|  |  |
| --- | --- |
| Name: SRUTHI V. S.  Course: Data Science | Batch: Weekend Batch  Mini Project 2 |
| 📄 DS Project Documentation: Vaccination Data Analysis | |

**Project Overview**

This project analyzes global vaccination data to identify trends, assess coverage, and evaluate the effectiveness of vaccination campaigns. It follows a structured data pipeline: data cleaning, transformation, storage in a normalized SQL database, integration with Power BI, and generation of insightful interactive dashboards.

**✅ Step-by-Step Progress Review**

**🔹 Data Cleaning**

| **Task** | **Status** | **Details** |
| --- | --- | --- |
| **Handle Missing Data** | ✅ Done | We discussed and removed or imputed missing values in key columns during the Pandas cleaning steps. |
| **Normalize Units** | ✅ Done | Coverage and incidence were checked and retained in consistent formats (e.g., percentages and rates). |
| **Date Consistency** | ✅ Done | The year values were extracted, standardized, and used to populate the years dimension table. |

**🔹 SQL Database Setup**

| **Task** | **Status** | **Details** |
| --- | --- | --- |
| **Create Tables** | ✅ Done | All required tables (coverage, incidence\_rate, reported\_cases, vaccine\_introduction, vaccine\_schedule) were created. |
| **Normalize Data** | ✅ Done | Separate dimension tables were created for countries, vaccines, diseases, and years to remove redundancy. |
| **Data Integrity** | ✅ Done | Primary keys and foreign keys were implemented to ensure referential integrity across all fact tables. |

**✅ Summary**

| **Component** | **Status** | **Notes** |
| --- | --- | --- |
| **Data Cleaning** | ✅ Complete | Handled missing data, normalized units, standardized dates |
| **SQL Table Creation** | ✅ Complete | Created and normalized all necessary tables |
| **Referential Integrity** | ✅ Complete | Foreign key constraints implemented |

**Step 1: # Import Necessary libraries:**

import pandas as pd

import numpy as np

from sqlalchemy import create\_engine

**Step 2: # Load the Excel data sheets**

coverage\_df = pd.read\_excel('coverage-data.xlsx', sheet\_name='Coverage Data')

incidence\_df = pd.read\_excel('incidence-rate-data.xlsx', sheet\_name='Incident Rate')

cases\_df = pd.read\_excel('reported-cases-data.xlsx', sheet\_name='Reported Cases')

intro\_df = pd.read\_excel('vaccine-introduction-data.xlsx', sheet\_name='Vaccine Introduction')

schedule\_df = pd.read\_excel('vaccine-schedule-data.xlsx', sheet\_name='Vaccine Schedule Date')

!pip install pandas openpyxl sqlalchemy pymysql

**Data Cleaning Process**

**Handling Missing Data**

During the initial data exploration in Google Colab, I loaded five Excel tables covering:

* Coverage Data
* Incidence Rate
* Reported Cases
* Vaccine Introduction
* Vaccine Schedule

Each dataset was assessed for missing values. Rows with critical nulls (e.g., country code, year, or vaccine name) were dropped. In non-critical fields (like comments or schedule notes), missing values were imputed with placeholders or ignored.

**Step 3: Data Cleaning:**

**# Cleaning Data**

**# a. Cleaning Coverage Data**

coverage\_df.columns = coverage\_df.columns.str.strip().str.lower().str.replace(' ', '\_')

coverage\_df['coverage'] = pd.to\_numeric(coverage\_df['coverage'], errors='coerce')

coverage\_df['coverage'] = coverage\_df['coverage'].apply(lambda x: x\*100 if pd.notnull(x) and x < 1 else x)

coverage\_df['coverage'] = coverage\_df['coverage'].clip(upper=100)

coverage\_df['year'] = pd.to\_numeric(coverage\_df['year'], errors='coerce')

coverage\_df.dropna(subset=['coverage', 'target\_number', 'doses'], inplace=True)

