Explanation of CSV to SQL Conversion Code

Imports:

- import csv : Reads CSV files.
- import re : Cleans strings using regex.
- from datetime import datetime : Parses dates.

Why: Needed for CSV processing, cleaning names, and type inference.

Function: infer_sql_type(value)

- def infer_sql_type(value) : Defines function to guess SQL type.
- if value is None `or value.strip() == '' : return 'VARCHAR(255)': Empty/None → VARCHAR(255).
- try: int(value) ... return 'INTEGER' : Converts to int → INTEGER.
- except ValueError: pass :Ignores int conversion failure.
- try: float(value) ... return 'FLOAT' : Converts to float → FLOAT.
- except ValueError: pass : Ignores float conversion failure.
- try: datetime.strptime(value, '%Y-%m-%d') ... return 'DATE' : Parses YYYY-MM-DD → DATE.
- except ValueError: try: datetime.strptime(value, '%d/%m/%Y') ... return 'DATE' : Parses DD/MM/YYYY → DATE.
- except ValueError: pass : Ignores date parsing failure.
- return f'VARCHAR({min(255, len(value) + 10)})' : Defaults to VARCHAR with dynamic length.

Why: Infers SQL data types for columns.

Function: sanitize_name(name)

- def sanitize_name(name) : Defines function to clean column names.
- name = re.sub(r'[^a-zA-Z0-9\s]', '', name).strip() : Removes special chars, keeps letters/numbers/spaces, trims whitespace.
- name = re.sub(r'\s+', '_', name) : Replaces spaces with underscore.
- if name and name[0].isdigit(): name = f'_{name}' : Adds _ if name starts with digit.
- return name or 'column' : Returns cleaned name or 'column' if empty.

Why: Makes column names SQL-compatible.

Function: **csv_to_sql (csv_file_path, table_name='#')**

- def csv_to_sql(...) : Defines function to convert CSV to SQL.
- Try : Starts error handling.
- with open(csv_file_path, 'r', encoding='utf-8') as csv_file : Opens CSV file.
- csv_reader = csv.reader(csv_file) : Creates CSV reader.
- headers = next(csv_reader) : Reads first row as headers.
- sanitized_headers = [sanitize_name(header) ...] : Sanitizes headers.
- first_row = next(csv_reader, None) : Reads first data row or None.
- column_types = [] : Initializes list for data types.
- if first_row: column_types = [infer_sql_type(value) ...] : Infers types from first row.
- else: column_types = ['VARCHAR(255)' ...] : Defaults to VARCHAR(255) if no data.
- columns_def = [f'"{col}" {col_type}' ...] : Pairs column names with types.
- create_table = f'CREATE TABLE {table_name} (\n ' : Starts CREATE TABLE.
- create_table += ',\n '.join(columns_def) : Adds column definitions.
- create_table += '\n);' : Closes CREATE TABLE.
- insert_statements = [] : Initializes list for INSERT statements.
- if first_row: : Checks if data exists.
- values = ['NULL' if val.strip() == '' else f"'{val.replace('\'', '\'\'')}'" ...] : Formats first row: empty → NULL, else quoted.
- insert = f'INSERT INTO {table_name} ... : Starts INSERT with column names.
- insert += f'VALUES ({", ".join(values)});' : Adds values to INSERT.
- insert_statements.append(insert) : Stores first INSERT.
- for row in csv_reader: : Loops through remaining rows.
- values = ['NULL' if val.strip() == '' else ...] : Formats row values.
- insert = f'INSERT INTO {table_name} ... : Starts INSERT for row.
- insert += f'VALUES ({", ".join(values)});' : Adds row values.
- insert_statements.append(insert) : Stores INSERT.
- sql_output = [create_table] : Initializes output with CREATE TABLE.
- if insert_statements: sql_output.append('\n-- Insert statements') : Adds comment if INSERTs exist.
- sql_output.extend(insert_statements) : Adds all INSERTs.
- return '\n'.join(sql_output) : Joins statements into single string.
- except FileNotFoundError: ... : Returns error if file not found.
- except Exception as e: ... : Returns error for other issues.

Why: Converts CSV to SQL CREATE and INSERT statements.

Function: main()

- def main() : Defines main function.
- csv_file = r"C:\Users\Tahseen Ashrafi\Downloads\CSV\Sample_data.csv" : Sets CSV path.
- table_name = '#' : Sets default table name.
- sql_statements = csv_to_sql(...) : Generates SQL.
- print("\nGenerated SQL Statements:\n") : Prints header.
- print(sql_statements) : Prints SQL.
- save = input("\nSave to SQL file? (y/n): ").lower() == 'y' : Asks to save; checks for 'y'.
- if save: : Checks if saving.
- output_file = f"{table_name}.sql" : Sets output file name.
- with open(output_file, 'w', encoding='utf-8') as f : Opens file for writing.
- f.write(sql_statements) : Writes SQL to file.
- print(f"SQL statements saved to {output_file}") : Confirms save.

Why: Runs conversion, displays output, and saves to file if requested.

Entry Point:

- if __name__ == "__main__": main() : Runs main() if script is executed directly.

Why: Ensures program runs only when script is run, not imported.