**DWH project report**

**Members:**

Shahmeer Khan - 25156

Uzair Nadeem - 24928

**Schema URL:** [**https://github.com/GziXnine/Hospital\_Management\_System**](https://github.com/GziXnine/Hospital_Management_System)

**Github URL: https://github.com/shahmeerkm10/DWH\_project**

**Video URL:** [**https://youtu.be/DaL45NUHpUc?si=cpU6hLxrvlSXqgdT**](https://youtu.be/DaL45NUHpUc?si=cpU6hLxrvlSXqgdT)

**Base Idea:**

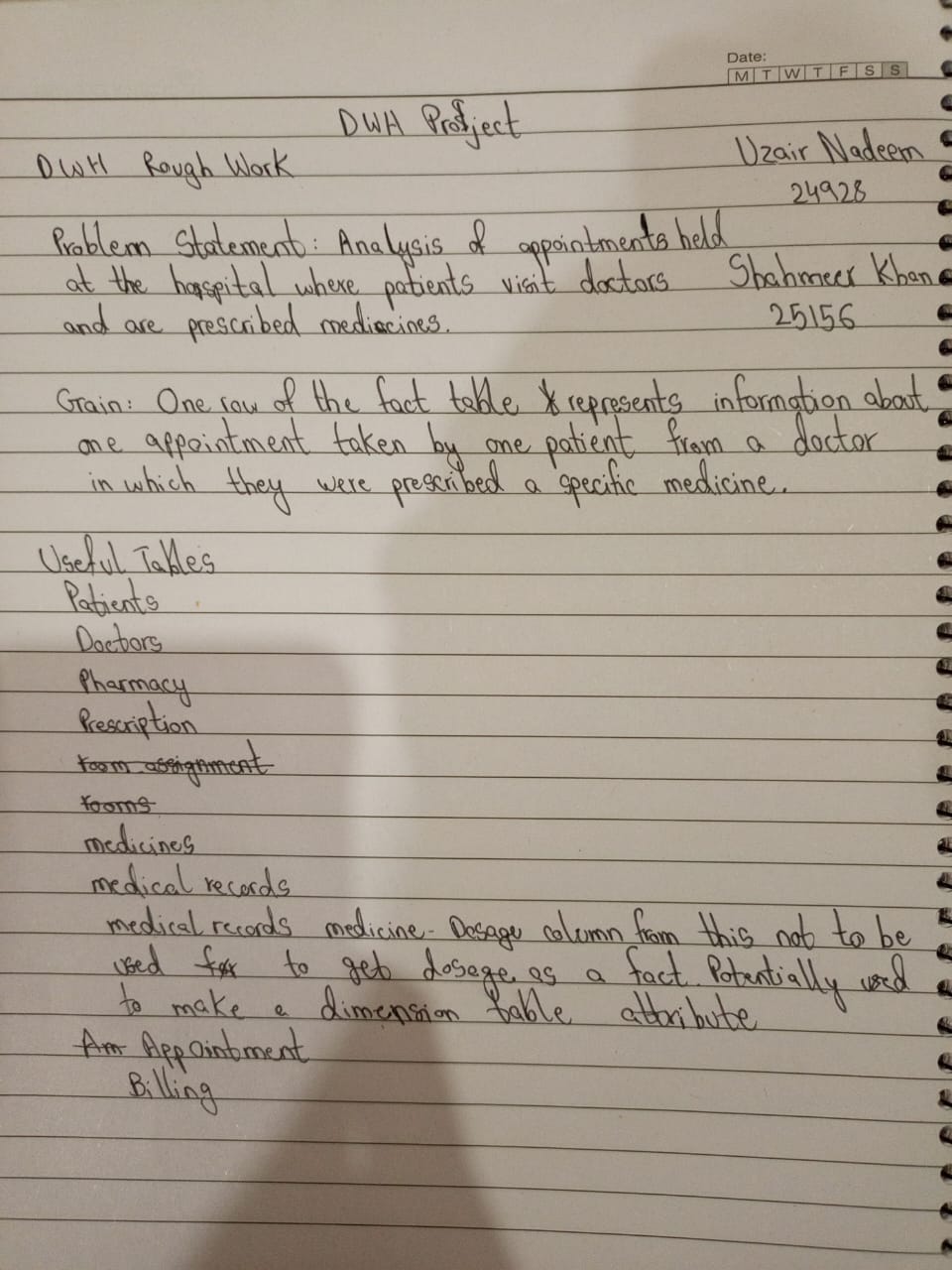
We opted for a healthcare dataset warehouse to define a data mart.