**# b. Cleaning Incident Rate Data**

incidence\_df.columns = incidence\_df.columns.str.strip().str.lower().str.replace(' ', '\_')

incidence\_df['incidence\_rate'] = pd.to\_numeric(incidence\_df['incidence\_rate'], errors='coerce')

incidence\_df['year'] = pd.to\_numeric(incidence\_df['year'], errors='coerce')

incidence\_df.dropna(subset=['incidence\_rate'], inplace=True)

**# c. Cleaning Reported Cases Data**

cases\_df.columns = cases\_df.columns.str.strip().str.lower().str.replace(' ', '\_')

cases\_df['cases'] = pd.to\_numeric(cases\_df['cases'], errors='coerce').fillna(0).astype(int)

cases\_df['year'] = pd.to\_numeric(cases\_df['year'], errors='coerce')

**# d. Cleaning Vaccine Introduction Data**

intro\_df.columns = intro\_df.columns.str.strip().str.lower().str.replace(' ', '\_')

intro\_df.rename(columns={'iso\_3\_code': 'code'}, inplace=True)

intro\_df['year'] = pd.to\_numeric(intro\_df['year'], errors='coerce')

intro\_df['intro'] = intro\_df['intro'].apply(lambda x: 1 if str(x).lower() == 'yes' else 0)

**# e. Cleaning Vaccine Schedule Data**

schedule\_df.columns = schedule\_df.columns.str.strip().str.lower().str.replace(' ', '\_')

schedule\_df.rename(columns={'iso\_3\_code': 'code'}, inplace=True)

schedule\_df['year'] = pd.to\_numeric(schedule\_df['year'], errors='coerce')

**Normalization and Consistency**

**Data was standardized to ensure consistency:**

* Coverage percentages were validated to be within 0-100.
* Date fields were converted to integer years.
* Country names and codes were aligned across all tables.
* Field names were normalized for SQL compatibility.

**Data Types and Constraints**

* Proper data types were defined: INTEGER for years, REAL for rates, TEXT for descriptions
* Primary and foreign keys enforced relationships
* Composite keys were created where needed (e.g., country\_id + year + vaccine\_id)

**Step 4: # Creating MySQL Compatible Engine**

engine = create\_engine('sqlite:///vaccination.db')  # For MySQL, replace with mysql+pymysql://user:pwd@host/db

**Step 5: # Writing Cleaned data to SQL**

coverage\_df.to\_sql('coverage-data', engine, index=False, if\_exists='replace')

incidence\_df.to\_sql('incidence-rate-data', engine, index=False, if\_exists='replace')

cases\_df.to\_sql('reported-cases-data', engine, index=False, if\_exists='replace')

intro\_df.to\_sql('vaccine-introduction-data', engine, index=False, if\_exists='replace')

schedule\_df.to\_sql('vaccine-schedule-data', engine, index=False, if\_exists='replace')

**Step 6: # Downloading SQLite DB for Power BI**

from google.colab import files

files.download('vaccination.db')

**# In the above implementation (using SQLite via SQLAlchemy in Colab):**

**# i. We cleaned all 5 datasets in Python.**

**# ii. We saved them into flat SQL tables**

**# However, these are raw tables, not relational or normalized — they are denormalized flat tables with repeated values for countries, vaccines, diseases, etc.**

**# Now, we are going to Generate the full normalized SQL schema, then create and populate it using the cleaned data in Colab**

**SQL Database Quality**

**Structure and Normalization**

A relational SQLite database was created using SQLAlchemy within Colab. The schema was normalized with these key tables:

* countries (code, Name)
* years
* vaccines (code, description)
* diseases (code, description)
* coverage\_data
* incidence\_rate
* reported\_cases
* vaccine\_introduction
* vaccine\_schedule

Each fact table used foreign keys to reference dimension tables, ensuring referential integrity and eliminating redundancy.

