**Census Data Standardization and Analysis Pipeline**

**Technologies Used: Python, SQL, MongoDB, Streamlit**

**Problem Statement :**

The task is to clean, process, and analyze census data from a given source, including data renaming, missing data handling, state/UT name standardization, new state/UT formation handling, data storage, database connection, and querying. The goal is to ensure uniformity, accuracy, and accessibility of the census data for further analysis and visualization.

**Task 1: Rename the Column names**

For uniformity in the datasets and taking into consideration the census year, we need to rename some columns.

State name  to State/UT

District name  to District

Male\_Literate to Literate\_Male

Female\_Literate to Literate\_Female

Rural\_Households  to Households\_Rural

Urban\_ Households  to Households\_Urban

Age\_Group\_0\_29 to Young\_and\_Adult

Age\_Group\_30\_49 to Middle\_Aged

Age\_Group\_50 to Senior\_Citizen

Age not stated to Age\_Not\_Stated

**Solution:**

import pandas as pd

census\_raw = pd.read\_csv("D:\\python workbook\\census\_2011.csv")

#to rename columns

census\_raw.rename(*columns*={

    "State name": "State\_or\_UT",

    "District name": "District",

    "Male\_Literate": "Literate\_Male",

    "Female\_Literate": "Literate\_Female",

    "Rural\_Households": "Households\_Rural",

    "Urban\_Households": "Households\_Urban",

    "Age\_Group\_0\_29": "Young\_and\_Adult",

    "Age\_Group\_30\_49": "Middle\_Aged",

    "Age\_Group\_50": "Senior\_Citizen",

    "Age not stated": "Age\_Not\_Stated"

}, *inplace*=True)

**Task 2: Rename State/UT Names**

The State/UT names are in all caps in the census data, For uniformity across datasets we use the names so that only the first character of each word in the name is in upper case and the rest are in lower case. However, if the word is “and” then it should be all lowercase.

Examples:

Andaman and  Nicobar Islands

Arunachal Pradesh

Bihar

**Solution:**

def normalize\_state\_ut\_Camel(*state\_name*):

    words = state\_name.lower().split()

    for i, word in enumerate(words):

        if word != "and" or i == 0:

            words[i] = word.capitalize()

    return " ".join(words)

census\_raw["State\_or\_UT"] = census\_raw["State\_or\_UT"].apply(normalize\_state\_ut\_Camel)

**Task 3: New State/UT formation**

* In 2014 Telangana was formed after it split from Andhra Pradesh, The districts that were included in Telangana are stored in *Data/Telangana.txt* . Read the text file and Rename the State/UT From “Andhra Pradesh” to “Telangana” for the given districts.

* In 2019 Ladakh was formed after it split from Jammu and Kashmir, which included the districts Leh and Kargil.  Rename the State/UT From “Jammu and Kashmir” to “Ladakh” for the given districts.

telangana\_districts=open("Telangana.txt", "r")

telangana\_districts\_list= telangana\_districts.read().splitlines()

ladakh\_district\_list=['Jammu and Kashmir']

census\_raw.loc[census\_raw['District'].isin(telangana\_districts\_list), 'State\_or\_UT'] = 'Telangana'

census\_raw.loc[census\_raw['District'].isin(ladakh\_district\_list), 'State\_or\_UT'] = 'Ladakh'

**Task 4: Find and process Missing Data**

Find and store the percentage of data missing for the columns.

Some data can be found and filled in by using information from other cells. Try to find the correct data by using information from other cells and filling it in. Find and store the percentage of data missing for each column.

Hint:

* Population = Male + Female
* Literate = Literate\_Male + Literate\_Female
* Population  = Young\_and\_Adult+  Middle\_Aged + Senior\_Citizen + Age\_Not\_Stated
* Households = Households\_Rural + Households\_Urban

compares the amount of missing data before and after the data-filling process was done.

