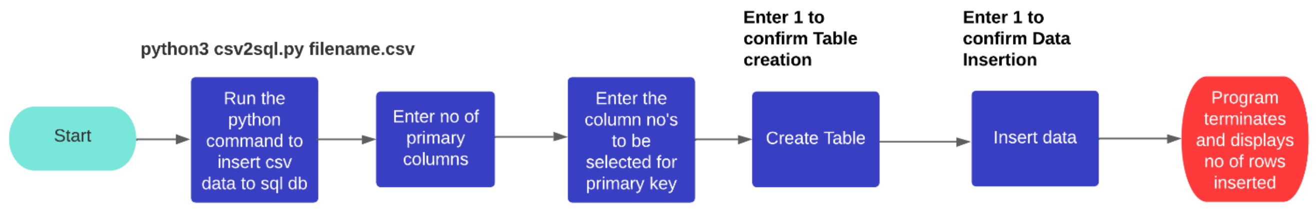
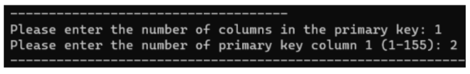
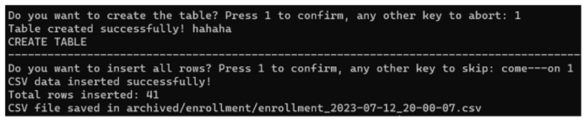
Table of Contents

CSV/Excel to PostgreSQL dB table


python3 csv2sql.py filename.csv
Start
Run the python command to insert csv data to sql db
Enter no of primary columns
Enter the column no's to be selected for primary key
Enter 1 to confirm Table creation
Create Table
Enter 1 to confirm Data Insertion
Insert data
Program terminates and displays no of rows inserted

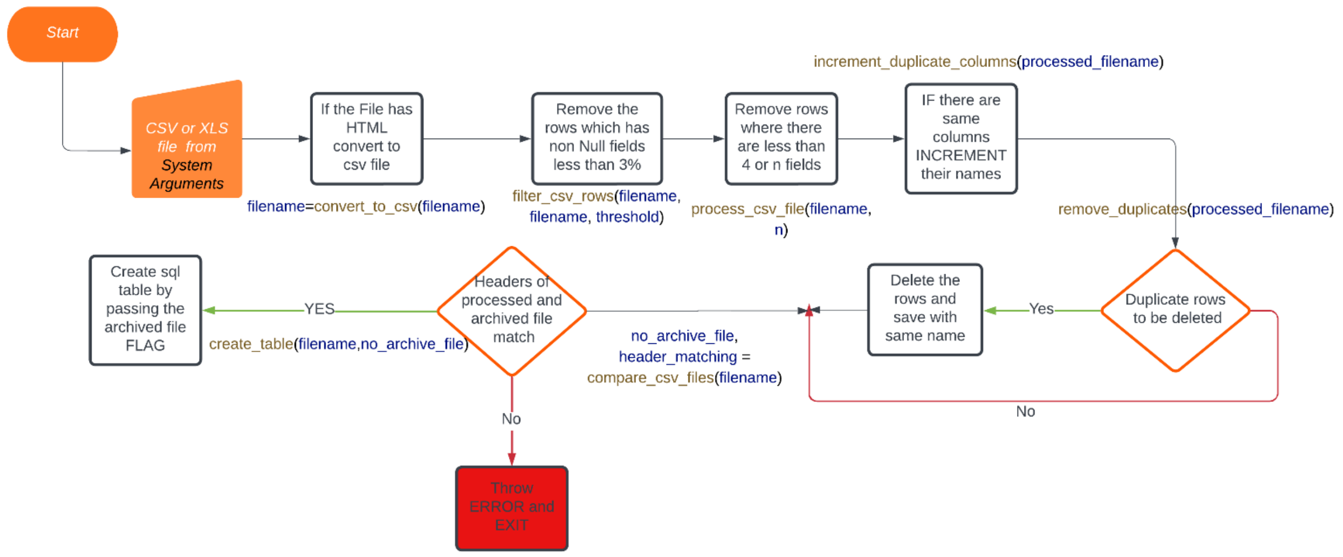

```
-----------------------------------
Please enter the number of columns in the primary key: 1
Please enter the number of primary key column 1 (1-155): 2
----------------------------------------------------------
```


```
Do you want to create the table? Press 1 to confirm, any other key to abort: 1
Table created successfully! hahaha
CREATE TABLE
-----------------------------------------------------------------------------------
Do you want to insert all rows? Press 1 to confirm, any other key to skip: come---on 1
CSV data inserted successfully!
Total rows inserted: 41
CSV file saved in archived/enrollment/enrollment_2023-07-12_20-00-07.csv
```

The detailed flow chart in two steps:

Flow Chart 1:

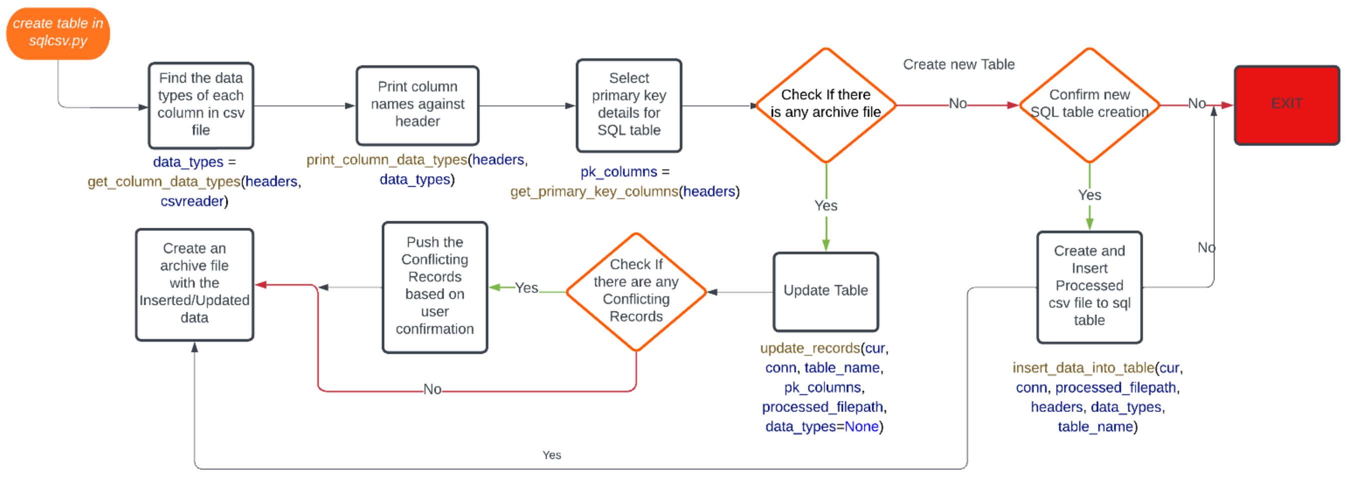
This flow chart explains the initial process where a file is processed for any noise or unwanted data and takes users confirmation and cleans the data. The data is then saved to a processed folder which will be then used to create a SQL table.


Start
CSV or XLS file from System Arguments
If the File has HTML convert to csv file
filename=convert_to_csv(filename)
Remove the rows which has non Null fields less than 3%
filter_csv_rows(filename, filename, threshold)
Remove rows where there are less than 4 or n fields
process_csv_file(filename, n)
increment_duplicate_columns(processed_filename)
IF there are same columns INCREMENT their names
remove_duplicates(processed_filename)
Duplicate rows to be deleted
Yes
Delete the rows and save with same name
No
no_archive_file, header_matching = compare_csv_files(filename)
Headers of processed and archived file match
YES
create_table(filename,no_archive_file)
Create sql table by passing the archived file FLAG
No
Throw ERROR and EXIT

CSV or Excel Pre-processing

Flow chart 2:

This flow chart follows the first flow chart and mainly covers the SQL table creation or insertion/updation in detail. The data types of the columns are decided by script automatically considering first 100 rows which can be changed if needed. We take user input for selecting the primary key based on the column nos. If there is any archive file created, we update the SQL table since the data is already inserted else, we create a new table and insert the data. While updating the records If there are any conflicting records, we ask for user input to update them or ignore them.



create table in sqlcsv.py
Find the data types of each column in csv file
data_types = get_column_data_types(headers, csvreader)
Print column names against header
print_column_data_types(headers, data_types)
Select primary key details for SQL table
pk_columns = get_primary_key_columns(headers)
Check If there is any archive file
Create new Table
No
Confirm new SQL table creation
No
EXIT
Yes
Yes
Update Table
update_records(cur, conn, table_name, pk_columns, processed_filepath, data_types=None)
Create and Insert Processed csv file to sql table
insert_data_into_table(cur, conn, processed_filepath, headers, data_types, table_name)
No
Check If there are any Conflicting Records
Yes
Push the Conflicting Records based on user confirmation
No
Create an archive file with the Inserted/Updated data
Yes

SQL table creation and data insertion

Installation and setup:

Open the ubuntu server or instance and run the following commands to setup the server.

```
apt-get update
apt-get install nano
apt-get install python3
apt-get install python3-pip
apt-get install postgresql
```

You might be asked to select region (select 2 America)

```
service postgresql start

su - postgres -c "createuser alok"

su - postgres -c "psql -c \"ALTER USER alok WITH PASSWORD 'reddy';\""

su - postgres -c "createdb meddb"

su - postgres -c "psql -c \"GRANT ALL PRIVILEGES ON DATABASE meddb TO alok;\""
```

Alternate commands for a sudo user:

```
sudo apt-get update

sudo apt-get install nano

sudo apt-get install python3

sudo apt-get install python3-pip

sudo apt-get install postgresql
```

You might be asked to select region (select 2 America)

```
sudo service postgresql start

sudo -u postgres createuser alok

sudo -u postgres psql -c "ALTER USER alok WITH PASSWORD 'reddy';"

sudo -u postgres createdb meddb

sudo -u postgres psql -c "GRANT ALL PRIVILEGES ON DATABASE meddb TO alok;"
```

---

requirements.txt

```
pandas, psycopg2-binary, lxml
```

There are three python libraries that are needed for the data insertion. We need to run the above requirements.txt file using below command. This command installs all required python packages.

```
# pip3 install -r requirements.txt

# python3 csv2sql.py enrollment.xls
```

Step by Step Explanation:

1) Have the csv or Excel files which are to be processed and saved to the database in the same directory of csv2sql.py file.
2) Run the command as below for sql table creation and insertion.
   This command is to be run in terminal with python and necessary packages installed.
   Command for csv: `python3 csv2sql.py filename.csv`
   Command for excel: `python3 csv2sql.py filename.xls`
   `python3 csv2sql.py filename.xlsx`
   Ex: `python3 make_trimmedcsv.py publications.csv`
3) **html to csv:**
   If the data file (csv or excel) provided has any html content it will be parsed using beautifulsoup and table is saved with same name as csv file.
4) **Filter csv rows with a given percentage of nonnull fields:**
   The csv will be processed to remove the rows that have fields less than the given threshold (hardcode to 3%) can be changed.
5) **Duplicate rows:**
   The script looks for any duplicate rows and if found the user will be asked to confirm to delete the repeating records.
6) **Process the new csv file:**
   We now process the new csv file and check for any rows having more than 4 fields (this can be changed) and save it to processed folder.
7) **Check for archived folder:**
   We then check for the archived folder with the filename that is processed. If there are any files, we check the headers and if headers don't match, we exit the program. If there are no files in archived folder for the processed file, we go ahead with new SQL table creation.
8) **Create Primary Key:**
   The first 100 rows in the csv file are used to determine the data of the columns. The columns are displayed with numbers for the user to select the primary key.
9) **Table Creation:**

Once the columns are selected a create query is generated and executed upon user confirmation.