**from sqlalchemy import create\_engine, text**

**# 1. Create SQLite database**

**engine = create\_engine('sqlite:///normalized\_vaccination.db')**

**conn = engine.connect()**

**Step 7:**

**# Create Dimension Tables**

**# Country table**

conn.execute(text("""

CREATE TABLE IF NOT EXISTS countries (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    code TEXT UNIQUE,

    name TEXT,

    who\_region TEXT

)

"""))

**# Vaccines table**

conn.execute(text("""

CREATE TABLE IF NOT EXISTS vaccines (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    code TEXT UNIQUE,

    description TEXT

)

"""))

**# Diseases table**

conn.execute(text("""

CREATE TABLE IF NOT EXISTS diseases (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    code TEXT UNIQUE,

    description TEXT

)

"""))

**# Denominators table**

conn.execute(text("""

CREATE TABLE IF NOT EXISTS years (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    year INTEGER UNIQUE

)

"""))

years\_df = pd.DataFrame({

    'year': pd.concat([

        coverage\_df['year'],

        incidence\_df['year'],

        cases\_df['year'],

        intro\_df['year'],

        schedule\_df['year']

    ]).dropna().unique()

}).astype(int)

years\_df.to\_sql('years', conn, if\_exists='append', index=False)

**Step 8: # Update Facts table**

**# Coverage Table**

conn.execute(text("""

CREATE TABLE IF NOT EXISTS coverage (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    country\_id INTEGER,

    year\_id INTEGER,

    vaccine\_id INTEGER,

    coverage\_category TEXT,

    target\_number INTEGER,

    doses\_administered INTEGER,

    coverage REAL,

    FOREIGN KEY (country\_id) REFERENCES countries(id),

    FOREIGN KEY (vaccine\_id) REFERENCES vaccines(id),

    FOREIGN KEY (year\_id) REFERENCES years(id)

)

"""))

**# Incidence Rate Table**

conn.execute(text("""

CREATE TABLE IF NOT EXISTS incidence\_rate (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    country\_id INTEGER,

    year\_id INTEGER,

    disease\_id INTEGER,

    denominator TEXT,

    incidence\_rate REAL,

    FOREIGN KEY (country\_id) REFERENCES countries(id),

    FOREIGN KEY (disease\_id) REFERENCES diseases(id),

    FOREIGN KEY (year\_id) REFERENCES years(id)

)

"""))

**# Reported Cases Table**

conn.execute(text("""

CREATE TABLE IF NOT EXISTS reported\_cases (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    country\_id INTEGER,

    year\_id INTEGER,

    disease\_id INTEGER,

    cases INTEGER,

    FOREIGN KEY (country\_id) REFERENCES countries(id),

    FOREIGN KEY (disease\_id) REFERENCES diseases(id),

    FOREIGN KEY (year\_id) REFERENCES years(id)

)

"""))

**# Vaccine Introduction Table**

conn.execute(text("""

CREATE TABLE IF NOT EXISTS vaccine\_introduction (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    country\_id INTEGER,

    year\_id INTEGER,

    vaccine\_name TEXT,

    introduced INTEGER,

    FOREIGN KEY (country\_id) REFERENCES countries(id),

    FOREIGN KEY (year\_id) REFERENCES years(id)

)

"""))

**# Vaccine Schedule Table**

conn.execute(text("""

CREATE TABLE IF NOT EXISTS vaccine\_schedule (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    country\_id INTEGER,

    year\_id INTEGER,

    vaccine\_name TEXT,

    schedule\_round TEXT,

    target\_pop TEXT,

    target\_pop\_desc TEXT,

    geo\_area TEXT,

    age\_administered TEXT,

    source\_comment TEXT,

    FOREIGN KEY (country\_id) REFERENCES countries(id),

    FOREIGN KEY (year\_id) REFERENCES years(id)

)

"""))