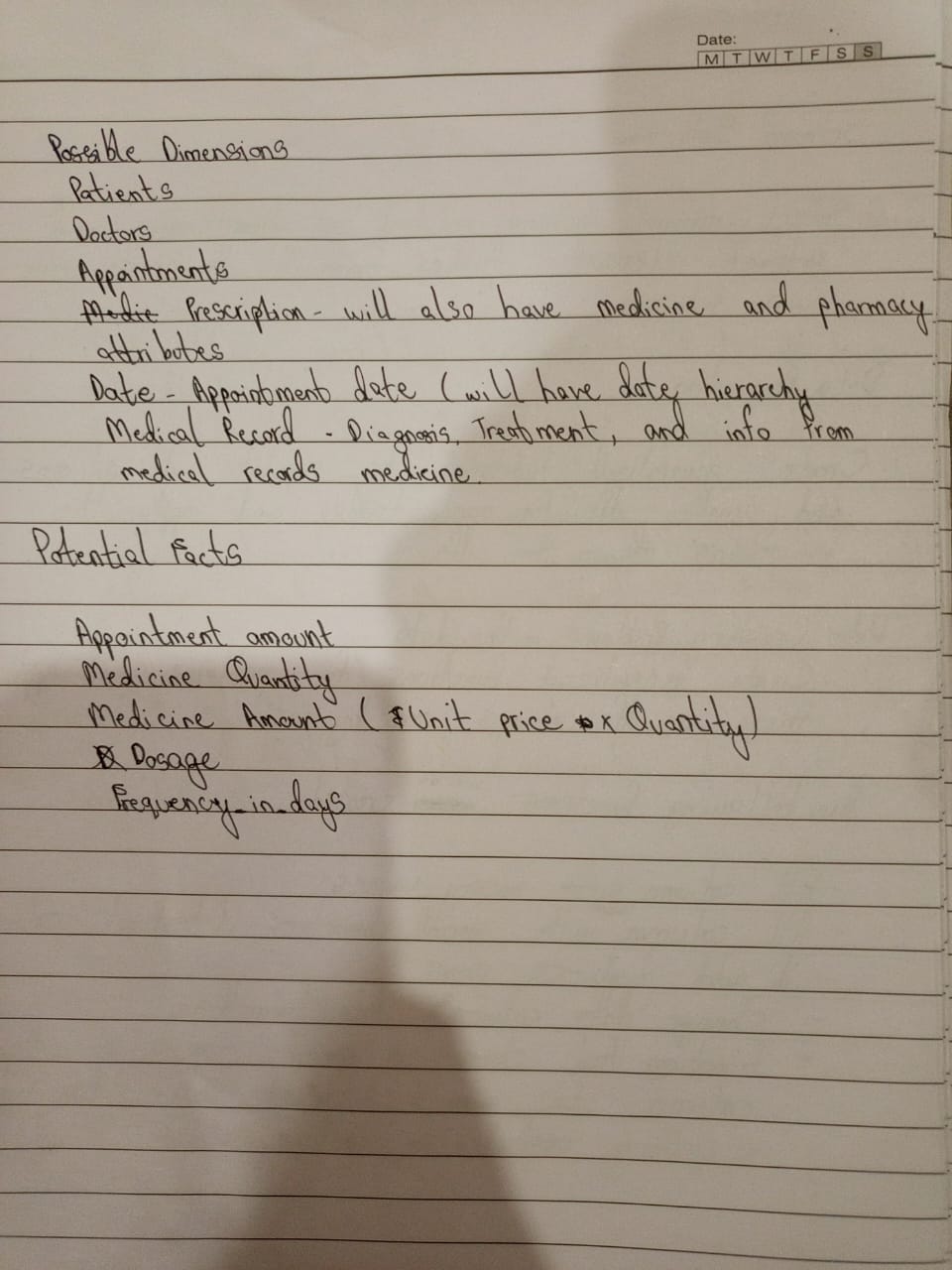
**Tech stack:**

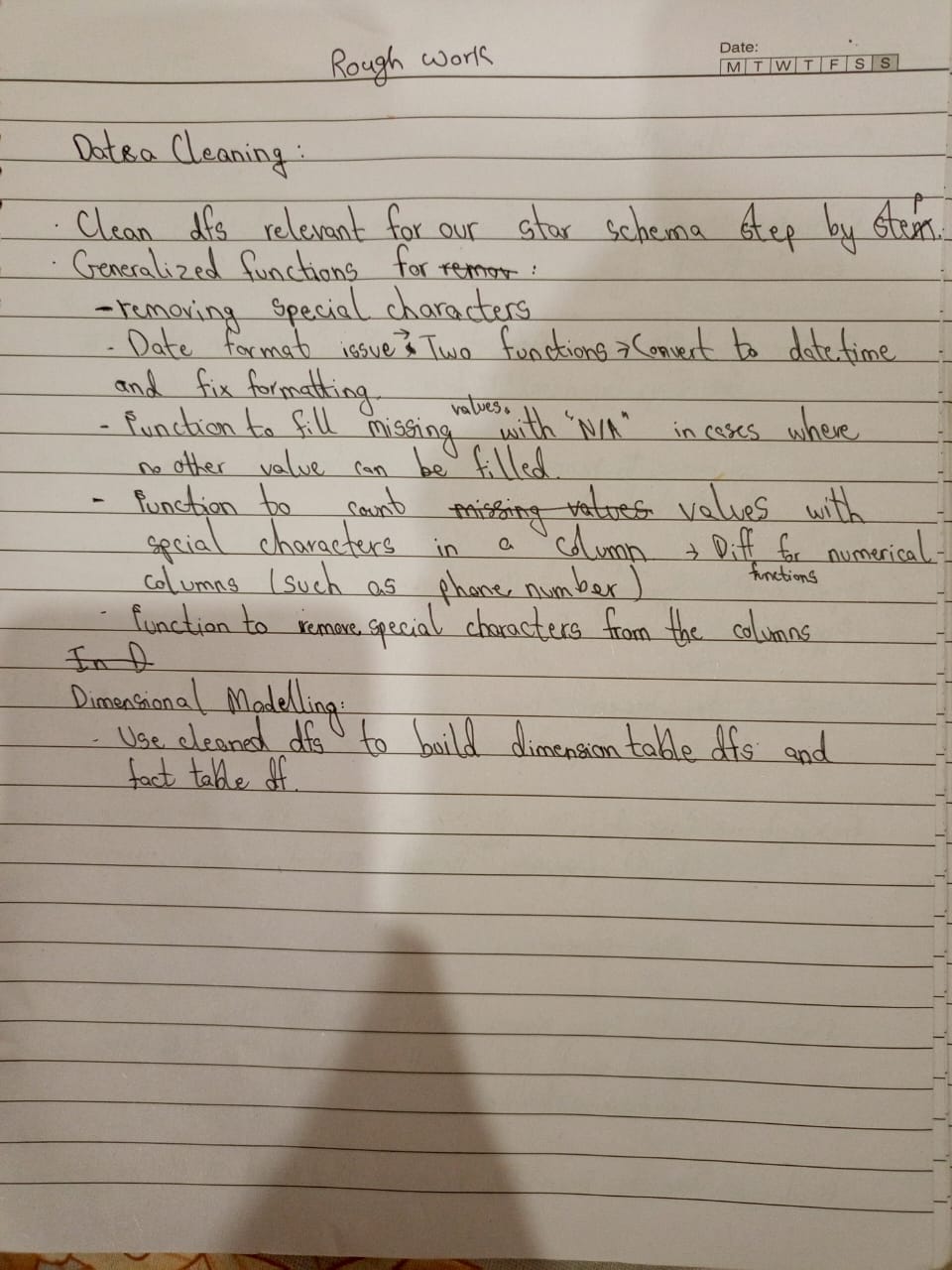
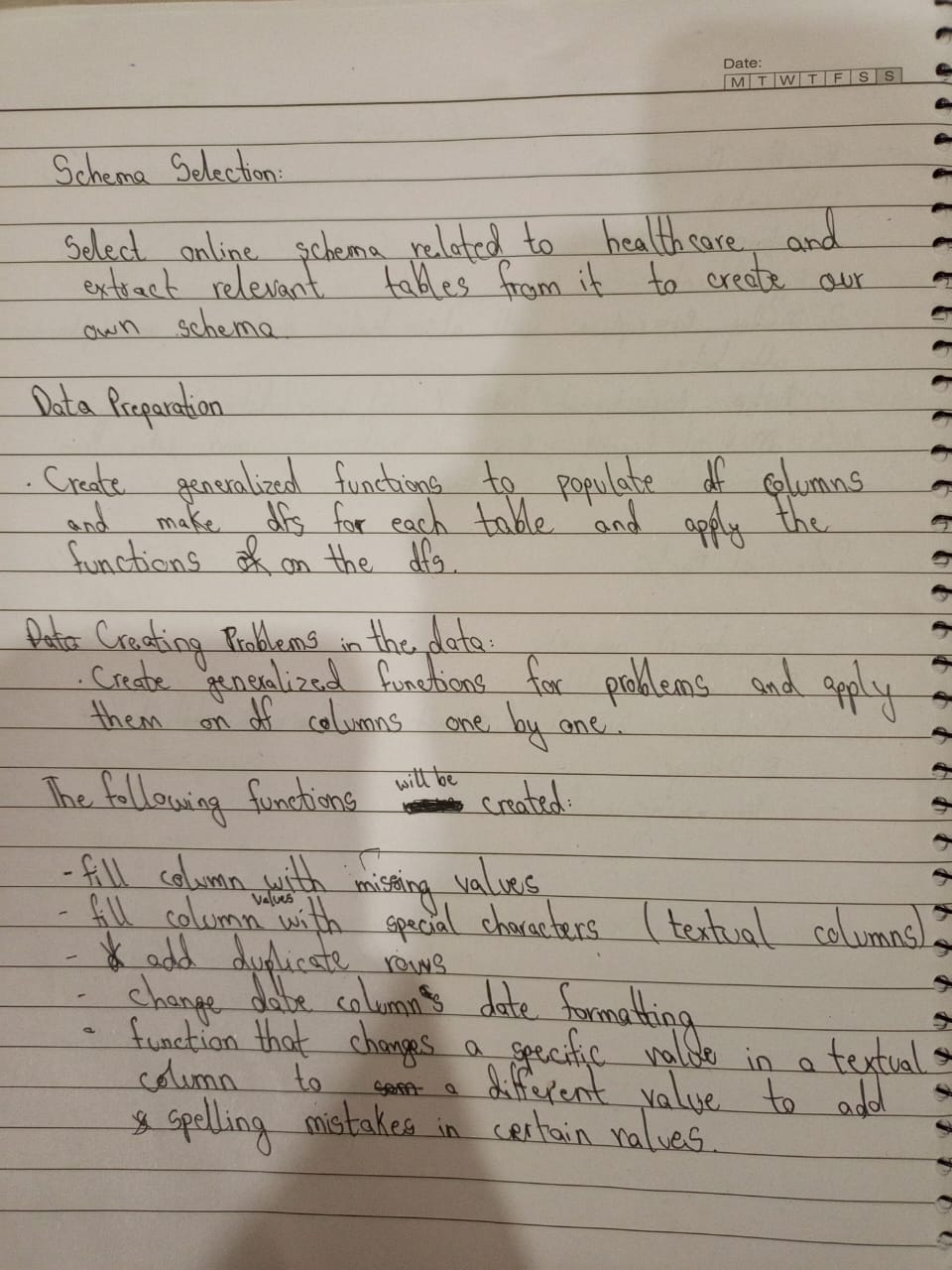
* **Airflow** for pipelining
* **Python** for scripts
* **Colab** for defining scripts via python notebooks
* **WSL Ubuntu** via Windows Powershell for managing Airflow
* **Snowflake** for DB and dashboarding