10) **Insert Records:**

The user is then asked for the confirmation to insert the records.

11) **Update Records:**

If there are files in the archived folder for the given processed file, we update the records from the csv file.
We also check for any conflicting rows and insert them upon user confirmation.

Command to connect the PostgreSQL DB:

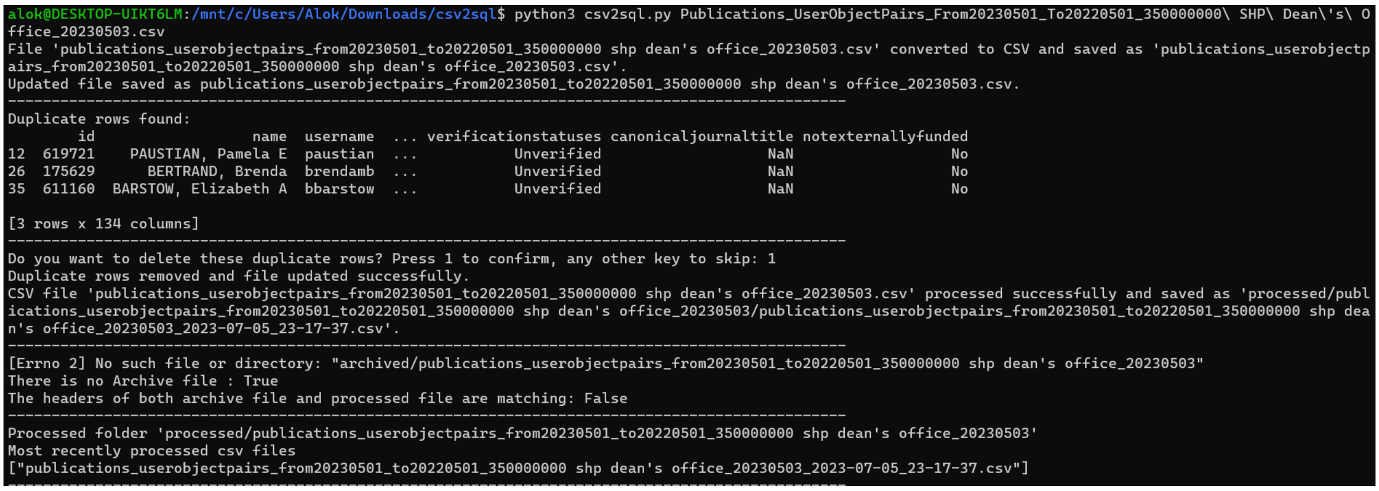
The data pushed can be verified by connecting the postgres database and running few select queries.

```
psql -d dbname -U username -W -h hostname -p portnumber
```

```
psql -d meddb -U alok -W -h localhost -p 5432
```

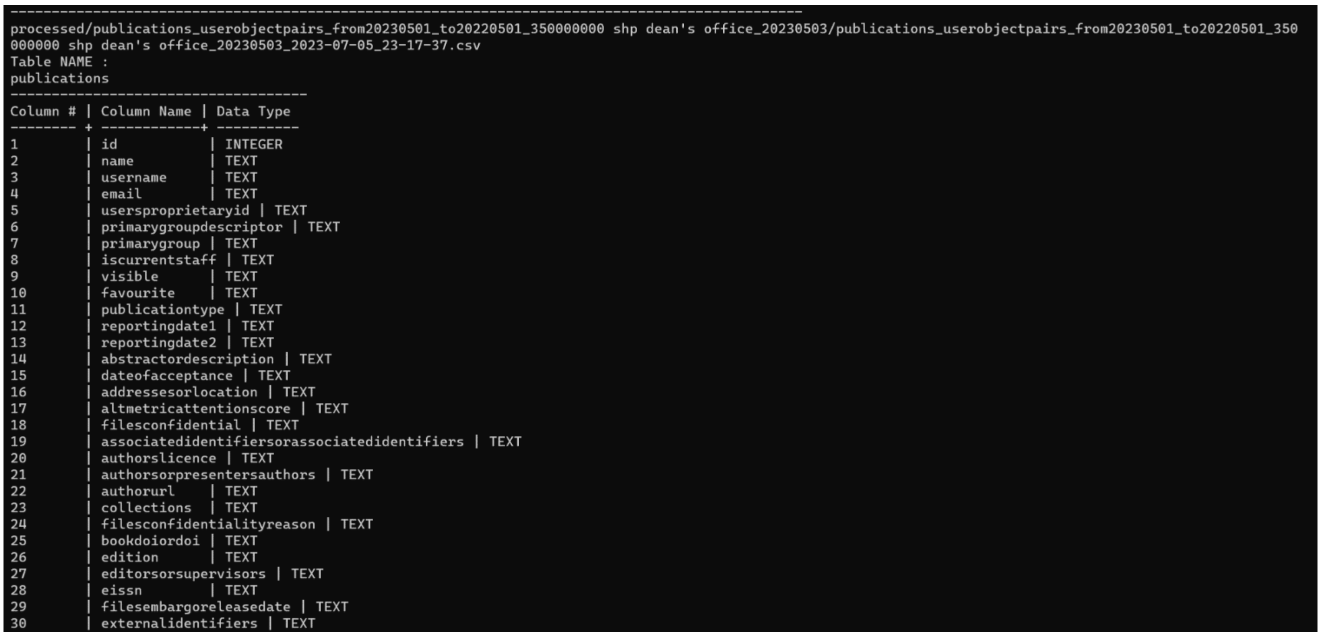
password: reddy

Screen shots:



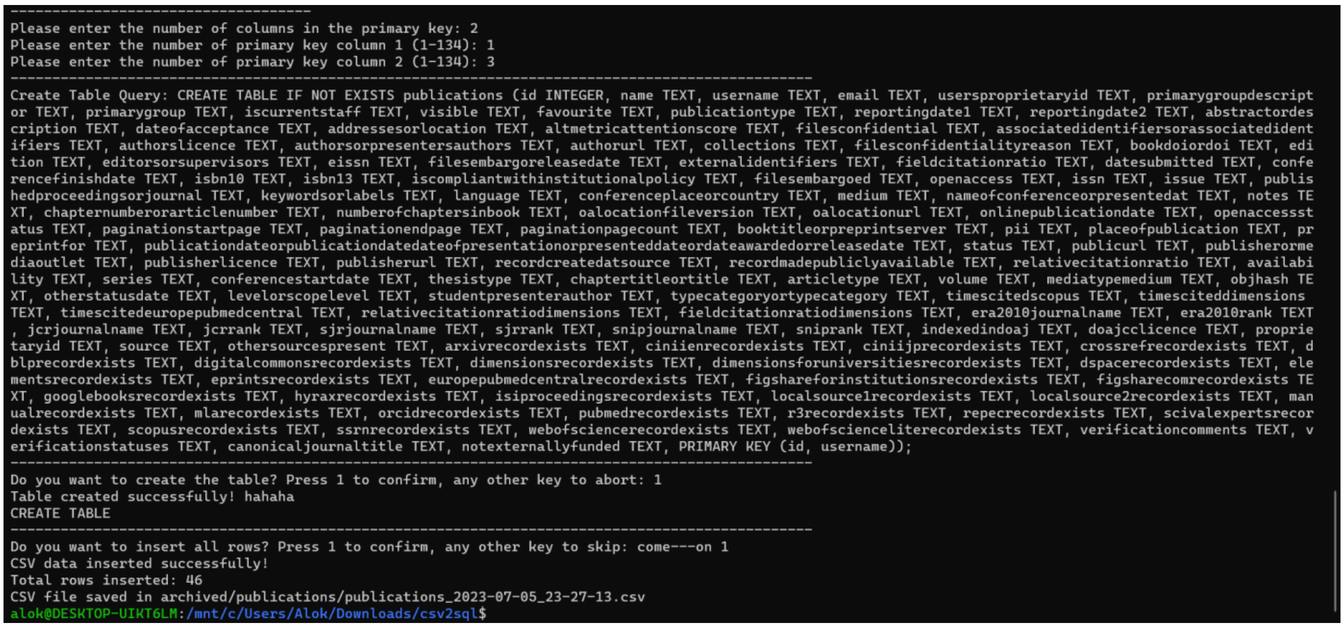
```
alok@DESKTOP-UIKT6LM:/mnt/c/Users/Alok/Downloads/csv2sql$ python3 csv2sql.py Publications_UserObjectPairs_From20230501_To20220501_350000000\ SHP\ Dean\'s\ O
ffice_20230503.csv
File 'publications_userobjectpairs_from20230501_to20220501_350000000 shp dean's office_20230503.csv' converted to CSV and saved as 'publications_userobjectp
airs_from20230501_to20220501_350000000 shp dean's office_20230503.csv'.
Updated file saved as publications_userobjectpairs_from20230501_to20220501_350000000 shp dean's office_20230503.csv.
------------------------------------------------------------------------------------------------
Duplicate rows found:
        id                  name  username  ... verificationstatuses canonicaljournaltitle notexternallyfunded
12  619721    PAUSTIAN, Pamela E  paustian  ...           Unverified                   NaN                  No
26  175629      BERTRAND, Brenda  brendamb  ...           Unverified                   NaN                  No
35  611160  BARSTOW, Elizabeth A  bbarstow  ...           Unverified                   NaN                  No

[3 rows x 134 columns]
------------------------------------------------------------------------------------------------
Do you want to delete these duplicate rows? Press 1 to confirm, any other key to skip: 1
Duplicate rows removed and file updated successfully.
CSV file 'publications_userobjectpairs_from20230501_to20220501_350000000 shp dean's office_20230503.csv' processed successfully and saved as 'processed/publ
ications_userobjectpairs_from20230501_to20220501_350000000 shp dean's office_20230503/publications_userobjectpairs_from20230501_to20220501_350000000 shp dea
n's office_20230503_2023-07-05_23-17-37.csv'.
------------------------------------------------------------------------------------------------
[Errno 2] No such file or directory: "archived/publications_userobjectpairs_from20230501_to20220501_350000000 shp dean's office_20230503"
There is no Archive file : True
The headers of both archive file and processed file are matching: False
------------------------------------------------------------------------------------------------
Processed folder 'processed/publications_userobjectpairs_from20230501_to20220501_350000000 shp dean's office_20230503'
Most recently processed csv files
["publications_userobjectpairs_from20230501_to20220501_350000000 shp dean's office_20230503_2023-07-05_23-17-37.csv"]
```