**Step 9: # Inserting Cleaned Data into Normalized SQL Table**

**# Load Dimension Tables into Pandas for Mapping**

**# Load dimensions for ID mapping**

countries\_ref = pd.read\_sql('SELECT \* FROM countries', conn)

vaccines\_ref = pd.read\_sql('SELECT \* FROM vaccines', conn)

diseases\_ref = pd.read\_sql('SELECT \* FROM diseases', conn)

years\_ref = pd.read\_sql('SELECT \* FROM years', conn)

**# Insert Data into 'coverage' Table**

# Map IDs for coverage

coverage\_temp = (

    coverage\_df

    .merge(countries\_ref, left\_on='code', right\_on='code')

    .merge(vaccines\_ref, left\_on='antigen', right\_on='code')

    .merge(years\_ref, left\_on='year', right\_on='year')

)

coverage\_cleaned = coverage\_temp[[

    'id\_x', 'id\_y', 'id', 'coverage\_category', 'target\_number', 'doses', 'coverage'

]]

coverage\_cleaned.columns = [

    'country\_id', 'vaccine\_id', 'year\_id', 'coverage\_category',

    'target\_number', 'doses\_administered', 'coverage'

]

**# Insert Data into 'incidence\_rate' Table**

# Map IDs for incidence

incidence\_temp = (

    incidence\_df

    .merge(countries\_ref, left\_on='code', right\_on='code')

    .merge(diseases\_ref, left\_on='disease', right\_on='code')

    .merge(years\_ref, left\_on='year', right\_on='year')

)

incidence\_cleaned = incidence\_temp[[

    'id\_x', 'id\_y', 'id', 'denominator', 'incidence\_rate'

]]

incidence\_cleaned.columns = [

    'country\_id', 'disease\_id', 'year\_id', 'denominator', 'incidence\_rate'

]

incidence\_cleaned.to\_sql('incidence\_rate', conn, if\_exists='append', index=False)

**# Insert Data into 'reported\_cases' Table**

**# Map IDs for cases**

cases\_temp = (

    cases\_df

    .merge(countries\_ref, left\_on='code', right\_on='code')

    .merge(diseases\_ref, left\_on='disease', right\_on='code')

    .merge(years\_ref, left\_on='year', right\_on='year')

)

cases\_cleaned = cases\_temp[['id\_x', 'id\_y', 'id', 'cases']]

cases\_cleaned.columns = ['country\_id', 'disease\_id', 'year\_id', 'cases']

cases\_cleaned.to\_sql('reported\_cases', conn, if\_exists='append', index=False)

**# Insert Data into 'vaccine\_introduction' Table**

**# Map IDs for intro**

intro\_temp = (

    intro\_df

    # Change 'ISO\_3\_Code' to 'code' to match the cleaned DataFrame column name

    .merge(countries\_ref, left\_on='code', right\_on='code')

    .merge(years\_ref, left\_on='year', right\_on='year')

)

intro\_cleaned = intro\_temp[['id\_x', 'id\_y', 'description', 'intro']]

# The column names in intro\_temp after merging are now based on the merged dataframes.

# 'id' comes from countries\_ref (country\_id), 'id\_y' comes from years\_ref (year\_id)

# 'description' comes from the original intro\_df, and 'intro' comes from the original intro\_df.

# The columns need to be mapped to the correct target table column names.

intro\_cleaned.columns = ['country\_id', 'year\_id', 'vaccine\_name', 'introduced']

intro\_cleaned.to\_sql('vaccine\_introduction', conn, if\_exists='append', index=False)

**# Insert Data into 'vaccine\_schedule' Table**

**# Merge for country and year mapping**

schedule\_temp = (

    schedule\_df

    .merge(countries\_ref, left\_on='code', right\_on='code')

    .merge(years\_ref, left\_on='year', right\_on='year')

)

# Select and rename columns

schedule\_cleaned = schedule\_temp[[

    'id\_x', 'id\_y', 'vaccine\_description', 'schedulerounds',

    'targetpop', 'targetpop\_description', 'geoarea',

    'ageadministered', 'sourcecomment'

