_LOCATION, DIM\_TEACHER\_SUBJECT\_LEVEL

FACT\_STUDENT\_PAYMENT, FACT\_TEACHER\_PAYOUT

**Tools Used**: Python (pandas, faker, random, datetime)

**Volume:**

DIM\_STUDENT: 1500 rows

DIM\_TEACHER: 600 rows

FACT\_STUDENT\_PAYMENT: 30,000+ rows

FACT\_TEACHER\_PAYOUT: 12,000+ rows

**Deliberate Errors Introduced:**

Missing values in critical columns (e.g., payment\_method\_id, rate\_per\_hour, amount\_original)

Typo variations in names and categories (e.g., pakistan vs Pakistan, tution vs tuition)

Outliers in numeric fields like bonus, total\_payout, and amount\_converted\_pkr

Redundant and irrelevant rows in raw CSVs

Mixed case and extra whitespace in string fields

These anomalies were designed to simulate real-world messy data and validate the robustness of the ETL pipeline.

**ETL and Cleaning Approach:**

Outliers and type mismatches were removed or coerced using pandas.to\_numeric()

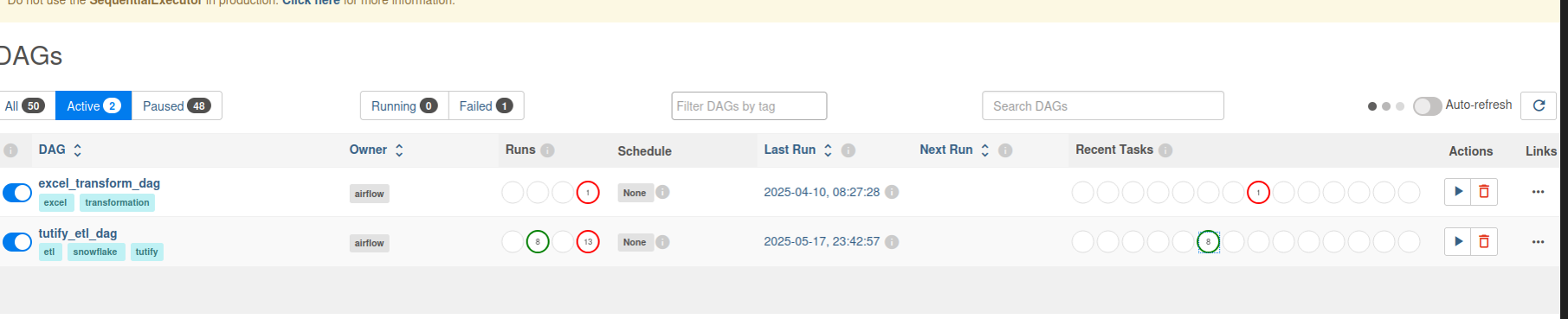
# 7. ETL/ELT Pipeline Design

**Tools Used**

* **Apache Airflow** for pipeline orchestration
* **Pandas** for data cleaning
* **Snowflake** as the cloud data warehouse
* **Local FileSystem** as staging layer

# 8. Pipeline Stages and DAG Overview

**Step 1: Data Cleaning Tasks**

* Raw CSV files cleaned using Python (Pandas)
* Nulls, outliers, types fixed
* 

**Step 2: Upload Clean Files to Snowflake Stage**

* PUT command pushes cleaned files to my\_stage

**Step 3: Load Clean Files into Staging Tables**

* COPY INTO from stage into STG\_STUDENT\_PAYMENT and STG\_TEACHER\_PAYOUT

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Step 4: Create/Update Fact Tables**

* MERGE operation used to populate FACT\_STUDENT\_PAYMENT and FACT\_TEACHER\_PAYOUT

**Step 5: Demonstrate Timestamp-Based Ingestion**

Demonstrated by checking updated timestamp on fact table before and after uploading modified CSV and triggering DAG.

A white sheet with blue lines

AI-generated content may be incorrect.

**9. Data Cleaning & EDA Summary**

* All null values handled through imputation/logical mapping
* Outliers cleaned using statistical thresholds
* Feature transformation included AgeGroup, Gender (from names), and derived fields

**10. Final Tables Snapshot in Snowflake**

Include screenshots showing final table structure in Snowflake:

* STG\_STUDENT\_PAYMENT
* STG\_TEACHER\_PAYOUT
* FACT\_STUDENT\_PAYMENT
* FACT\_TEACHER\_PAYOUT

A screenshot of a computer

AI-generated content may be incorrect.

**11. Dashboarding and Analytics**

* Sample dashboard queries written in Snowflake Worksheets
* Future scope: integrate Streamlit for interactive UI

Example Queries:

* Monthly revenue vs teacher payout
* Subject-wise student payments
* Top-earning teachers
* A screenshot of a computer

  AI-generated content may be incorrect.

**12. Challenges and Learnings**

* Handling circular dependencies in Airflow DAG
* Managing schema mismatch during Snowflake COPY
* Manual timestamp checks for proving pipeline effect

**13. Future Extensions**

* Add tracking of session attendance
* Integrate lesson scheduling and curriculum completion
* Add performance metrics per teacher/student

# Problems encountered.

**1. Broken DAG due to Operator Chaining (>>)**

* **Problem:**  
  Used list >> list chaining in Airflow, which caused a TypeError because >> only works between tasks or task groups, not lists directly.
* **Solution:**  
  We rewrote the chaining using for loops or separated chains per task to maintain linear and readable dependency structure.