steps 1 to steps 7



```
------------------------------------------------------------------------------------------------
processed/publications_userobjectpairs_from20230501_to20220501_350000000 shp dean's office_20230503/publications_userobjectpairs_from20230501_to20220501_350
000000 shp dean's office_20230503_2023-07-05_23-17-37.csv
Table NAME :
publications
------------------------------------
Column # | Column Name | Data Type
-------- + ------------+ ----------
1        | id          | INTEGER
2        | name        | TEXT
3        | username    | TEXT
4        | email       | TEXT
5        | usersproprietaryid | TEXT
6        | primarygroupdescriptor | TEXT
7        | primarygroup | TEXT
8        | iscurrentstaff | TEXT
9        | visible     | TEXT
10       | favourite   | TEXT
11       | publicationtype | TEXT
12       | reportingdate1 | TEXT
13       | reportingdate2 | TEXT
14       | abstractordescription | TEXT
15       | dateofacceptance | TEXT
16       | addressesorlocation | TEXT
17       | altmetricattentionscore | TEXT
18       | filesconfidential | TEXT
19       | associatedidentifiersorassociatedidentifiers | TEXT
20       | authorslicence | TEXT
21       | authorsorpresentersauthors | TEXT
22       | authorurl   | TEXT
23       | collections | TEXT
24       | filesconfidentialityreason | TEXT
25       | bookdoiordoi | TEXT
26       | edition     | TEXT
27       | editorsorsupervisors | TEXT
28       | eissn       | TEXT
29       | filesembargoreleasedate | TEXT
30       | externalidentifiers | TEXT
```

Step 8



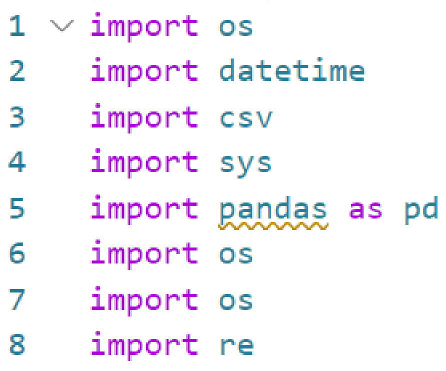
```
------------------------------------
Please enter the number of columns in the primary key: 2
Please enter the number of primary key column 1 (1-134): 1
Please enter the number of primary key column 2 (1-134): 3
------------------------------------------------------------------------------------------
Create Table Query: CREATE TABLE IF NOT EXISTS publications (id INTEGER, name TEXT, username TEXT, email TEXT, usersproprietaryid TEXT, primarygroupdescript
or TEXT, primarygroup TEXT, iscurrentstaff TEXT, visible TEXT, favourite TEXT, publicationtype TEXT, reportingdate1 TEXT, reportingdate2 TEXT, abstractordes
cription TEXT, dateofacceptance TEXT, addressesorlocation TEXT, altmetricattentionscore TEXT, filesconfidential TEXT, associatedidentifiersorassociatedident
ifiers TEXT, authorslicence TEXT, authorsorpresentersauthors TEXT, authorurl TEXT, collections TEXT, filesconfidentialityreason TEXT, bookdoiordoi TEXT, edi
tion TEXT, editorsorsupervisors TEXT, eissn TEXT, filesembargoreleasedate TEXT, externalidentifiers TEXT, fieldcitationratio TEXT, datesubmitted TEXT, confe
rencefinishdate TEXT, isbn10 TEXT, isbn13 TEXT, iscompliantwithinstitutionalpolicy TEXT, filesembargoed TEXT, openaccess TEXT, issn TEXT, issue TEXT, publis
hedproceedingsorjournal TEXT, keywordsorlabels TEXT, language TEXT, conferenceplaceorcountry TEXT, medium TEXT, nameofconferenceorpresentedat TEXT, notes TE
XT, chapternumberorarticlenumber TEXT, numberofchaptersinbook TEXT, oalocationfileversion TEXT, oalocationurl TEXT, onlinepublicationdate TEXT, openaccessst
atus TEXT, paginationstartpage TEXT, paginationendpage TEXT, paginationpagecount TEXT, booktitleorpreprintserver TEXT, pii TEXT, placeofpublication TEXT, pr
eprintfor TEXT, publicationdateorpublicationdatedateofpresentationorpresenteddateordateawardedorreleasedate TEXT, status TEXT, publicurl TEXT, publisherorme
diaoutlet TEXT, publisherlicence TEXT, publisherurl TEXT, recordcreatedatsource TEXT, recordmadepubliclyavailable TEXT, relativecitationratio TEXT, availabi
lity TEXT, series TEXT, conferencestartdate TEXT, thesistype TEXT, chaptertitleortitle TEXT, articletype TEXT, volume TEXT, mediatypemedium TEXT, objhash TE
XT, otherstatusdate TEXT, levelorscopelevel TEXT, studentpresenterauthor TEXT, typecategoryortypecategory TEXT, timescitedscopus TEXT, timesciteddimensions 
TEXT, timescitedeuropepubmedcentral TEXT, relativecitationratiodimensions TEXT, fieldcitationratiodimensions TEXT, era2010journalname TEXT, era2010rank TEXT
, jcrjournalname TEXT, jcrrank TEXT, sjrjournalname TEXT, sjrrank TEXT, snipjournalname TEXT, sniprank TEXT, indexedindoaj TEXT, doajcclicence TEXT, proprie
taryid TEXT, source TEXT, othersourcespresent TEXT, arxivrecordexists TEXT, ciniienrecordexists TEXT, ciniijprecordexists TEXT, crossrefrecordexists TEXT, d
blprecordexists TEXT, digitalcommonsrecordexists TEXT, dimensionsrecordexists TEXT, dimensionsforuniversitiesrecordexists TEXT, dspacerecordexists TEXT, ele
mentsrecordexists TEXT, eprintsrecordexists TEXT, europepubmedcentralrecordexists TEXT, figshareforinstitutionsrecordexists TEXT, figsharecomrecordexists TE
XT, googlebooksrecordexists TEXT, hyraxrecordexists TEXT, isiproceedingsrecordexists TEXT, localsource1recordexists TEXT, localsource2recordexists TEXT, man
ualrecordexists TEXT, mlarecordexists TEXT, orcidrecordexists TEXT, pubmedrecordexists TEXT, r3recordexists TEXT, repecrecordexists TEXT, scivalexpertsrecor
dexists TEXT, scopusrecordexists TEXT, ssrnrecordexists TEXT, webofsciencerecordexists TEXT, webofsciencelitерecordexists TEXT, verificationcomments TEXT, v
erificationstatuses TEXT, canonicaljournaltitle TEXT, notexternallyfunded TEXT, PRIMARY KEY (id, username));
------------------------------------------------------------------------------------------
Do you want to create the table? Press 1 to confirm, any other key to abort: 1
Table created successfully! hahaha
CREATE TABLE
------------------------------------------------------------------------------------------
Do you want to insert all rows? Press 1 to confirm, any other key to skip: come---on 1
CSV data inserted successfully!
Total rows inserted: 46
CSV file saved in archived/publications/publications_2023-07-05_23-27-13.csv
alok@DESKTOP-UIKT6LM:/mnt/c/Users/Alok/Downloads/csv2sql$
```