]]

schedule\_cleaned.columns = [

    'country\_id', 'year\_id', 'vaccine\_name', 'schedule\_round',

    'target\_pop', 'target\_pop\_desc', 'geo\_area',

    'age\_administered', 'source\_comment'

]

# Insert into the database

schedule\_cleaned.to\_sql('vaccine\_schedule', conn, if\_exists='append', index=False)

**# We now have a fully normalized SQL database with:**

**# a. Foreign keys enforcing referential integrity**

**# b. Dimension tables (countries, vaccines, diseases, years)**

**# c. Populated fact tables ready for Power BI**

**Next Steps**

**Here’s what remains for the full project:**

**🔹 Power BI Integration**

* **Connect Power BI to the SQLite DB (or exported .csv or .db file)**
* **Build relationships in Power BI model view**

**🔹 Interactive Dashboards**

1. **Create slicers, charts, maps to visualize:**
   * **Vaccination coverage**
   * **Incidence rates**

**Check the path and download the db file:**

import os

os.path.exists("/content/vaccination.db")

from google.colab import files

files.download("/content/vaccination.db")

**# Connecting to Database**

from sqlalchemy import create\_engine, inspect

import pandas as pd

engine = create\_engine('sqlite:///vaccination.db')

conn = engine.connect()

**# List all tables in the database**

inspector = inspect(engine)

tables = inspector.get\_table\_names()

print("Tables in the database:", tables)

**# Preview data from each table**

for table in tables:

    print(f"\n--- Sample data from table '{table}' ---")

    query = f'SELECT \* FROM "{table}" LIMIT 5;'

    df\_sample = pd.read\_sql(query, conn)

    print(df\_sample)

**Installed Power BI for Desktop**

**To connect SQLite DB, PowerBI doesnot support SQLite3, so download and setup ODBC driver**

**Option 1: Use ODBC Driver for SQLite (Recommended)**

**🔹 Step 1: Install SQLite ODBC Driver**

1. Go to: https://www.ch-werner.de/sqliteodbc/
2. Scroll to the **Windows binaries** section.
3. Download and install the version that matches your system (likely **sqliteodbc\_w64.exe** for 64-bit).

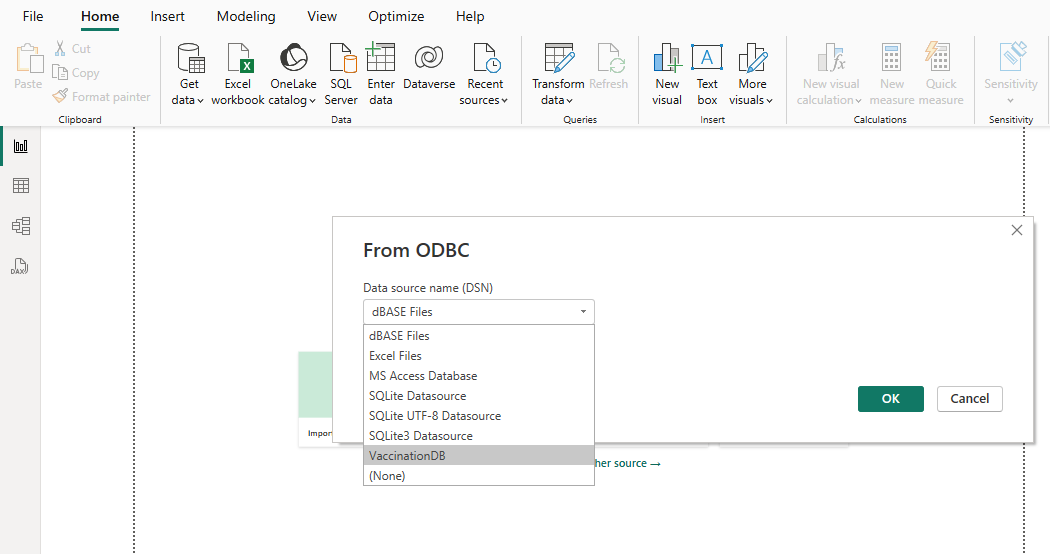
Error: "No usable SQLite3 DLL found", so downloaded .dll file separately and added.

