Assignment 3: Identify issues with dataset

Submitted to: Amad Mumtaz
Submitted by: Huzaifa Munir
Dated: 05th November 2024

Task: Identify issues with dataset

PFA Customers data. This has some updated values in the rows.

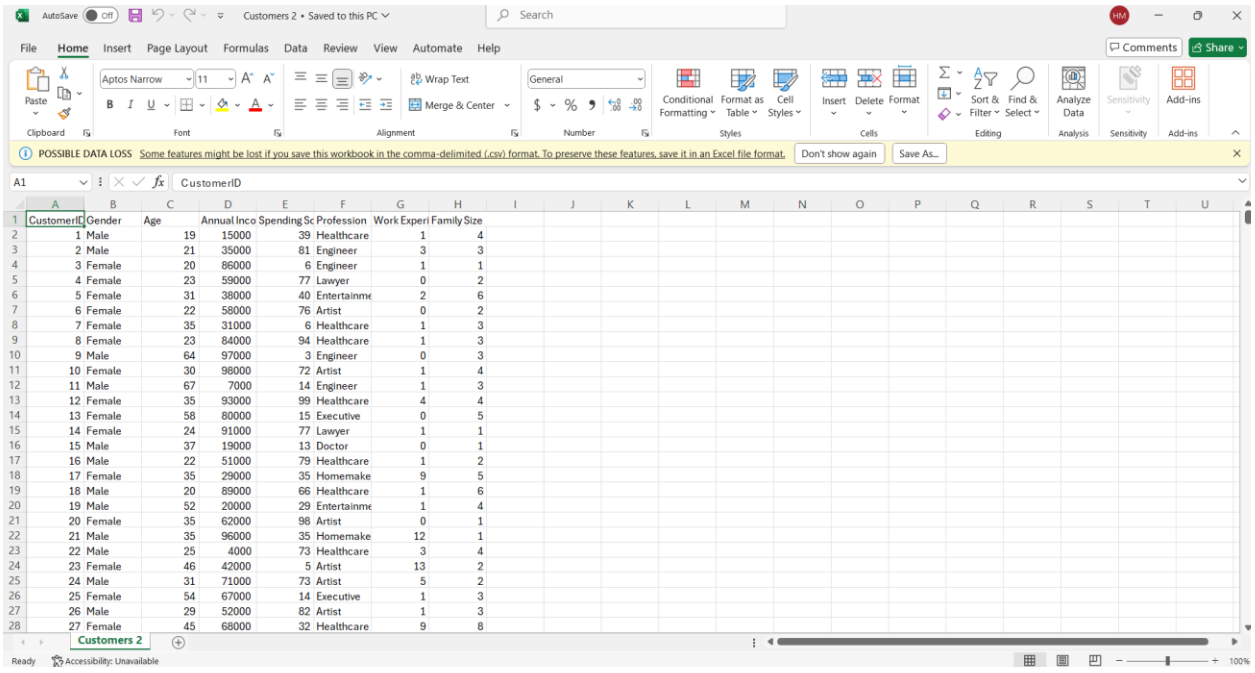
Considerations:

- Consider this data as inserted in an application's database -> customer’s table.
- The application has a form (UI) which loads a data table including all values fetched from this customers table (having attached/ below dataset)
- During the page/ data table loading, the page does not provide complete values i.e. fails to load.
- This could be a result of faulty dataset inserted in the database table.
- Some points:
    - Customers form fetches and displays data in the web app over following fields:
    - CustomerID (int primary key)
    - Customer name (varchar) (assume coming from another referenced table)
    - Gender (varchar)
    - Age (int)
    - Annual income (int)
    - Now the form fails to load properly as the you might be missing an implementation which caters duplicate records, float displaying over an int field, or a missing value NULL (could be any error occurring in the application).
    - As a QA Engineer, you task is to explore this dataset and find what could be the values/ rows/ records which are causing these problems.
        - Example, a duplicate record in id column
        - A missing value in any other column i.e. NULL such that there is no condition in your code to cater this.
        - varchar being stored in int field

Load this dataset in your database and create new table. Run queries to know your dataset and highlight any problems which you think is impacting your application.