steps 9 to steps 11

Code Explanation

The csv2sql.py file performs various operations on a CSV file.

Importing necessary modules:

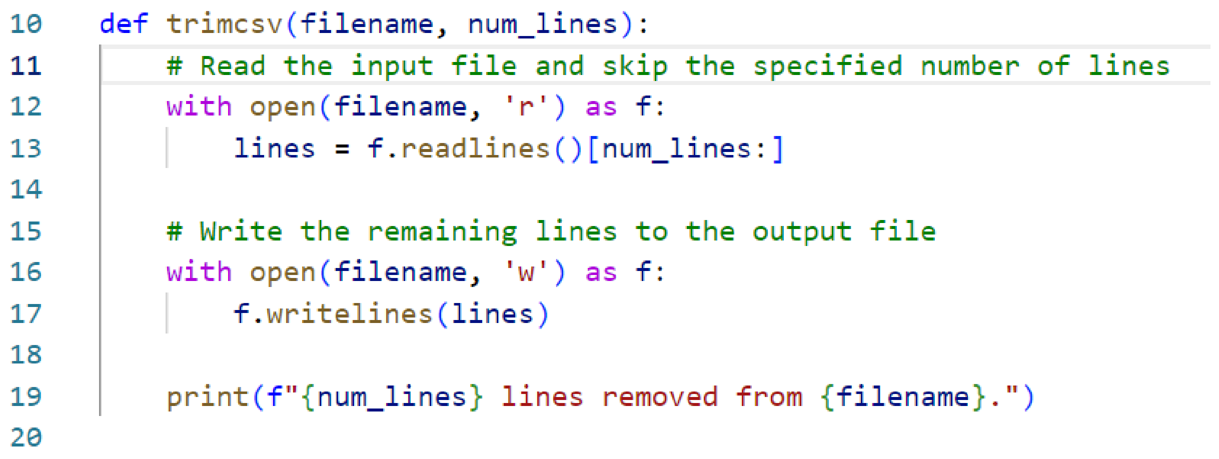


```python
import os
import datetime
import csv
import sys
import pandas as pd
import os
import os
import re
```

This section imports the required modules for file manipulation, date and time operations, CSV handling, system-related functions, and regular expressions.

Defining the trimcsv function:

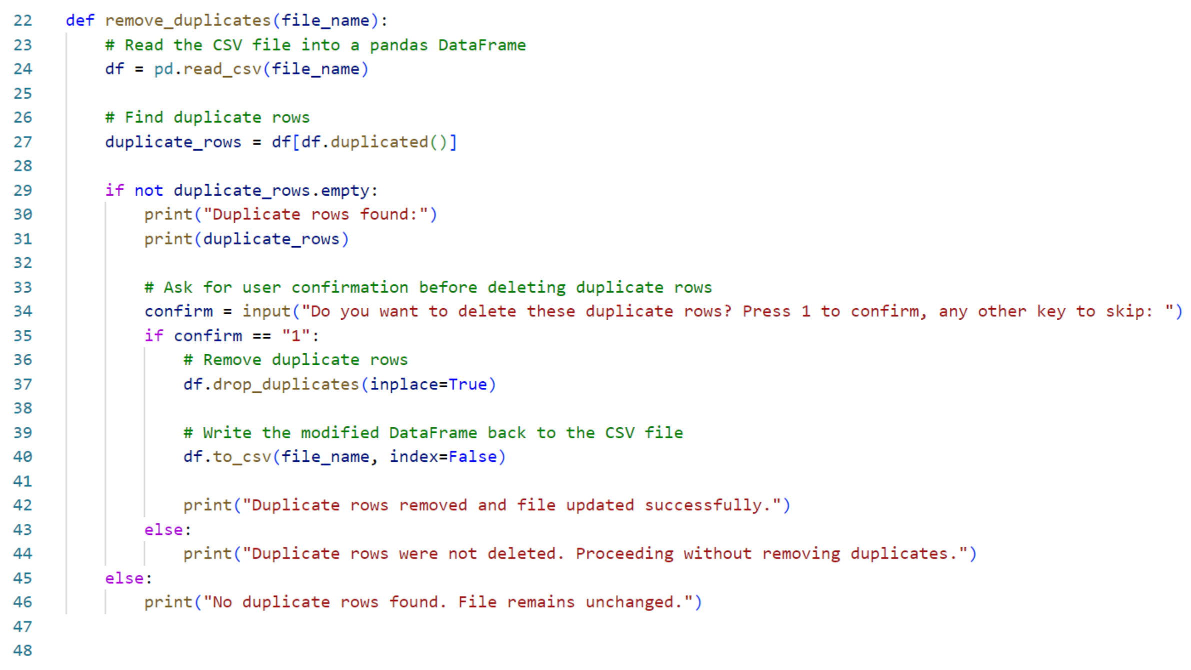
This function reads a file and removes a specified number of lines from the beginning. It then writes the remaining lines back to the same file.



```python
def trimcsv(filename, num_lines):
    # Read the input file and skip the specified number of lines
    with open(filename, 'r') as f:
        lines = f.readlines()[num_lines:]

    # Write the remaining lines to the output file
    with open(filename, 'w') as f:
        f.writelines(lines)

    print(f"{num_lines} lines removed from {filename}.")