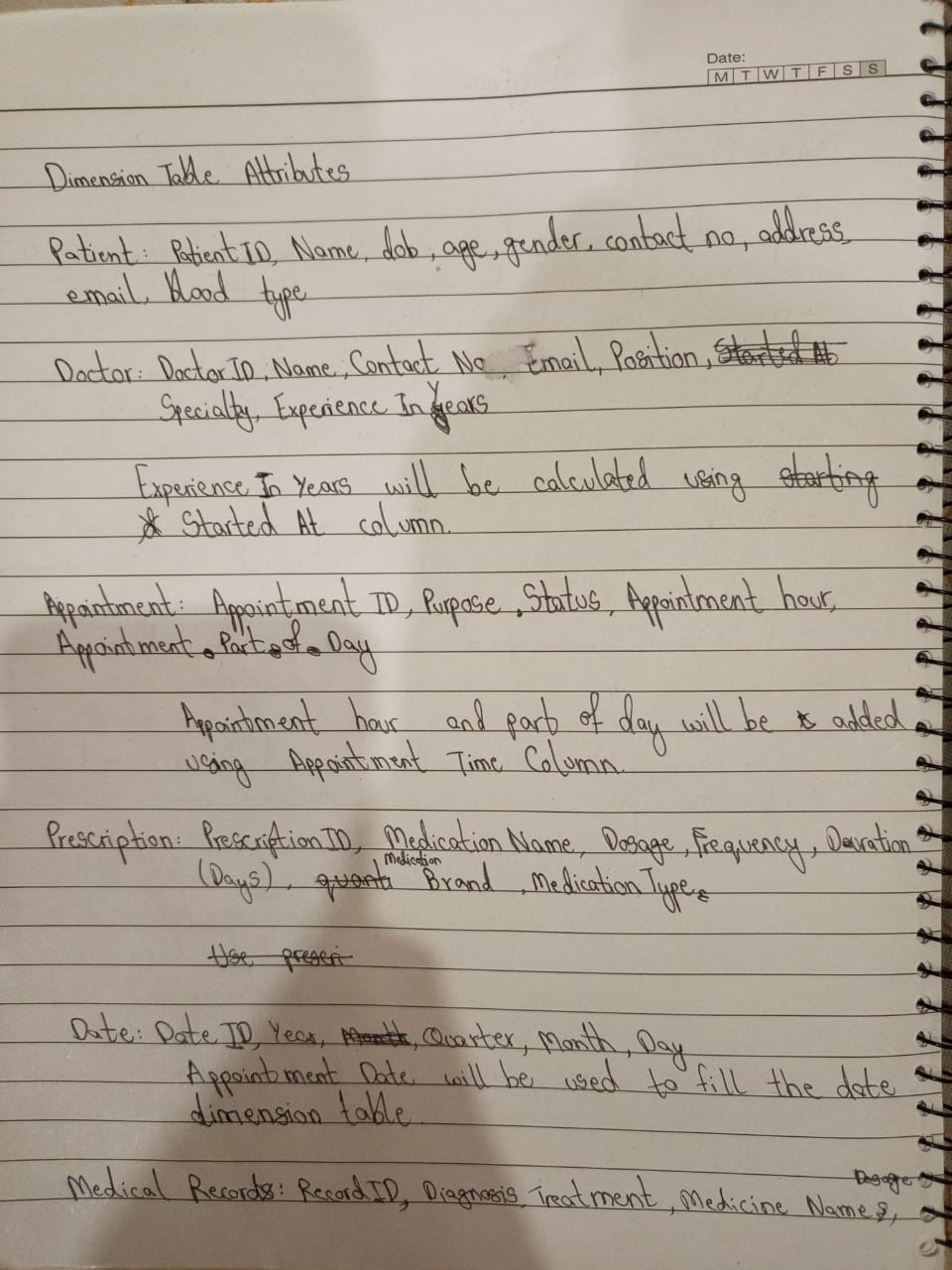
**Rough Work:**

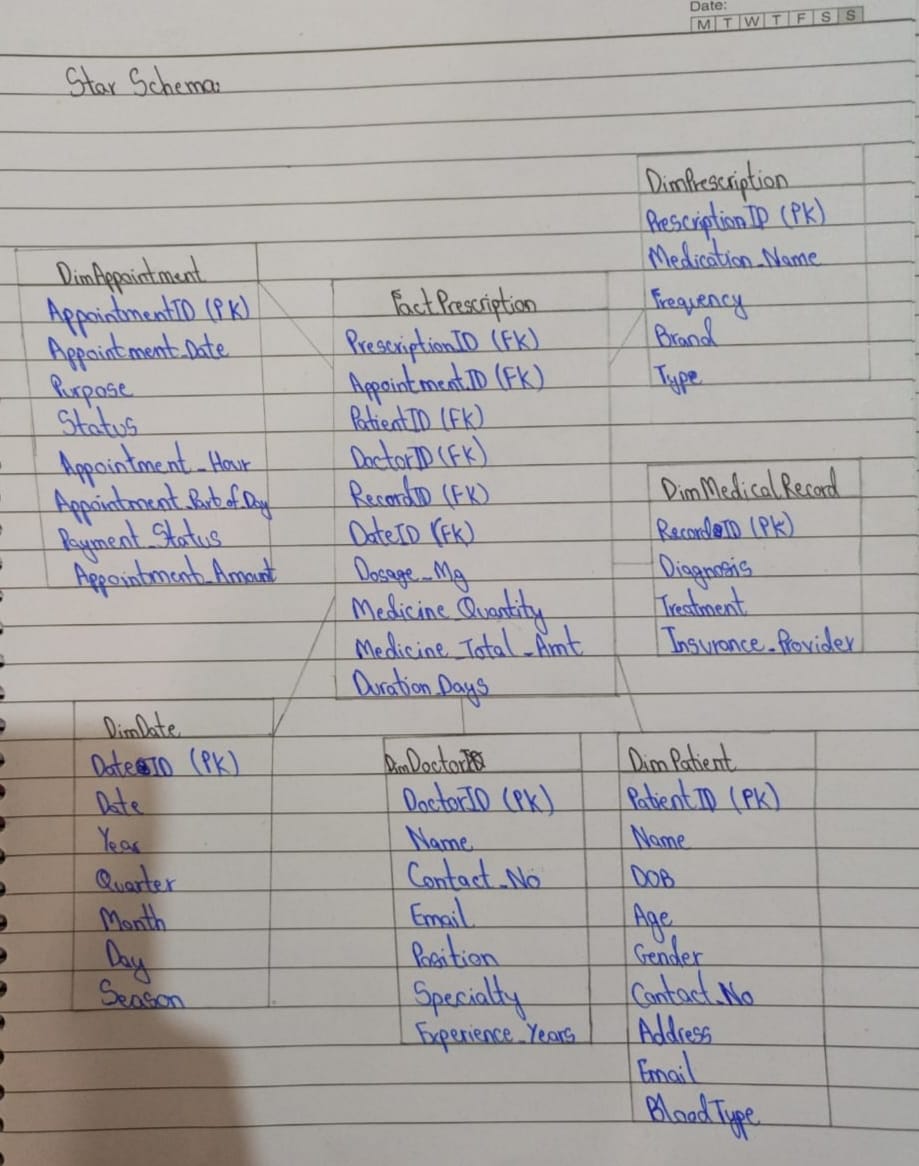
This section contains images of the rough work conducted when deciding our approach to the warehousing problem.



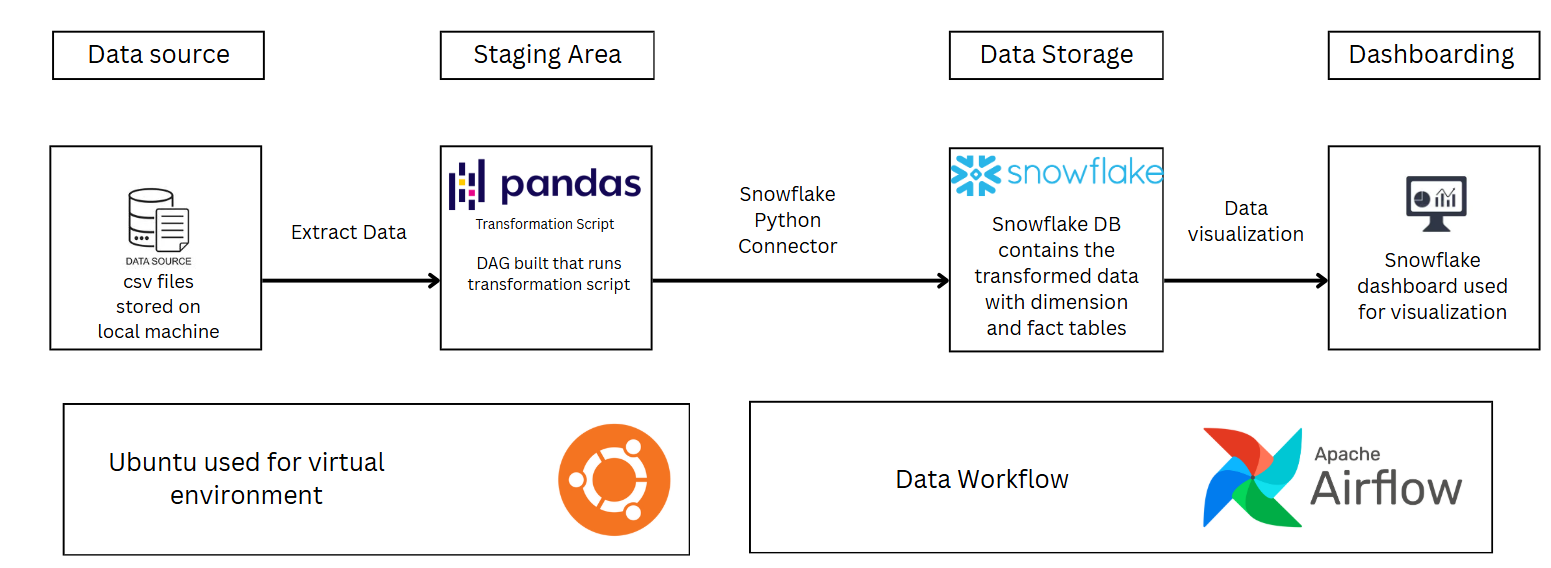








**System Architecture Diagram:**

****

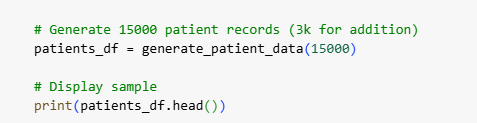
**Data Generation:**

Tables were populated via Faker library; Schema was defined using the ERD in the Schema URL above (not all tables were used).

Faker was used for majority columns; some were given arrays like blood types and faker was made to choose random values from the array to populate the relevant table.



Across tables a few were given bonus rows like the 3k here for patients, these will late be used as additional data to update the DB and dashboards.



In some cases, like appointments table, which depended on doctor and patient ids to generate appointments, these ids were taken from the relevant tables. This approach was implemented across all tables that had foreign key dependencies.



**Tables generated via Faker (not injected with data issues):**

Patients – 15k rows (3k extra)

Doctors – 800 rows

Appointments – 10k rows (3k extra)

Prescriptions – 15k rows (3k extra)

Medicines – 800 rows

Ambulances – 200 rows

Ambulance logs – 600 rows

Rooms – 200 rows

Medical records – 10k rows (3k extra)

Pharmacy – 15k rows (3k extra)

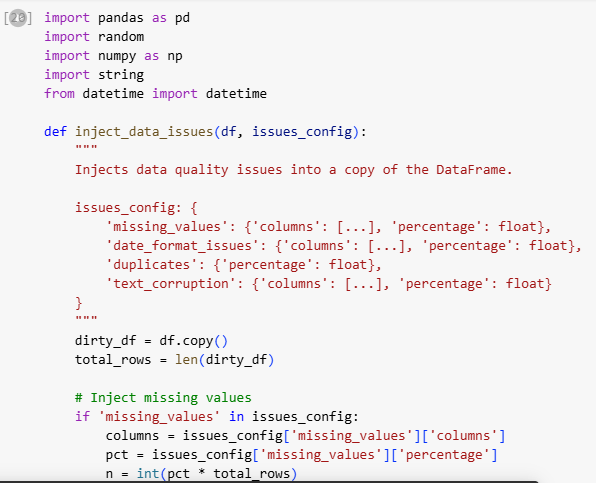
Room assignments – 7k rows

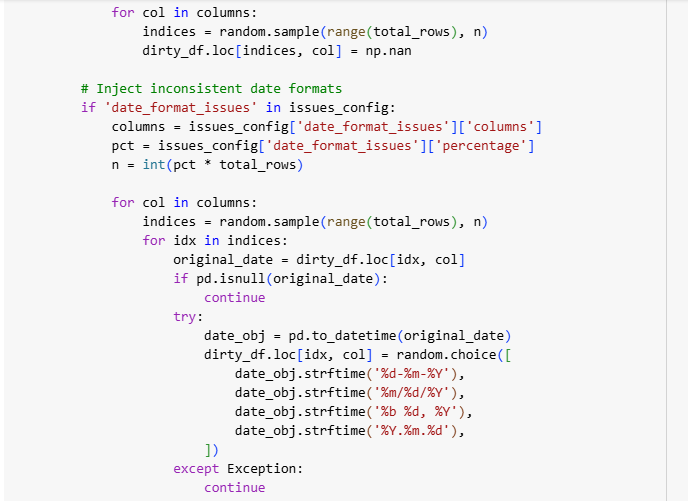
Medical record medicine – 15k rows (3k extra)