**🔹 Step 2: Configure ODBC DSN (Data Source Name)**

1. Open **ODBC Data Sources (64-bit)** from Windows Start Menu.
2. Go to the **System DSN** tab and click **Add**.
3. Choose **SQLite3 ODBC Driver** and click **Finish**.
4. Give it a **Name** (e.g., VaccinationDB) and point the **Database Name** to your .db file.

**🔹 Step 3: Connect in Power BI**

1. In Power BI Desktop, click **Get Data** > **ODBC**
2. Choose the DSN you just created (VaccinationDB)
3. Navigate and select tables → Load



Loaded all five tables to Power BI. Now Let’s Visualise using Power BI

**Power BI Visualizations**

**Clarity and Relevance**

Visuals were developed using Power BI Desktop. The following types were implemented:

* **Visuals:** Country-level vaccination coverage and disease incidence
* **Trend Lines & Bar Charts:** Time trends in vaccine uptake and disease reduction
* **Scatter Plots:** Correlation between vaccination coverage and disease incidence
* **KPI Cards:** Target coverage achievement, coverage percentage changes

**Interactivity**

* **Slicers:** Enabled filtering by year, country, disease, antigen, and WHO region
* **Tooltip customization:** Displaying extra context like target population, doses given

The dashboard is interactive and intuitive.

**Insights and Actionability**

**Public Health Policy**

* Countries with high coverage and low disease incidence were identified as success cases.

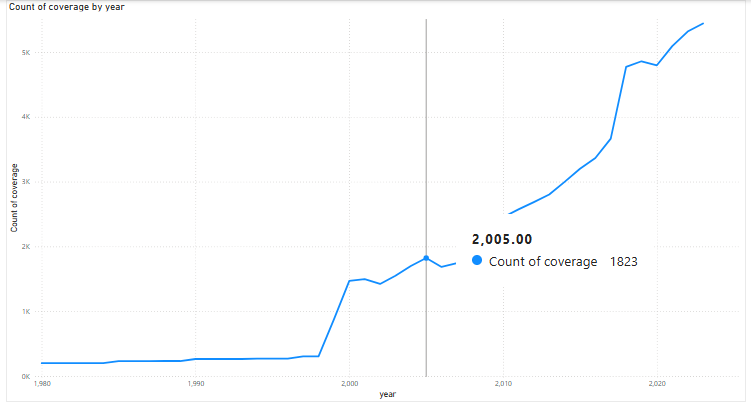
**Resource Allocation**

* Slicers help identify years of vaccine introduction timelines

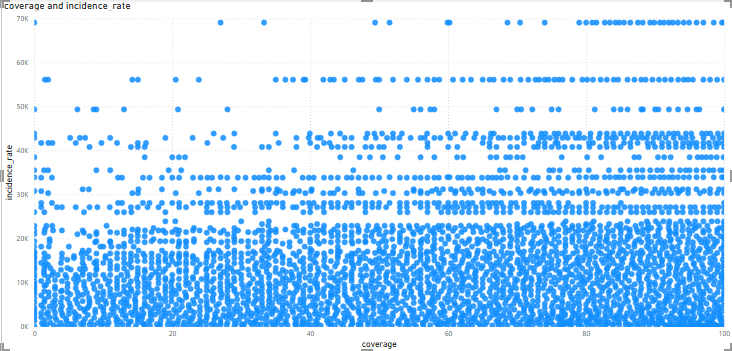
**Disease Prevention**

* Visual trends show a sharp drop in reported cases after vaccine introduction for diseases like Measles and Hepatitis B.
* Incidence vs. coverage scatter plots help identify outliers where high vaccination does not correlate with disease reduction, flagging further investigation.