**Solution:**

missing\_value=(census\_raw.isnull().sum())

    print("missing values before processing missing data")

    print(missing\_value)

    census\_raw['Population'].fillna(census\_raw['Male']+census\_raw['Female'],*inplace*=True)

    census\_raw['Male'].fillna(census\_raw['Population']-census\_raw['Female'],*inplace*=True)

    census\_raw['Female'].fillna(census\_raw['Population']-census\_raw['Male'],*inplace*=True)

    census\_raw['Literate'] .fillna( census\_raw['Literate\_Male'] + census\_raw['Literate\_Female'],*inplace*=True)

    census\_raw['Literate\_Male'].fillna(census\_raw['Literate']-census\_raw['Literate\_Female'],*inplace*=True)

    census\_raw['Literate\_Female'].fillna(census\_raw['Literate']-census\_raw['Literate\_Male'],*inplace*=True)

    census\_raw['SC'].fillna(census\_raw['Male\_SC']+census\_raw['Female\_SC'],*inplace*=True)

    census\_raw['Male\_SC'].fillna(census\_raw['SC']-census\_raw['Female\_SC'],*inplace*=True)

    census\_raw['Female\_SC'].fillna(census\_raw['SC']-census\_raw['Male\_SC'],*inplace*=True)

    census\_raw['ST'].fillna(census\_raw['Male\_ST']+census\_raw['Female\_ST'],*inplace*=True)

    census\_raw['Male\_ST'].fillna(census\_raw['ST']-census\_raw['Female\_ST'],*inplace*=True)

    census\_raw['Female\_ST'].fillna(census\_raw['ST']-census\_raw['Male\_SC'],*inplace*=True)

    census\_raw['Workers'].fillna(census\_raw['Male\_Workers']+census\_raw['Female\_Workers'],*inplace*=True)

    census\_raw['Male\_Workers'].fillna(census\_raw['Workers']-census\_raw['Female\_Workers'],*inplace*=True)

    census\_raw['Female\_Workers'].fillna(census\_raw['Workers']-census\_raw['Male\_Workers'],*inplace*=True)

    census\_raw['Total\_Education'].fillna(census\_raw['Literate\_Education']+census\_raw['Illiterate\_Education'],*inplace*=True)

    census\_raw['Literate\_Education'].fillna(census\_raw['Total\_Education']-census\_raw['Illiterate\_Education'],*inplace*=True)

    census\_raw['Illiterate\_Education'].fillna(census\_raw['Total\_Education']-census\_raw['Literate\_Education'],*inplace*=True)

    missing\_value=(census\_raw.isnull().sum())

    print("missing values after processing missing data")

    print(missing\_value)

    census\_raw.to\_csv("updated\_missing\_values\_1.csv", *index*=False)

    print("Handled possible missing data - step 4 completed ")

**Task  5: Save Data to MongoDB**

Save the processed data to mongoDB with a collection named “*census*” .

**Solution:**

# Load the processed data into a DataFrame

    processed\_data = pd.read\_csv("updated\_missing\_values\_1.csv")

    # Convert DataFrame to dictionary

    data\_dict = processed\_data.to\_dict(*orient*='records')

    client = MongoClient("mongodb+srv://manishdeva:manish1234@cluster0.3rnfpb7.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0")

    db = client['Census\_demo']

    collection = db['census']

    collection=db.get\_collection("census")

    result=collection.delete\_many({})

    # Insert data into MongoDB collection

    collection.insert\_many(data\_dict)

    client.close()

    print("Data saved to MongoDB successfully.")

    print("step 5 completed")

**Task 6: Database connection and data upload**

Data should be fetched from the mongoDB and to be uploaded to a relational database using python code . The table names should be the same as the file names without the extension.

The primary key and foreign key constraints should be included in the tables wherever required.