**2. Snowflake STAGE PUT Error – “No current schema”**

* **Problem:**  
  Snowflake threw 090106 (22000) error because PUT command didn’t have a schema set in the session.
* **Solution:**  
  We added explicit commands in every SQL block:

USE WAREHOUSE TUTIFY\_WH;

USE DATABASE TUTIFY\_DB;

USE SCHEMA TUTIFY\_SCHEMA;

**3. Airflow Webserver Crash due to PID Conflict**

* **Problem:**  
  Restarting the Airflow webserver failed because the port was still in use or old processes were not killed.
* **Solution:**

pkill -f airflow-webserver

and manually cleaned stale PID files before restarting.

**4. FileNotFoundError for CSVs during PythonOperator**

* **Problem:**  
  The cleaning functions in the DAG failed because CSV files were not placed in the expected directory (/home/uthred/airflow/data/raw).
* **Solution:**  
  correct paths and ensured files were available before running the DAG.

**5. Snowflake COPY INTO Fails – “No warehouse selected”**

* **Problem:**  
  Missing USE WAREHOUSE in the COPY INTO task caused 000606 (57P03) errors.
* **Solution:**  
  Added USE WAREHOUSE TUTIFY\_WH; explicitly in all SQL tasks.

**6. Wrong Columns in Cleaned CSV**

* **Problem:**  
  Airflow cleaning tasks failed with KeyError because expected columns (like PAYMENT\_ID) were not present in uploaded CSVs.
* **Solution:**  
  Re-uploaded clean versions of the CSV files with **correct column names**, verified using df.columns.

**7. Old Staged Files Were Being Used**

* **Problem:**  
  Snowflake still used earlier manually uploaded files, not the freshly generated ones from Airflow.
* **Solution:**  
  Overwrite was enforced with:
* PUT ... OVERWRITE=TRUE;

and we validated by checking timestamps and content on Snowflake side.

**8. Airflow Retry Loops Hiding Root Cause**

* **Problem:**  
  Multiple retries made it difficult to debug the original issue due to noise in logs.
* **Solution:**  
  Used manual airflow tasks run with --local to isolate and quickly debug the task.

**9. Wrong Column Mapping in Fact Table**

* **Problem:**  
  The fact table was initially created without matching data types or naming conventions (e.g., mismatch in amount\_converted\_pkr).
* **Solution:**  
  updated the schema and MERGE statements to match the cleaned data exactly.

**10. Confusion Between Final vs. Manual Uploads**

* **Problem:**  
  Difficulty verifying if Airflow was using the **latest cleaned CSV** vs. earlier staged ones.
* **Solution:**  
  A clear demo plan was developed using:
  + Snapshot before DAG run
  + Snapshot after file overwrite
  + Timestamp inspection on Snowflake table

# Data governance.

**1. Data Quality Management**

**Implemented via:**

* **Cleaning & EDA stage** in Airflow DAG  
  → handled:
  + Missing values (e.g., using mode, mean, logical inference)
  + Data type inconsistencies (e.g., date parsing, float conversion)
  + Outlier detection (optional boxplot/skew removal if done)

**Data Governance Principle Applied:**

**Ensure accuracy, completeness, and consistency of data across the pipeline.**

**2. Data Lineage and Traceability**

**Implemented via:**

* Well-defined **ETL DAG with clearly named tasks** like:
  + clean\_student\_payment
  + upload\_student\_to\_stage
  + load\_stg\_student\_payment
  + create\_fact\_student\_payment

Each step is logged and timestamped, giving **traceability** from source to fact table.

**Data Governance Principle Applied:**

**Track where data comes from, how it's transformed, and where it ends up.**

**3. Data Security and Access**

**Partially Implemented via:**

* Using **Snowflake roles** like SYSADMIN
* Using **staging via my\_stage** instead of directly loading into sensitive tables

**Data Governance Principle Applied:**

**Control access to data based on roles and ensure secure staging/loading.**

**4. Metadata Management**

**Implemented via:**

* Creating a **clear star schema**
* Naming conventions (DIM\_, FACT\_, STG\_) that communicate the role of each table
* Well-defined attribute naming in dimension tables (e.g., subject\_id, payment\_method\_id, etc.)

**Data Governance Principle Applied:**

**Enable clear understanding of what each dataset and field represents.**

**5. Policy Enforcement Through Automation**

**Implemented via:**

* **Airflow orchestration** ensures cleaning always happens before loading
* **No direct overwrite** of fact tables — using MERGE allows policy-controlled updates

**Data Governance Principle Applied:**

**Automation enforces defined steps, order, and logic to maintain trust in data.**

Our Tutify ETL pipeline applies essential data governance principles such as data quality enforcement through cleaning steps, data lineage via DAG tracking, access control using Snowflake roles and staging, consistent metadata via naming conventions, and automation of policy-compliant workflows using Airflow.”

**14. Conclusion**

This data warehouse enables clear visibility into Tutify's financial flows and academic engagement. The pipeline is scalable, maintainable, and integrates cleanly with analytical tools.