Billing – 10k rows (3k extra)

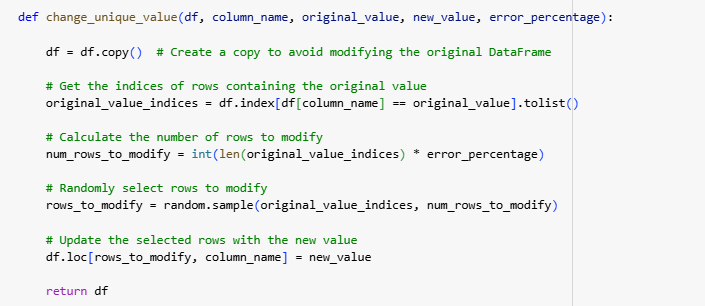
**Data issues functions:**

These functions were applied at random across all the data to generate data issues

****

****

****

****

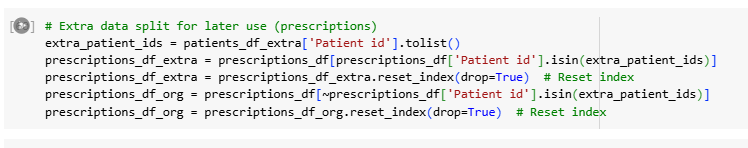
Functions applied as such

****

**Data splits for extra data:**

We did this logically where we split the patients table into 3k and 12k rows, then applied random data issues to both to ensure no duplicates exist among the splits, using the 3k patients, we then split the other tables in order, so appointments for those 3k, prescriptions for those 3k and so on, for extra data.

****

****

**CSV Downloads**

Converted the data-frames to CSV for original and extra data and downloaded them onto our local systems for use later on.

****

****

****

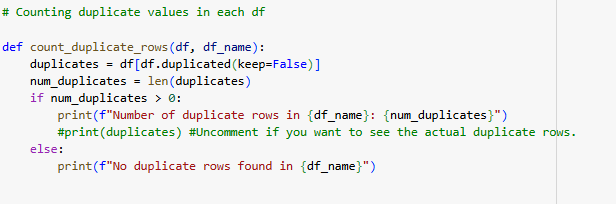
**Data transformation:**

After downloading the data, the csvs were read and then cleaned before dimensional modelling.

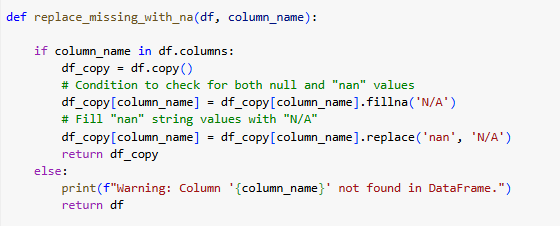
**Data Cleaning**

**Generic functions for data cleaning:**

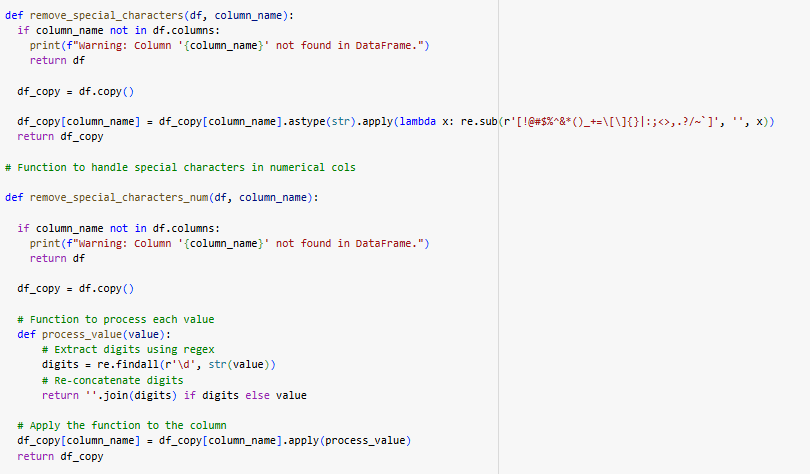
Function that counts duplicate values

****

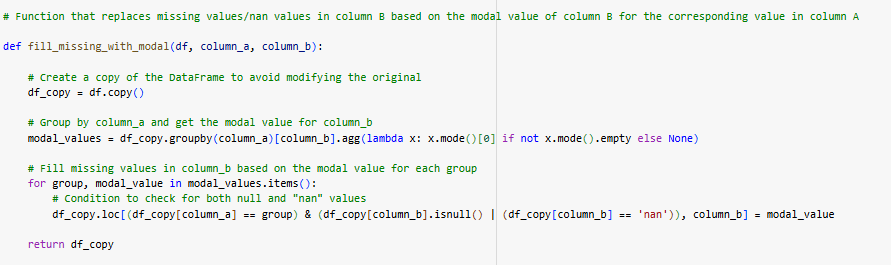
Function to replace missing vals with N/A, this was used a lot as logically, if a patient’s diagnoses was missing, u can’t use a modal value to fill it, similarly several columns with missing vals made no logical sense to fill except for with N/A.

****

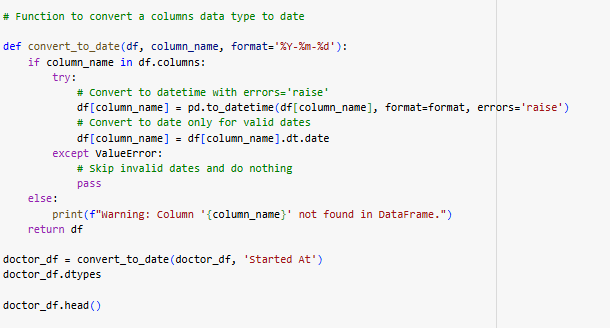
These two functions worked to remove special characters, they were used to deal with text corruption, the one with num in it was specifically used for contact numbers etc.

****

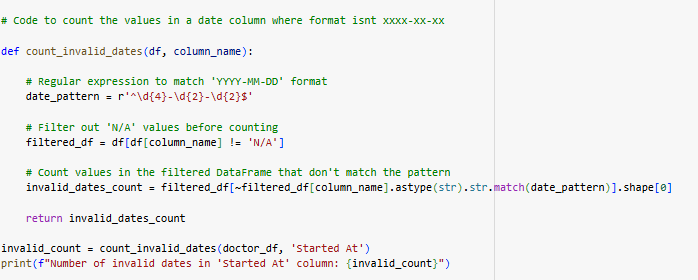
A more logical function that replaced column values based on modes in correlation to a different column, for example, used to get doctor specialty based on position

****

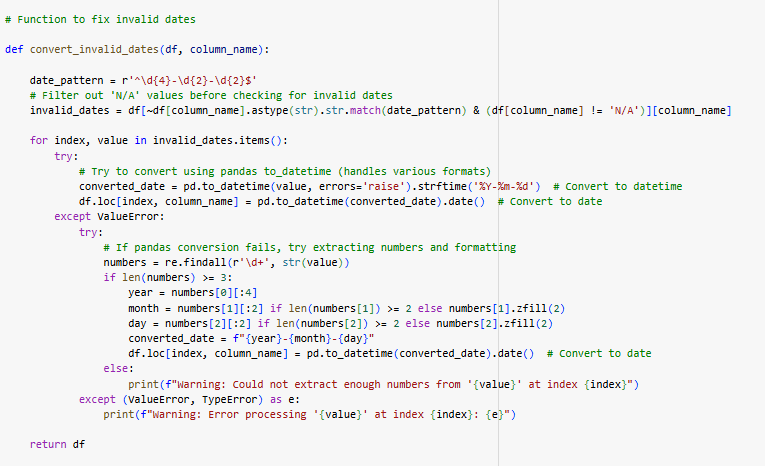
Function to handle date formatting issues, changes datatype to date before running the secondary function



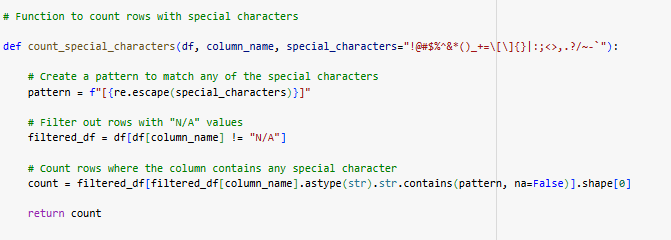
Function to count invalid dates i.e. dates that don’t follow the format yyyy-mm-dd