**Solution:**

# MongoDB connection details

    mongo\_client = MongoClient("mongodb+srv://manishdeva:manish1234@cluster0.3rnfpb7.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0")

    mongo\_db = mongo\_client['Census\_demo']

    mongo\_collection = mongo\_db['census']

    # Fetch data from MongoDB

    data = *list*(mongo\_collection.find())

    # Convert MongoDB data to DataFrame

    df = pd.DataFrame(data)

    df.rename(*columns* = {'Households\_with\_TV\_Computer\_Laptop\_Telephone\_mobile\_phone\_and\_Scooter\_Car'

                        :'Households\_with\_TV\_Computer\_Laptop\_Telephone\_mobile\_Scooter\_Car',

                        'Type\_of\_latrine\_facility\_Night\_soil\_disposed\_into\_open\_drain\_Households':

                        'Type\_of\_latrine\_facility\_Night\_soil\_disposed\_into\_open\_drain',

                        'Type\_of\_latrine\_facility\_Flush\_pour\_flush\_latrine\_connected\_to\_other\_system\_Households':

                        'Type\_of\_latrine\_facility\_pour\_flush\_joined\_to\_other\_system',

                        'Not\_having\_latrine\_facility\_within\_the\_premises\_Alternative\_source\_Open\_Households':

                        'Not\_having\_latrine\_facility\_within\_the\_premises\_Alter\_source',

                        'Main\_source\_of\_drinking\_water\_Handpump\_Tubewell\_Borewell\_Households':

                        'Main\_source\_of\_drinking\_water\_Handpump\_Tubewell\_Borewell\_Houses',

                        'Main\_source\_of\_drinking\_water\_Other\_sources\_Spring\_River\_Canal\_Tank\_Pond\_Lake\_Other\_sources\_\_Households':

                        'Main\_source\_of\_water\_Other\_Spring\_River\_Canal\_Tank\_Pond\_Lake'}, *inplace* = True)

    print("\nAfter modifying first column:\n", df.columns)

    print(df)

    df['\_id'] = df['\_id'].astype(*str*)

    # MySQL connection details

    mysql\_engine = create\_engine('mysql+mysqlconnector://root:@localhost:3306/census\_1')

    # Define table schema

    metadata = MetaData()

    # Create tables in the MySQL database

    metadata.create\_all(mysql\_engine)

    # Insert data into MySQL

    df.to\_sql('census', mysql\_engine, *if\_exists*='replace', *index*=False)

    print("Data has been successfully saved to MySQL.")

    print("step 6 completed")

**Task 7: Run Query on the database and show output on Streamlit**

**Solution:**

mydb = mysql.connector.connect(

*host*="localhost",

*user*="root",

*password*="",

    )

    mycursor = mydb.cursor(*buffered*=True)

    mycursor.execute("SELECT district, SUM(population) AS total\_population FROM census\_1.census GROUP BY district")

    out=mycursor.fetchall()

    print("1. What is the total population of each district?")

    print(tabulate(out,*headers*=[i[0] for i in mycursor.description],  *tablefmt*='psql'))

    mycursor.execute("SELECT district,sum(Literate\_Male),sum(Literate\_Female) FROM census\_1.census GROUP BY district")

    out=mycursor.fetchall()

    print("2.How many literate males and females are there in each district?")

    print(tabulate(out,*headers*=[i[0] for i in mycursor.description],  *tablefmt*='psql'))

    mycursor.execute("""SELECT district,ROUND(SUM(Male\_Workers) \* 100.0 / SUM(Workers)) AS male\_percentage,

    ROUND(SUM(Female\_Workers) \* 100.0 / SUM(Workers)) AS female\_percentage

    FROM

    census\_1.census

    GROUP BY

    district""")

    out=mycursor.fetchall()

    print("3.What is the percentage of workers (both male and female) in each district?")

    print(tabulate(out,*headers*=[i[0] for i in mycursor.description],  *tablefmt*='psql'))

st.title('Display Database Data')

# Set up a connection to your SQLite database

mydb = mysql.connector.connect(

*host*="localhost",

*user*="root",

*password*="",

)

mycursor = mydb.cursor(*buffered*=True)

# Write SQL query to retrieve data from the database

query1 = "SELECT district, SUM(population) AS total\_population FROM census\_1.census GROUP BY district"

mycursor.execute(query1)

# Fetch all the data retrieved by the query

data = mycursor.fetchall()