Prepare a word file and show what errors/ discrepancies you have observed in the dataset. Provide reasons to why this could be causing any issues. Submit your solutions in the shared folder.

Provided Dataset:



	A	B	C	D	E	F	G	H
1	CustomerID	Gender	Age	Annual Inco	Spending Sc	Profession	Work Experi	Family Size
2	1	Male	19	15000	39	Healthcare	1	4
3	2	Male	21	35000	81	Engineer	3	3
4	3	Female	20	86000	6	Engineer	1	1
5	4	Female	23	59000	77	Lawyer	0	2
6	5	Female	31	38000	40	Entertainme	2	6
7	6	Female	22	58000	76	Artist	0	2
8	7	Female	35	31000	6	Healthcare	1	3
9	8	Female	23	84000	94	Healthcare	1	3
10	9	Male	64	97000	3	Engineer	0	3
11	10	Female	30	98000	72	Artist	1	4
12	11	Male	67	7000	14	Engineer	1	3
13	12	Female	35	93000	99	Healthcare	4	4
14	13	Female	58	80000	15	Executive	0	5
15	14	Female	24	91000	77	Lawyer	1	1
16	15	Male	37	19000	13	Doctor	0	1
17	16	Male	22	51000	79	Healthcare	1	2
18	17	Female	35	29000	35	Homemake	9	5
19	18	Male	20	89000	66	Healthcare	1	6
20	19	Male	52	20000	29	Entertainme	1	4
21	20	Female	35	62000	98	Artist	0	1
22	21	Male	35	96000	35	Homemake	12	1
23	22	Male	25	4000	73	Healthcare	3	4
24	23	Female	46	42000	5	Artist	13	2
25	24	Male	31	71000	73	Artist	5	2
26	25	Female	54	67000	14	Executive	1	3
27	26	Male	29	52000	82	Artist	1	3
28	27	Female	45	68000	32	Healthcare	9	8

Customers 2 • Saved to this PC
POSSIBLE DATA LOSS Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.
Don't show again
Save As...
A1
CustomerID
Customers 2
Ready
Accessibility: Unavailable

We are provided with the customers database file In which there are issues and we have to identify those issues.

Solution:

1. Setting up the SQLite Command Line Interface and its Environment:

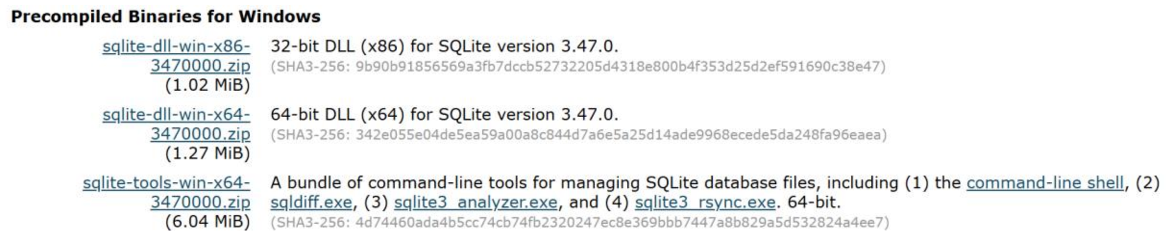
SQLite is a lightweight serverless database that is easy to use for small scale tasks.

a) Step1: Installation

i) Install SQLite from the web if it is not installed on your PC.

ii) Go to the official SQLite website and according to your operating system select the respective option.

iii) Under the **Precompiled Binaries for Windows** section, download the zip file for sqlite-tools-win-x64-3470000.zip (3rd one).