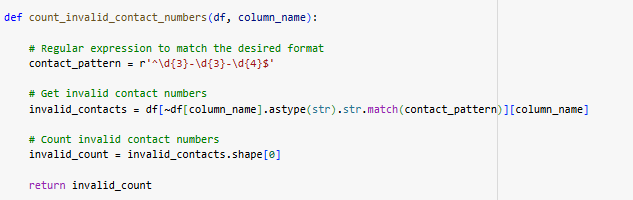
Function to fix invalid date formats

****

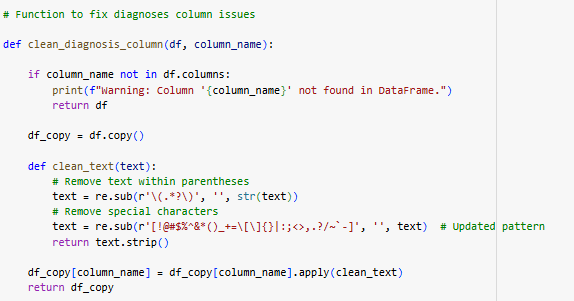
Function to count column values with special characters, used to look for text corruption



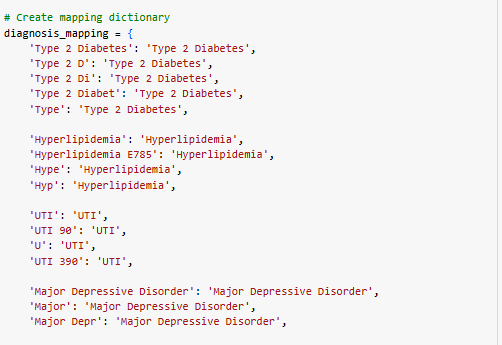
Function to count invalid contact details

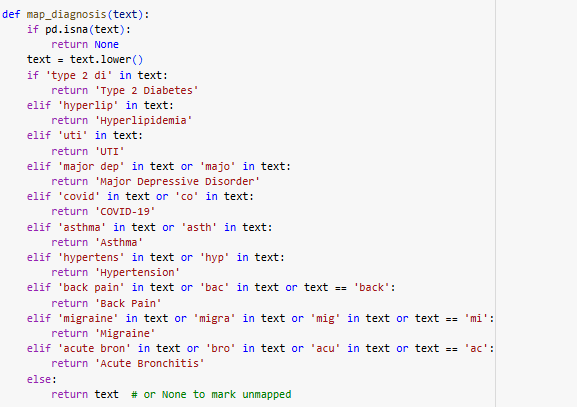


Partial function to clean diagnoses column in medical records

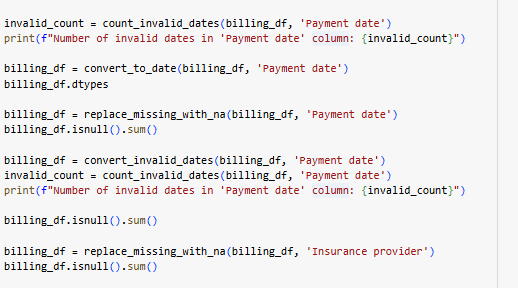


Diagnoses column had several issues with text corruption, needed column mapping to deal with some issues





Used the following functions as such to clean data, done for all dirty data tables based on the data issues in them



**Problem statement:**

A data mart that facilitates analysis of medicine prescriptions that were given to patients for a specific appointment

**Grain:**

One row of the fact table represents information about one prescription given to a patient in one appointment by one doctor for a specific medicine

**Dimensional modelling**

**Dimension tables:**

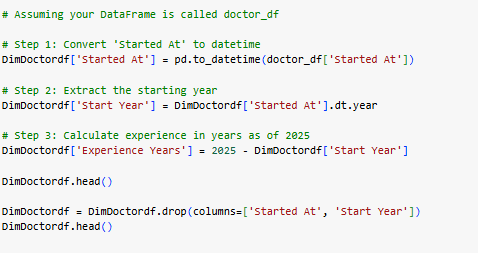
* DimDoctor
* DimPtient
* DimAppointment
* DimPrescription
* DimMedicalRecord
* DimDate

**Fact Table:**

* FactPrescription

Dimension modelling was done via several merges; some new columns were created using previously existing columns as well.

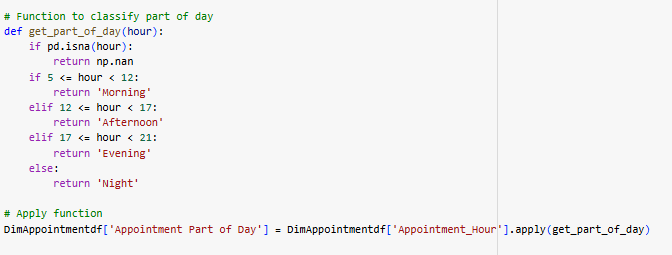
Creating experience years column using started at date for doctordf.



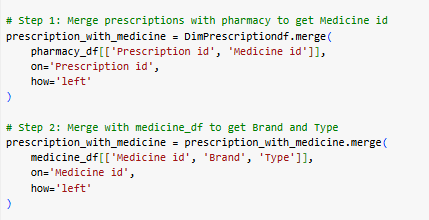
In some cases, entire cleaned tables were copied into relevant dimension table, columns were then dropped when not needed.



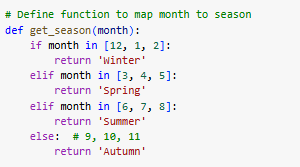
New column for DimAppointment



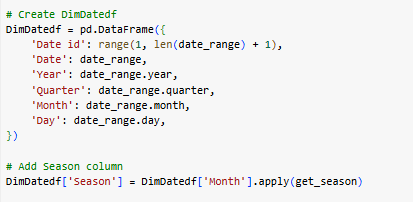
Merge functions to copy relevant columns using ids, done for all dimension table and fact table



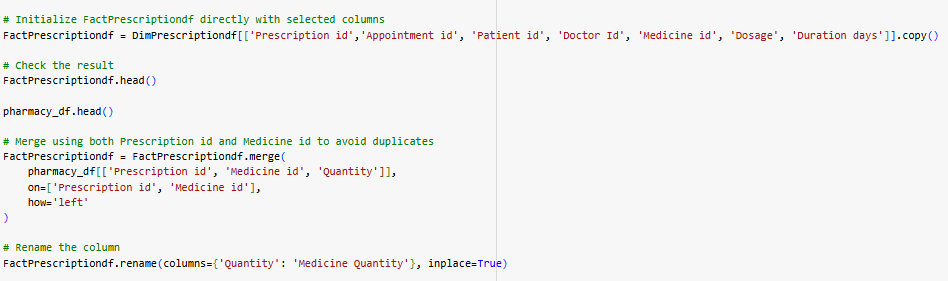
Defining a new season column for DimDate



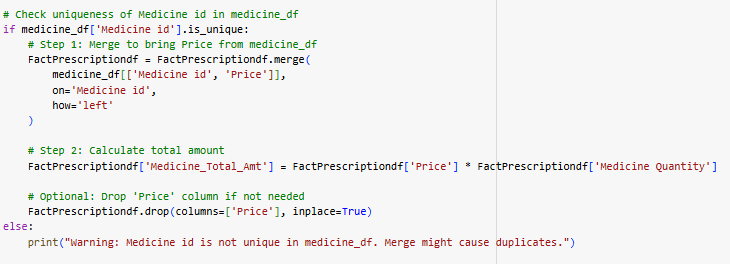
Defining a hierarchy for DimDate and adding season column



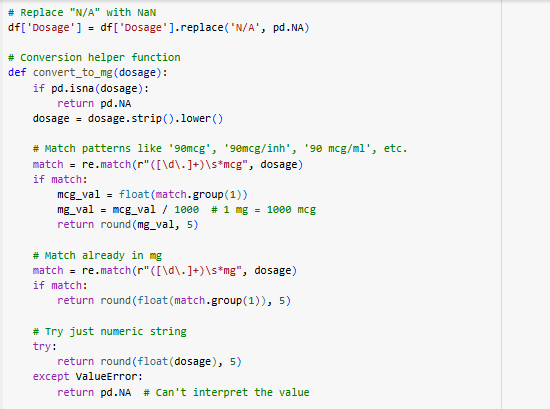
Defining fact table foreign keys and facts (quantity, dosage and duration in days)



Medicine amount as a fact using medicine price and quantity



Dosage as a fact, a function was needed to convert all dosage vals into mg



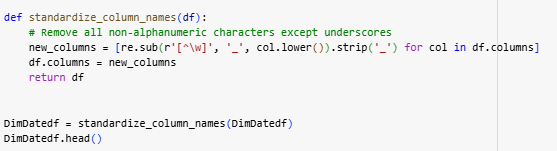
Converting dosage column to numerical for aggregation



Dropping irrelevant columns as such after dimensional modelling was completed, done for all dimension table and fact table



Function to standardize column names for the tables, lowercase and \_ instead of “ “



**Snowflake connection**

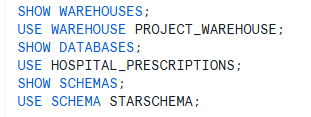
We connected to the snowflake DB with a connection string as such, we also used a pandas package write\_pandas to populate our snowflake database by directly uploading our data-frames to the relevant tables.

Overwrite was kept as true for doctors and date as they had to be reintegrated in the second run as well, appending would cause duplicates in snowflake.