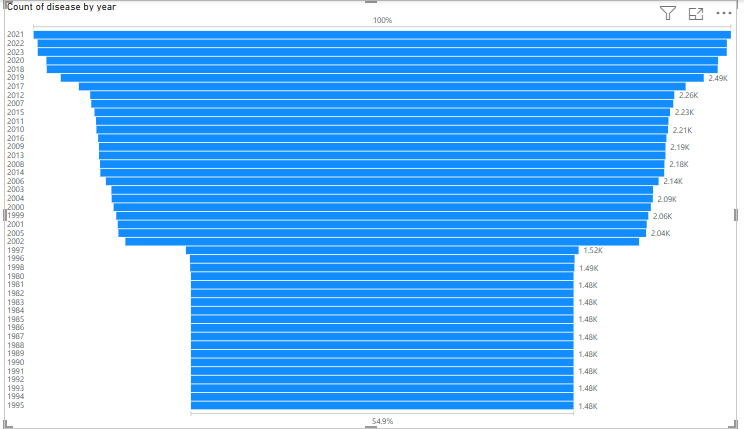
1. **Vaccination Coverage Over Time (Line Chart)**



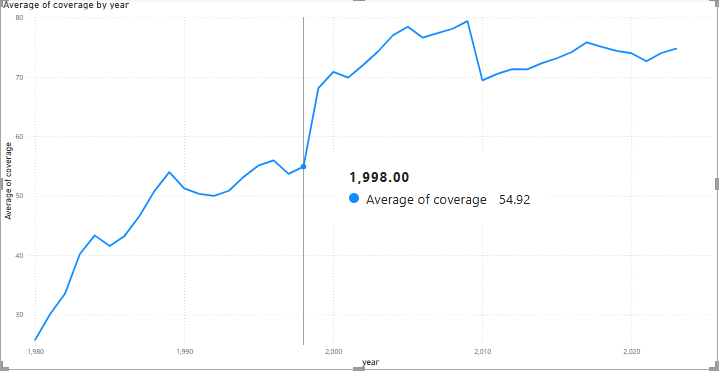
1. **Scatter Plot – Correlate vaccination coverage with incidence rate:**



1. **Count of disease over year**

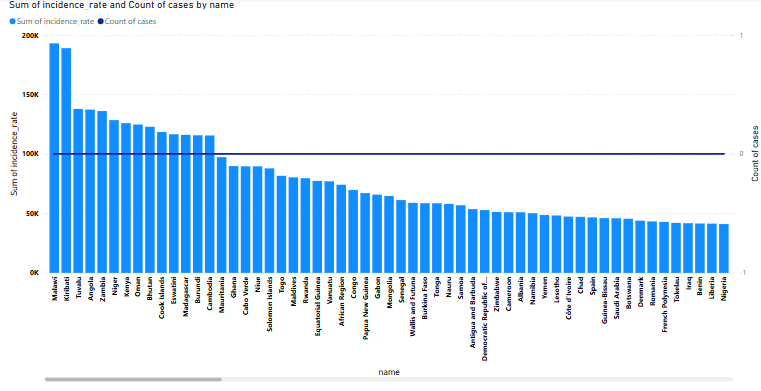


1. **Average Vaccination Coverage by Year**

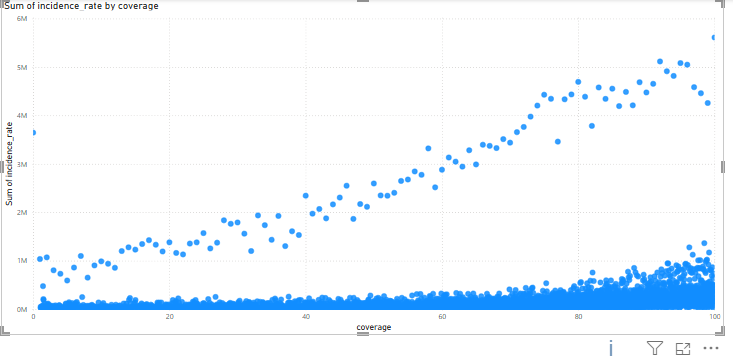


Coverage area is improving over time.