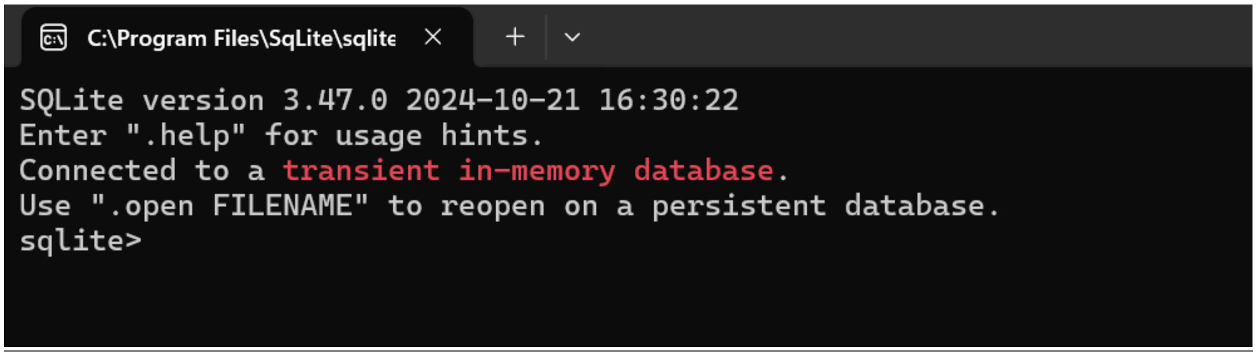
Precompiled Binaries for Windows

sqlite-dll-win-x86-3470000.zip (1.02 MiB)	32-bit DLL (x86) for SQLite version 3.47.0. (SHA3-256: 9b90b91856569a3fb7dccb52732205d4318e800b4f353d25d2ef591690c38e47)
sqlite-dll-win-x64-3470000.zip (1.27 MiB)	64-bit DLL (x64) for SQLite version 3.47.0. (SHA3-256: 342e055e04de5ea59a00a8c844d7a6e5a25d14ade9968ecede5da248fa96eaea)
sqlite-tools-win-x64-3470000.zip (6.04 MiB)	A bundle of command-line tools for managing SQLite database files, including (1) the command-line shell, (2) sqldiff.exe, (3) sqlite3_analyzer.exe, and (4) sqlite3_rsync.exe. 64-bit. (SHA3-256: 4d74460ada4b5cc74cb74fb2320247ec8e369bbb7447a8b829a5d532824a4ee7)

iv) After downloading it, now let's set up SQLite. Create a folder by the name of SQLite in the program files (recommended, not necessary) of your C:/ drive and then extract the zip file to this folder.

v) Open the folder where you extracted the files. Here, you will see 4 files: sqlite3_rsync.exe, sqlite3_analyzer.exe, sqlite3.exe, sqlite3.exe.

Double Click to open the sqlite3.exe file and it will open an SQLite command line interface.



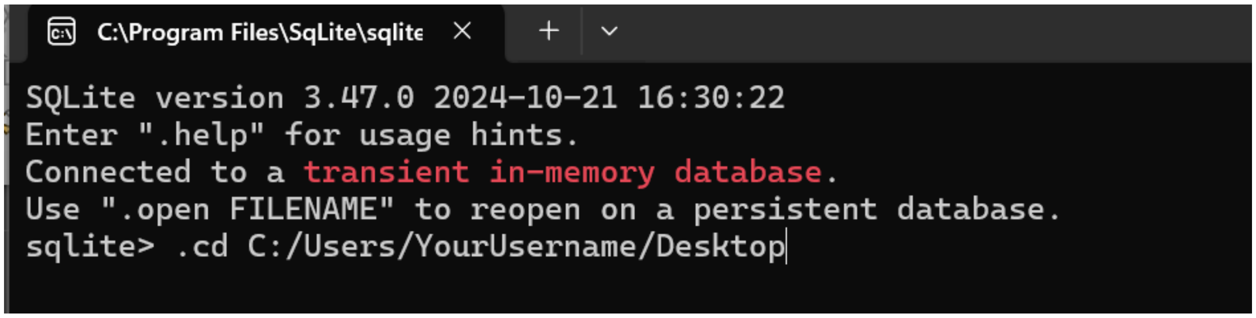
```
SQLite version 3.47.0 2024-10-21 16:30:22
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

Note: The above work is for Windows. Kindly, check your OS requirements before downloading and setting.

b) Step2: Database Creation

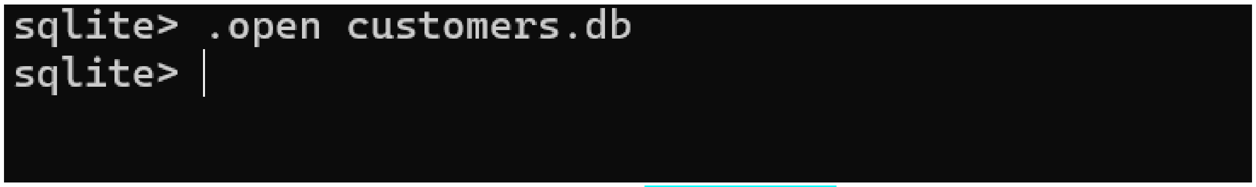
Now create a persistent database.

- To do this, first make sure you are in the right folder where you have full permissions.
- Firstly, change the directory to **Desktop.** For this, type the following command in the SQLite command line interface.



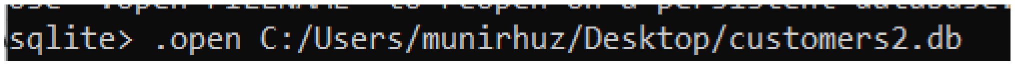
```
SQLite version 3.47.0 2024-10-21 16:30:22
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .cd C:/Users/YourUsername/Desktop
```

- Now, create the database by typing the following command.