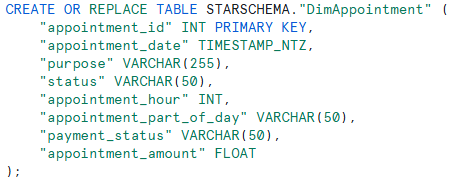
**Snowflake implementation**

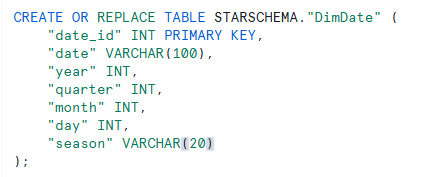
Defined a warehouse (medium sized) and a database to load the trasnsformed data into

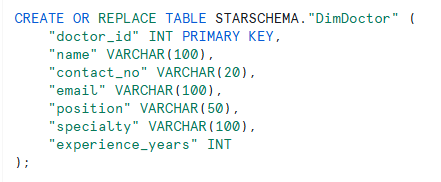
****

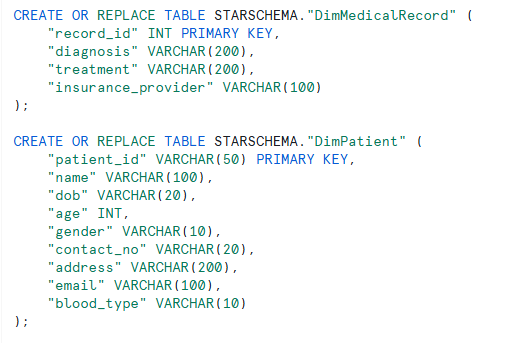
**Defining schemas for every table within snowflake**

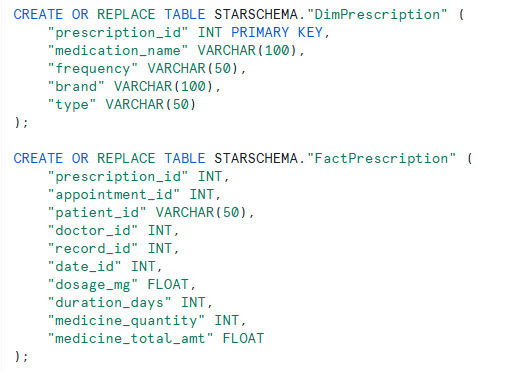
Column names should match the ones on the data-frames, otherwise errors are possible, defining these schemas ensures upload into snowflake is successful

****

****

****

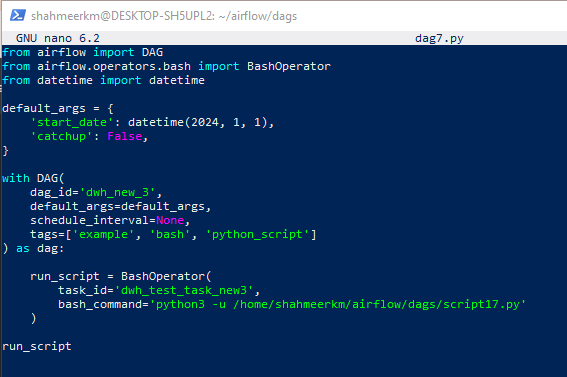
****

****

**DAG Implementation**

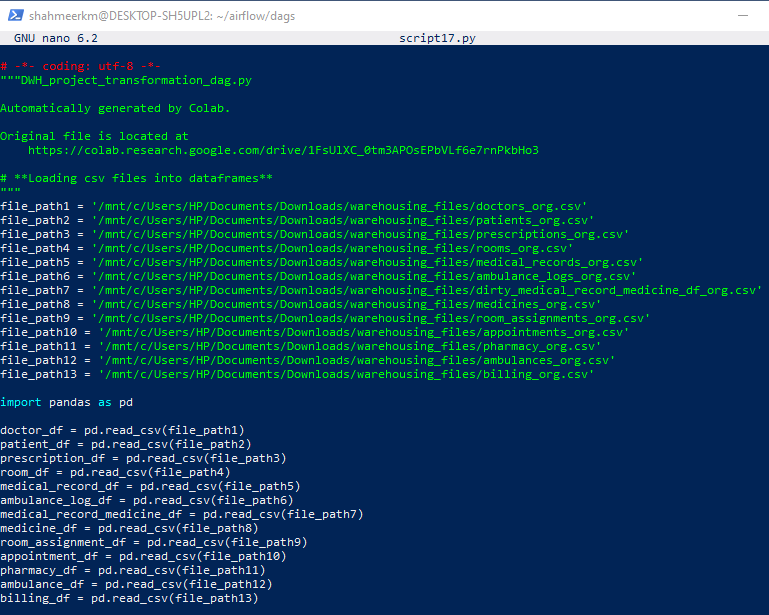
For the Dag implementation in airflow, we used the BashOperator library, this enabled us to directly execute scripts using the Dag.

Here we define the basic configuration for the Dag, the bas command executes your script as if it were a simple command line command, the script is stored in the same Dags folder for surety. The name of the dag on airflow will be dwh\_new\_3.

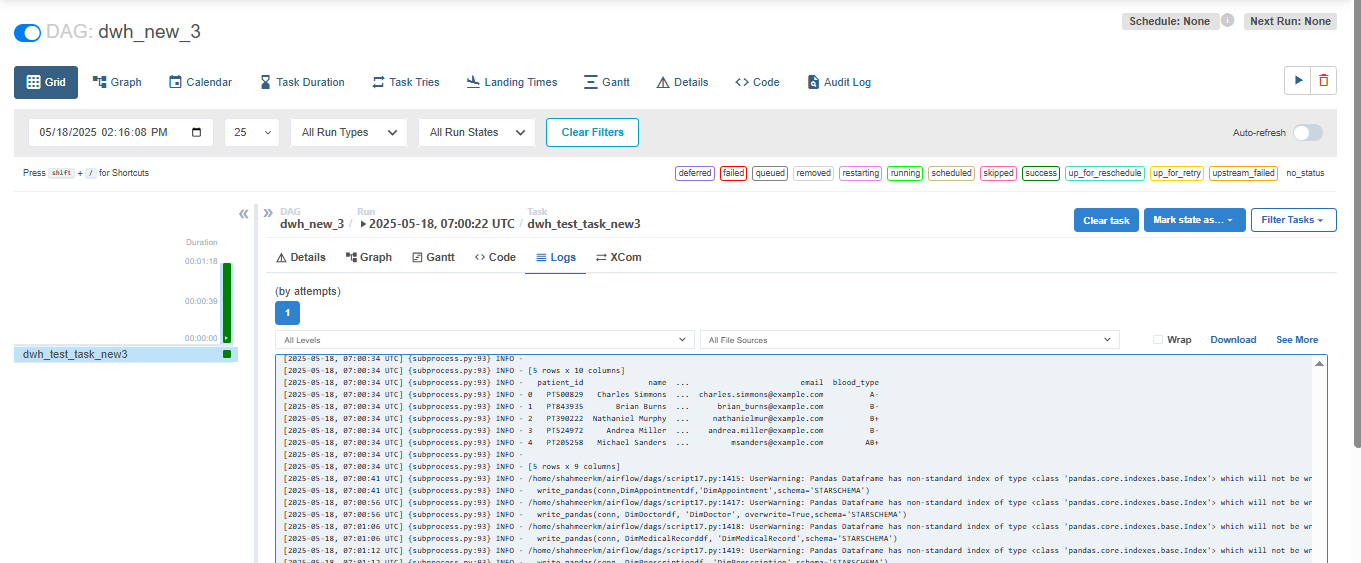


The script was as defined above in the transformation function, only difference being the file paths, as we’re using linux in a virtual env, we need to mount the files accordingly.

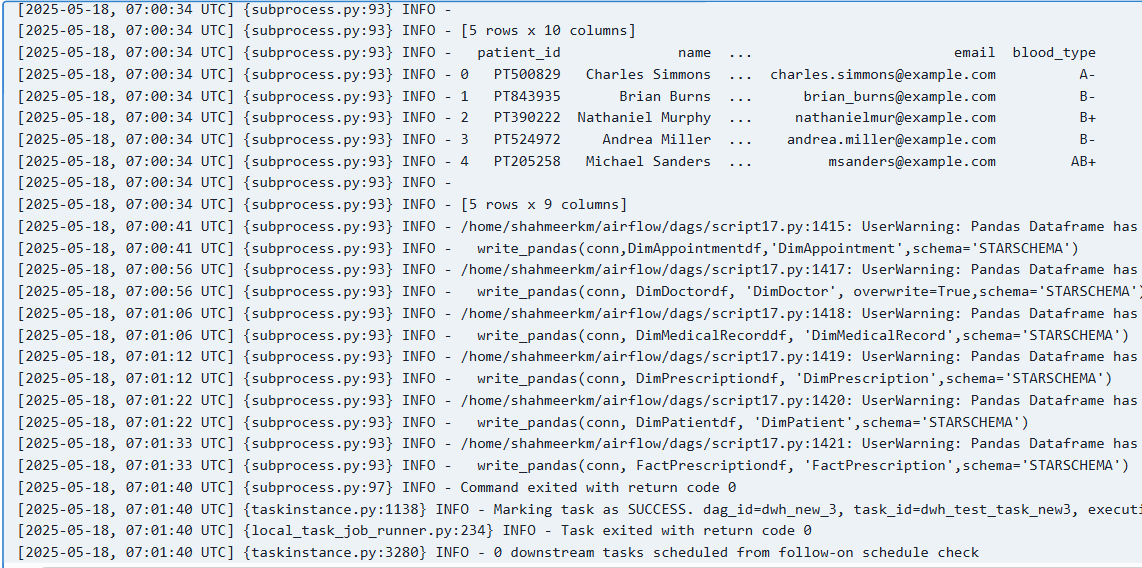
/mnt mounts the files into linux for use.



By accessing the airflow server on localhost we viewed and executed our Dag, here we see the successful execution of our Dag.