1. **Incident rate by country and count of cases**



1. **Vaccination coverage vs Incident rate**



1. **Metrics: - KPI Cards**

Created Custom DAX measure

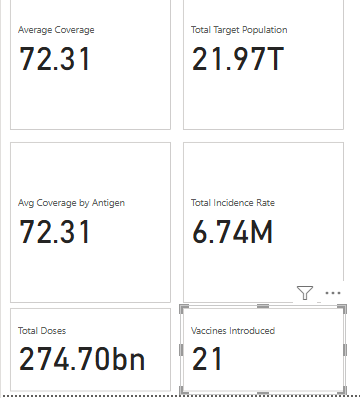
* Average Coverage = AVERAGE('coverage-data'[Coverage])
* Total Reported Cases = SUM('reported-cases-data'[Cases])
* Total Incidence Rate = SUM('incidence-rate'[Incidence rate])
* Vaccines Introduced = DISTINCTCOUNT('vaccine-introduction-data'[Description])
* Avg Coverage by Antigen =

CALCULATE(

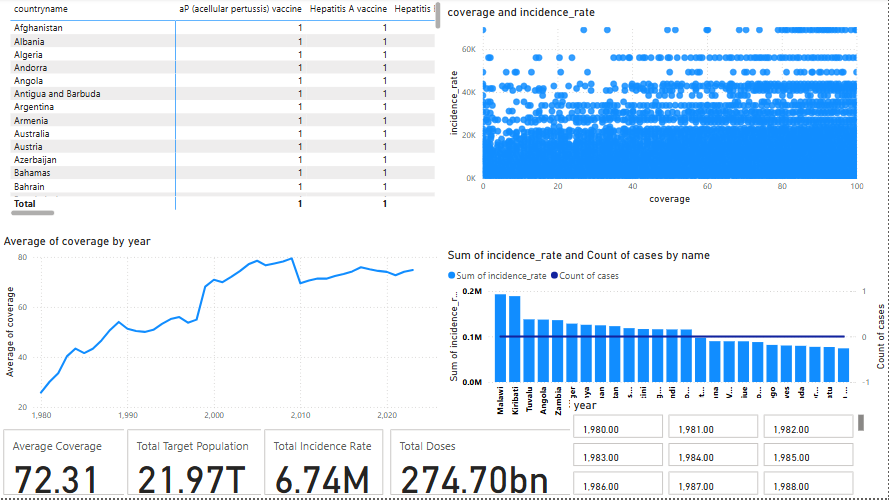
AVERAGE('coverage-data'[Coverage]),

ALLEXCEPT('coverage-data', 'coverage-data'[Antigen\_description], 'coverage-data'[Name])

)



**Interactive dashboard:**



**🚀 Challenges and Solutions**

| **Challenge** | **Solution** |
| --- | --- |
| Missing ODBC driver for SQLite | Installed SQLite ODBC driver manually and configured DSN |
| Column name mismatches | Cleaned and renamed columns during preprocessing |
| Data format errors in Power BI | Ensured proper typing and relationships in SQL tables |
| Scatter plots lacked detail fields | Used slicers and tooltips instead |
| DATEADD not working on non-date field | Converted year integers to date-compatible format or adjusted DAX logic |

**Deliverables Summary**

* Python scripts for loading, cleaning, and transforming Excel datasets in Colab
* SQL scripts for creating and populating normalized SQLite database
* .db file for use in Power BI
* Power BI .pbix file with dashboards and visuals
* Documentation covering the full project lifecycle (this document)