```

Defining the remove_duplicates function:

This function uses the pandas library to read a CSV file into a DataFrame. It identifies duplicate rows in the DataFrame, displays them, and prompts the user to confirm whether they want to delete those rows. If confirmed, the duplicate rows are removed, and the modified DataFrame is saved back to the CSV file.



```python
def remove_duplicates(file_name):
    # Read the CSV file into a pandas DataFrame
    df = pd.read_csv(file_name)

    # Find duplicate rows
    duplicate_rows = df[df.duplicated()]

    if not duplicate_rows.empty:
        print("Duplicate rows found:")
        print(duplicate_rows)

        # Ask for user confirmation before deleting duplicate rows
        confirm = input("Do you want to delete these duplicate rows? Press 1 to confirm, any other key to skip: ")
        if confirm == "1":
            # Remove duplicate rows
            df.drop_duplicates(inplace=True)

            # Write the modified DataFrame back to the CSV file
            df.to_csv(file_name, index=False)

            print("Duplicate rows removed and file updated successfully.")
        else:
            print("Duplicate rows were not deleted. Proceeding without removing duplicates.")
    else:
        print("No duplicate rows found. File remains unchanged.")


```

Defining the increment_duplicate_columns function:

```python
def increment_duplicate_columns(csv_filename):
```

This function reads a CSV file and performs the following operations:

- Reads the file and stores the data rows in a list.
- Retrieves the header row.
- Counts the occurrences of each processed column name by removing non-alphanumeric characters and extra spaces.
- Modifies column names to make them unique by appending a number to duplicates.
- Writes the modified data (including the updated header row) back to the CSV file.

Defining the process_csv_file function:

```python
def process_csv_file(filename, n=4):
    # Check if file exists
    if not os.path.isfile(filename):
        print(f"Error: File '{filename}' does not exist")
        return
    # Rest of the function...
```

This function processes a CSV file based on the provided filename. It takes an optional argument n (defaulted to 4) which represents the number of commas to consider for data processing.

The function performs the following steps:

- Reads the input file to find the maximum number of columns (max_columns) and the rows with that count (max_rows).
- Generates a timestamp for the processed file name.
- Creates a directory structure for the processed files.
- Creates the processed filename based on the original filename and the timestamp.
- Writes a new CSV file with cleaned rows (removes extra spaces and double quotes) based on the specified n and max_columns.
- Reads the new file, cleans the first row (header row) by removing extra spaces, symbols, and converting to lowercase.
- Writes the modified data back to the CSV file.
- Calls the increment_duplicate_columns function to handle duplicate column names.
- Calls the remove_duplicates function to remove duplicate rows.
- Prints a confirmation message.

Handling command-line arguments:

```
if len(sys.argv) < 2:
    print("Error: Please provide the CSV filename as an argument.")
    sys.exit(1)

filename = sys.argv[1]
n = 5  # Default value for n if not provided in sys arguments
num_lines = 0  # Default value for num_lines if not provided in sys arguments
```

```
if len(sys.argv) >= 3:
   n = int(sys.argv[2])

if len(sys.argv) >= 4:
   num_lines = int(sys.argv[3])
```

This section handles command-line arguments. The script expects at least one argument, which is the CSV filename. The optional second argument sets the value of n, representing the number of commas to consider for processing. The optional third argument sets the number of lines to remove using the trimcsv function.

Calling the filter_csv_rows function:

```
def filter_csv_rows(filename, output_filename, threshold):
```

This function filters rows in a CSV file based on the threshold of non-null fields percentage. It reads the input file, checks each row's non-null fields percentage, and either includes or skips the row based on the threshold.

Calling the create_table function:

We first get the most recent processed file which is used for table creation. We use psycopgs connection object to connect to database.

The connection object needs database name, username, password and port number.

```
conn = psycopg2.connect(database="meddb", user="alok",
password="reddy", host="localhost", port="5432")
```

Since the table columns shouldn’t start with numbers. We use a regex and key mappings for ordinal numbers. They will be replaced with text.

```
def replace_ordinal_numbers(column_name):
    # Replace ordinal numbers
    ordinal_mapping = {
        "1st": "first",
        "2nd": "second",
        "3rd": "third",
        "4th": "fourth",
        "5th": "fifth",
        "6th": "sixth",
        "7th": "seventh",
        "8th": "eighth",
        "9th": "ninth"
    }

     # Check if the first character is a digit and then replace
numbers without ordinal indicators
    if column_name[0].isdigit():
        column_name = re.sub(r'^(\d+)([a-zA-Z]+)', lambda m:
{**{"1": "first", "2": "second", "3": "third", "4": "fourth", "5":
"fifth", "6": "sixth", "7": "seventh", "8": "eighth", "9":
"ninth"}, **ordinal_mapping}[m.group(1)] + m.group(2), column_name)
    else:
        # Check if the first three letters match an ordinal mapping
and replace them
        first_three_letters = column_name[:3]
        if first_three_letters in ordinal_mapping:
            column_name = ordinal_mapping[first_three_letters] +
column_name[3:]

    return column_name
```