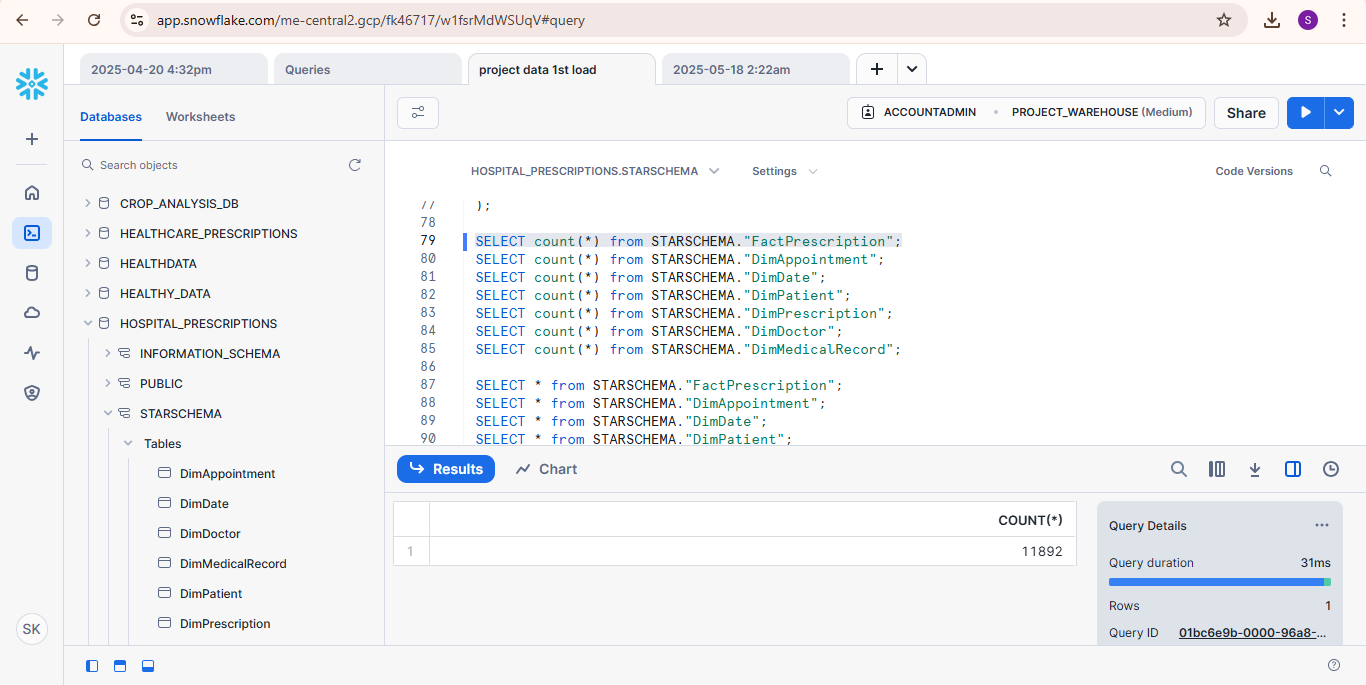


A closer look into the logs shows us that data was successfully loaded into snowflake. As we can see here, the Dag executed the entire python transformation script creating the star schema as requested (DimPatient printed in the logs below), it also uploaded the data-frames into relevant snowflake tables as needed via write\_pandas.

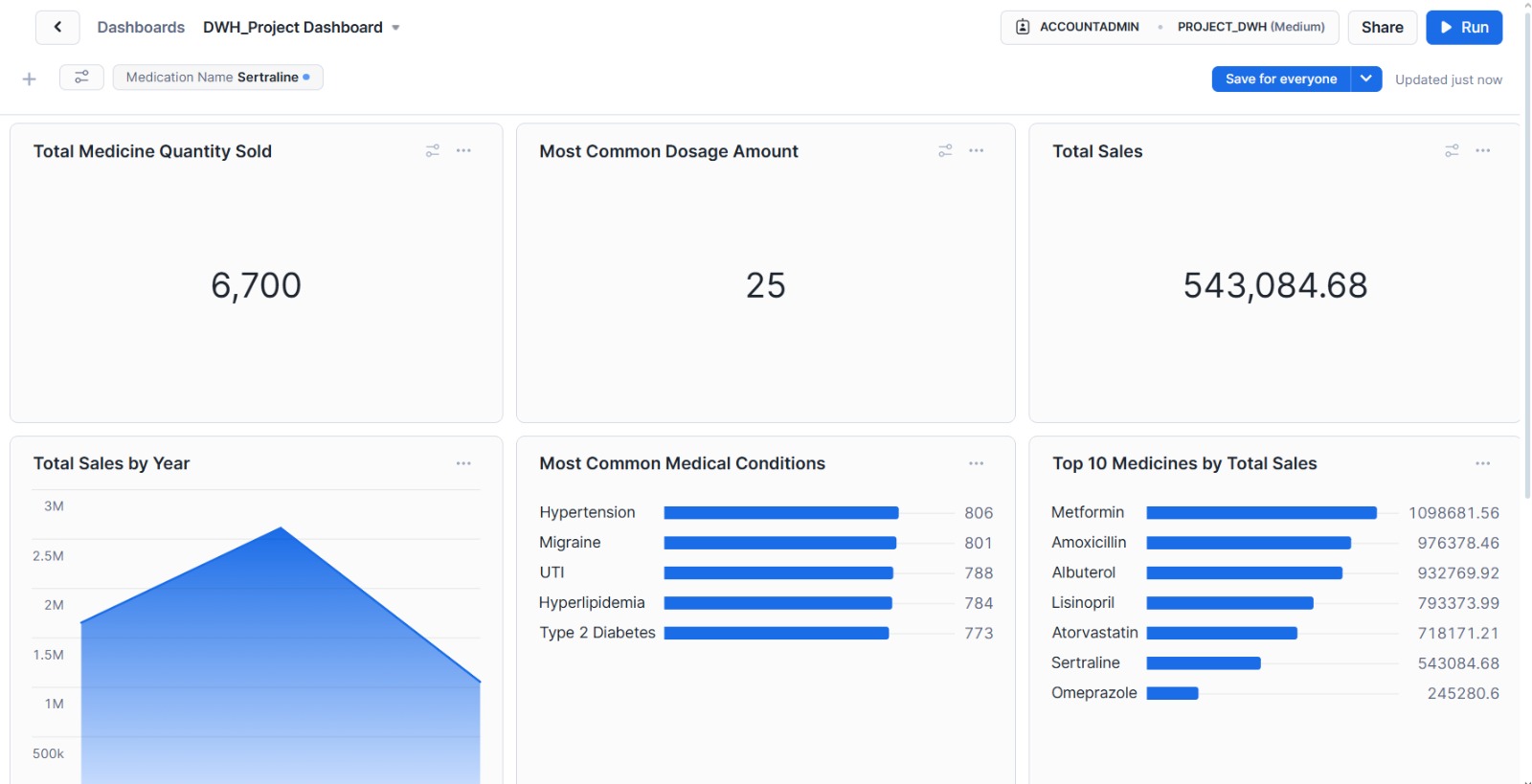


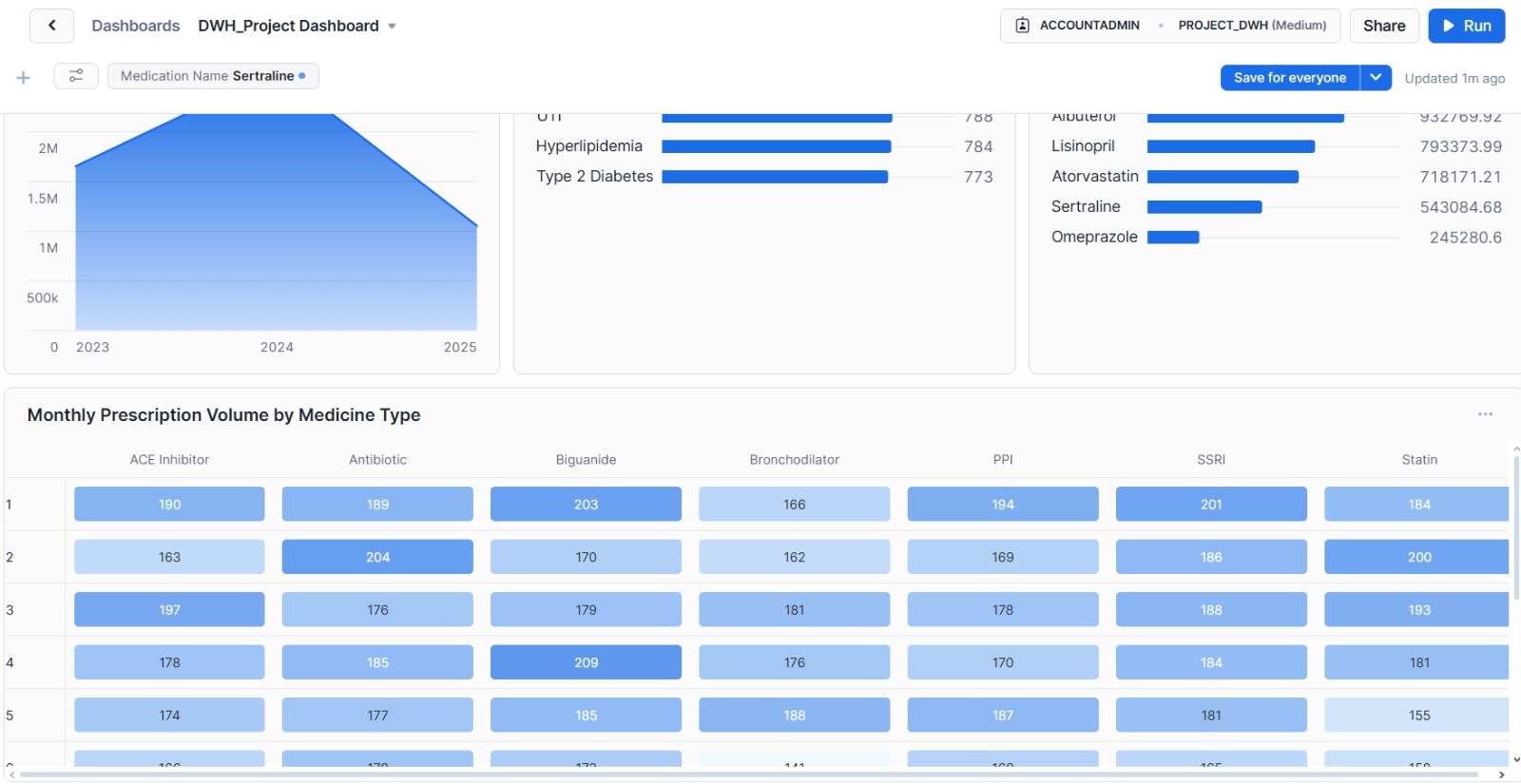
**Work on snowflake (Dashboarding)**

Here we can see that after the Dag’s execution, the snowflake tables were populated