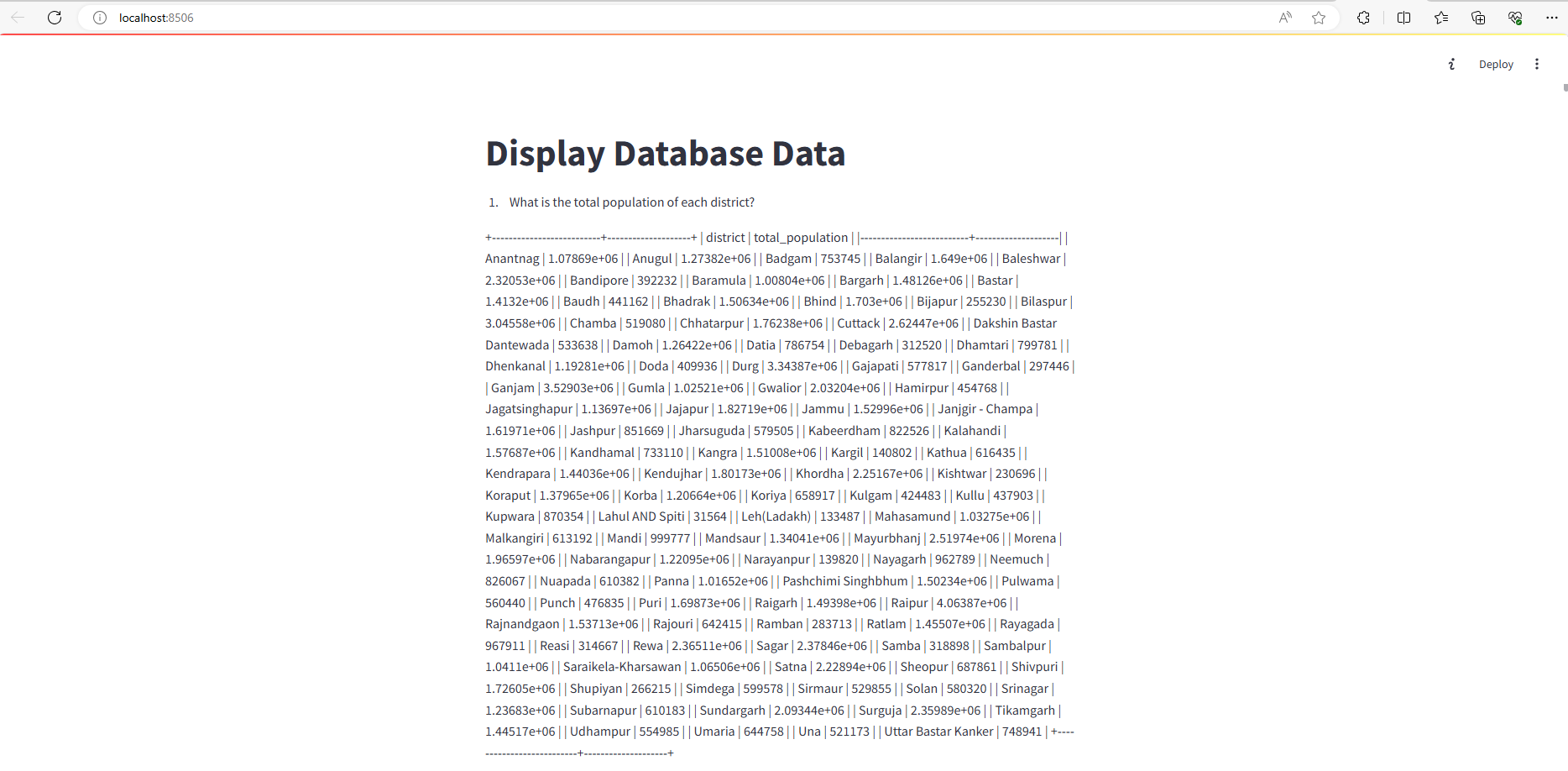
data=(tabulate(data,*headers*=[i[0] for i in mycursor.description],  *tablefmt*='psql'))

# Streamlit app code

# Display the data in a table using Streamlit

st.write("1. What is the total population of each district?")

st.write(data)

****