Defining the data types of the columns:
We use the first 100 rows in the csv file to determine the data type of the column. It can be Decimal, Integer, Text.

```
data_types = get_column_data_types(headers, csvreader)
```

This information will also be displayed in proper format and the user will be asked for primary key selection.

```
print_column_data_types(headers, data_types)

# Determine the primary key columns
pk_columns = get_primary_key_columns(headers)
```

Defining the Create table query:

```
create_table_query = f"CREATE TABLE IF NOT EXISTS {table_name} ({',
'.join([header.replace('(', '').replace(')', '') + ' ' +
data_types[i] for i, header in enumerate(headers)])}, PRIMARY KEY
({pk_constraint}));"
```

The information like primary key constraints and column names which are set in previous steps will be used for create table query creation and then will be executed.

```
cur.execute(create_table_query)
```

Insert data into table:

```
insert_data_into_table(cur, conn, processed_filepath, headers,
data_types, table_name)
```

We call the function insert data to push the data from csv to the table created.

```
         insert_query = sql.SQL("INSERT INTO {table} ({columns})
VALUES %s ON CONFLICT DO NOTHING;").format(
            table=sql.Identifier(table_name),
            columns=sql.SQL(', ').join(map(sql.Identifier,
headers_lower))
        )
```

The insert query is created and will be executed as per user confirmation.

After successful insertion the data from the table will be used to create an archive file with the same file name.

```
                  # Create archived folder if it doesn't exist
                if not os.path.exists('archived'):
                    os.makedirs('archived')

                # Create a folder with the name of the CSV file in
the archived folder if it doesn't exist
                foldername =
os.path.splitext(os.path.basename(filename))[0]
```

```
                foldername = foldername.split('_')[0]
                if not os.path.exists(f'archived/{foldername}'):
                    os.makedirs(f'archived/{foldername}')


                # Write the SQL table to a CSV file in the archived
folder
                with
open(f'archived/{foldername}/{table_name}_{timestamp}.csv', 'w') as
csvfile:
                    csvwriter = csv.writer(csvfile)
                    csvwriter.writerow(headers_lower)  # Write
lower case column names
                    cur.execute(f"SELECT * FROM {table_name}")
                    rows = cur.fetchall()
                    for row in rows:
                        csvwriter.writerow(row)
```

Call update function if archive file is not empty:

We create a query from the csv and execute as per user confirmation.

```
            query = sql.SQL("INSERT INTO {table} ({columns})
VALUES ({values}) ON CONFLICT ({conflict_columns}) DO NOTHING
RETURNING *;").format(
                table=sql.Identifier(table_name),
                columns=sql.SQL(', ').join(map(sql.Identifier,
headers)),
                values=sql.SQL(', ').join([sql.Placeholder()] *
len(headers)),
                conflict_columns=sql.SQL(',
').join(map(sql.Identifier, pk_columns))
            )
```

Checking for any conflicting records:

If there any conflicting records we add them to a separate list and user will be asked for confirmation to push them to database.

```
        if conflicting_records:
           print(f"{len(conflicting_records)} conflicting
records:")
           for record in conflicting_records:
               print(record)
```

```
            print(f"{len(conflicting_records)} conflicting
records:")
            # Ask for confirmation before pushing conflicting
records
            confirmation = input("Do you want to push the
conflicting records to the database? (yes/no): ")
            if confirmation.lower() == "yes":
                # Push conflicting records to the database
                for record in conflicting_records:
                    try:
                        cur.execute(query, record)
                        rows_inserted += 1
                    except psycopg2.IntegrityError:
                        print(f"Failed to insert conflicting
record: {record}")
                    else:
                        rows_updated += 1
                print("Conflicting records inserted.")
            else:
                print("Conflicting records not pushed to the
database.")
```

We use the same logic as in insert function to save the updated or pushed file to the archived folder.

ERROR Handling:

1)Error while updating the Table:

The following error can occur when the archived folder of the specific file is not empty. This means the table has already been created but deleted later.

Error can occur because of two scenarios.

1. Table is not present in Database, but we are trying to push or update the table.
2. There is an error while processing the csv file.

```
processed/enrollment/enrollment_2023-07-12_19-08-29.csv

Archive file is not empty. Table updation is done

The primary columns are :

[]

Error occurred: syntax error at or near ")"

LINE 1: ...53_level_1', 'Unnamed: 154_level_1') ON CONFLICT () DO NOTHI...
```

Example of scenario 1

How to debug more

We can connect to the sql db and check if table exists.

1. Connect to postgresql db
   psql -d dbname -U username -W -h hostname -p portnumber
   psql -d meddb -U alok -W -h localhost -p 5432
   password: reddy

2. Check the existing tables in the db
   \d
   The above command shows all the existing tables

3. Check specific table (In our case we are checking the specific table enrollment)
   \d tablename
   \d enrollment

How to fix the problem from local:

A simple fix for the above issue is to delete the local files for the specific table in archived folder. This means we are telling the script there is no table created earlier for the specific table.

Move and remove the files pushed earlier in the archived folder.

1. cd archived
2. Remove the files of the specific folder
   rm -r tablename/
   rm -r enrollment/
   Here we are removing the folder enrolment recursively so that the script can start with table creation and insertion. (Since we found out there is no table in early steps, the archive file needs to empty so that the data inserting starts with creation).

Now we can rerun the python csv2sql.py enrolment.xls command and everything works fine.