****

**Dashboards:**

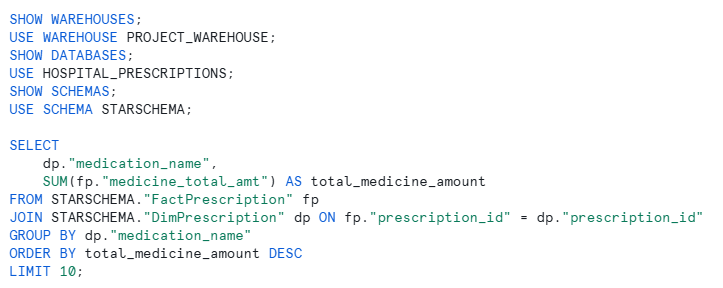
****

****

**Dashboard queries for charts:**

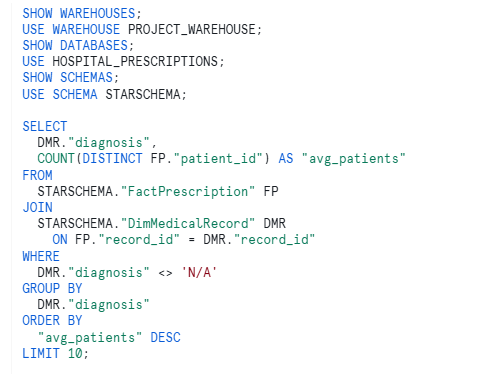
**Bar chart for Top Ten Medicines By Total Sales:**

Calculates total medication sales based on medication names

****

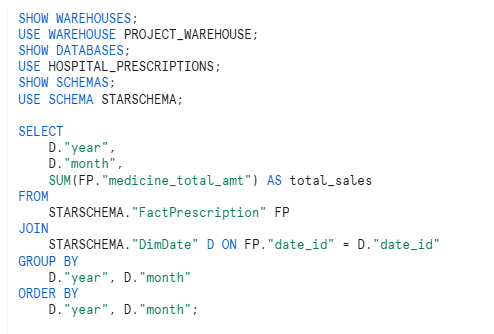
**Bar chart for Most Common Medical Conditions**

Calculates the top ten patient counts by diagnosis

****

**Line chart for total sales by year:**

Calculates total sales per year

****

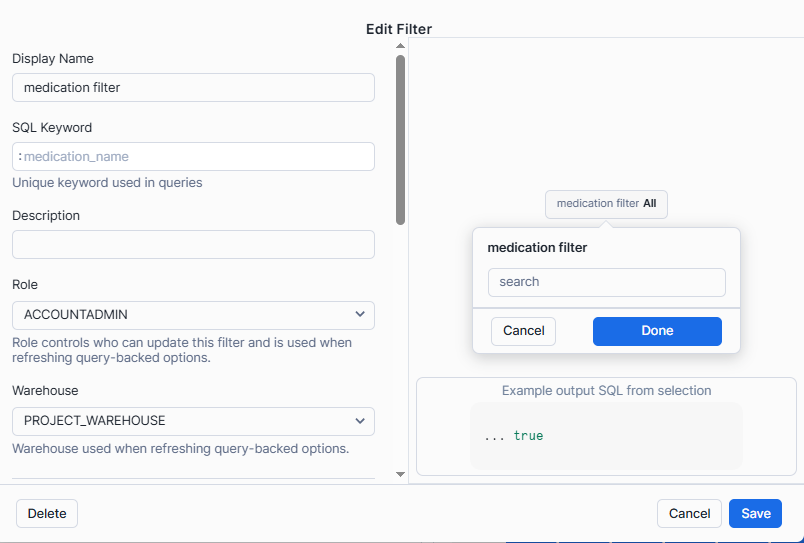
**Heatmap chart for Monthly Prescription Volume By Medicine Type:**

Calculates prescription counts per month for every medication

****

**Medication name filter for KPIs**

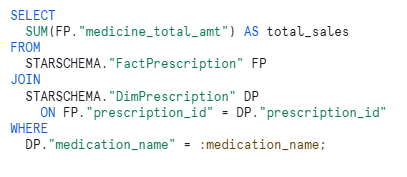
****

****

**Dashboard queries for KPIs using medication name filter:**

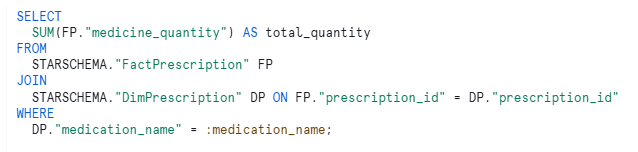
**Total sales:**

Calculates the total sales based on medication name

****

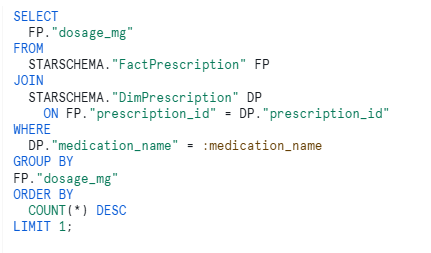
**Total Medicine Quantity Sold:**

Calculates the total quantity based on medication name

****

**Most Common Dosage Amount:**

Calculates the modal dosage based on medication name



**Problems faced**

This section highlights all the issues faced throughout this project

**Data generation:**

* Deciding on and generating proper functions for making the data dirty, several functions gave errors.
* Logically splitting the data-frames into original and extra tables, dependencies caused issues.
* We first ran our data issues functions on the main files and then split them, but this would cause problems after splits as duplicates mean the same record could exist in both splits, so instead injected data issues after split.

**Data transformation:**

* In certain tables the data issue functions really ruined our data so defining the proper generalized functions to work across all tables became a challenge.
* Came across several new issues with formatting that we hadn’t even injected into our data so had to deal with those as well.
* Column mapping with snowflake caused a lot of hurdles so had to redefine several column names to match the snowflake system as () are not allowed in snowflake columns but were used in several columns of ours.

**Dimensional modelling:**

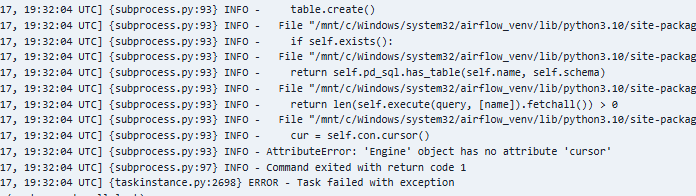
* Had to spend a lot of time deciding the grain, the dimension tables, figuring out facts etc as we went through multiple possible data marts to land on the prescription data mart in the end.

**Dags:**

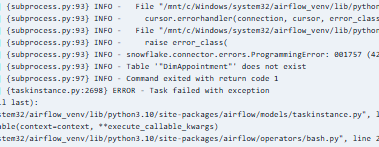
* Initially used PythonOperator, but for that we had to wrap the whole script in a function which was too inconvenient and caused several issues.
* We then went for BashOperator which was also really difficult to get a hold of and understand its implementation and execute it.

**Snowflake Connection:**

* Initially we opted for SQLAlchemy to push our data into snowflake however that kept giving us cursor errors as such

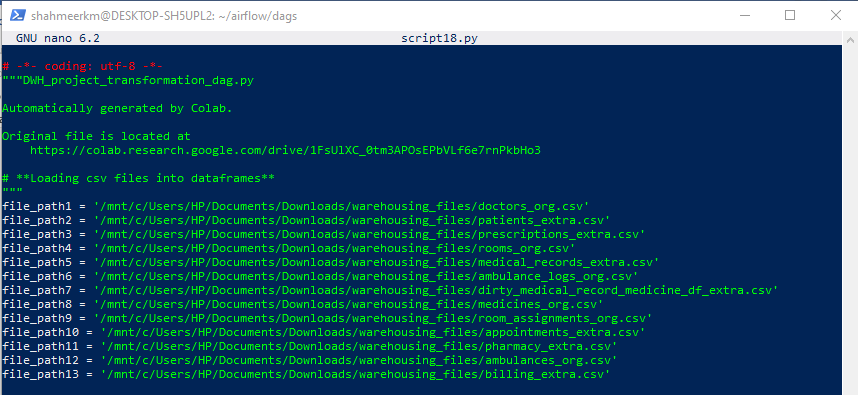


* We later went for pandas\_write as an alternative to SQLAclhemy, that however gave other new errors due to column names as mentioned before.
* Table names also caused issues due to case sensitivity, snowflake expects table names to either be in all caps or in “”, due to this it wasn’t recognizing our table names from the dfs as such, fixed this by having “” for every table and column name in our snowflake schema design.



**DB Update Task:**

We mentioned before that we prepared separate data for uploading, we simply ran this through the same pipeline as before using the same Dag and the same script with a minor change, updated file paths to this time take the new data instead. Here, the file paths now represent the extra data as well as some of the original data which will be needed for dimensional modelling of the extra data, this however has been ensured to not be duplicated via overwrite=true as mentioned before.



Once this data was uploaded to the same DB, the dashboard automatically updated as shown in the video tutorial.

Issues faced here were that we initially didn’t understand how to prevent duplication of data however that was dealt with via the overwrite command.