```
sqlite> .open customers.db
sqlite>
```

- Once the command runs successfully, a file called customers2.db should appear on your **Desktop** or the specified directory. You can name the file anything you like.
- You can also perform the same task with the following command.



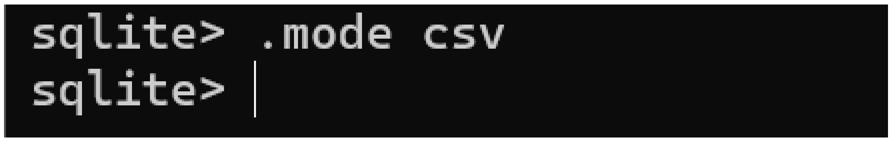
```
sqlite> .open C:/Users/munirhuz/Desktop/customers2.db
```

Note: Make sure that you are in a directory where you have permission to create files. The simplest way to avoid this issue is to navigate to a folder like your **Desktop** or **Documents** where you have full access.

c) **Step3: Importing the CSV File**

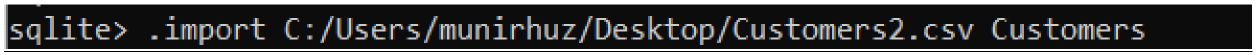
Now, import the CSV file.

- To perform this task, first set the mode to CSV by following command.



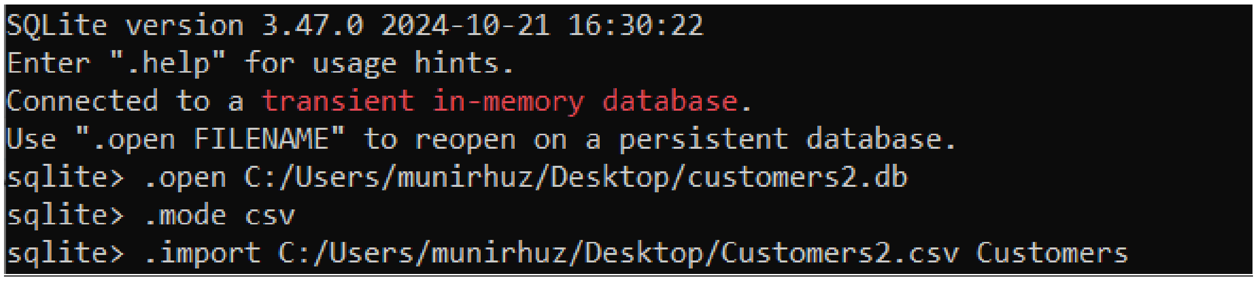
```
sqlite> .mode csv
sqlite>
```

- Now, import your CSV file. (Since, I kept it in my Desktop, you can do the same or choose your preferred directory.)



```
sqlite> .import C:/Users/munirhuz/Desktop/Customers2.csv Customers
```

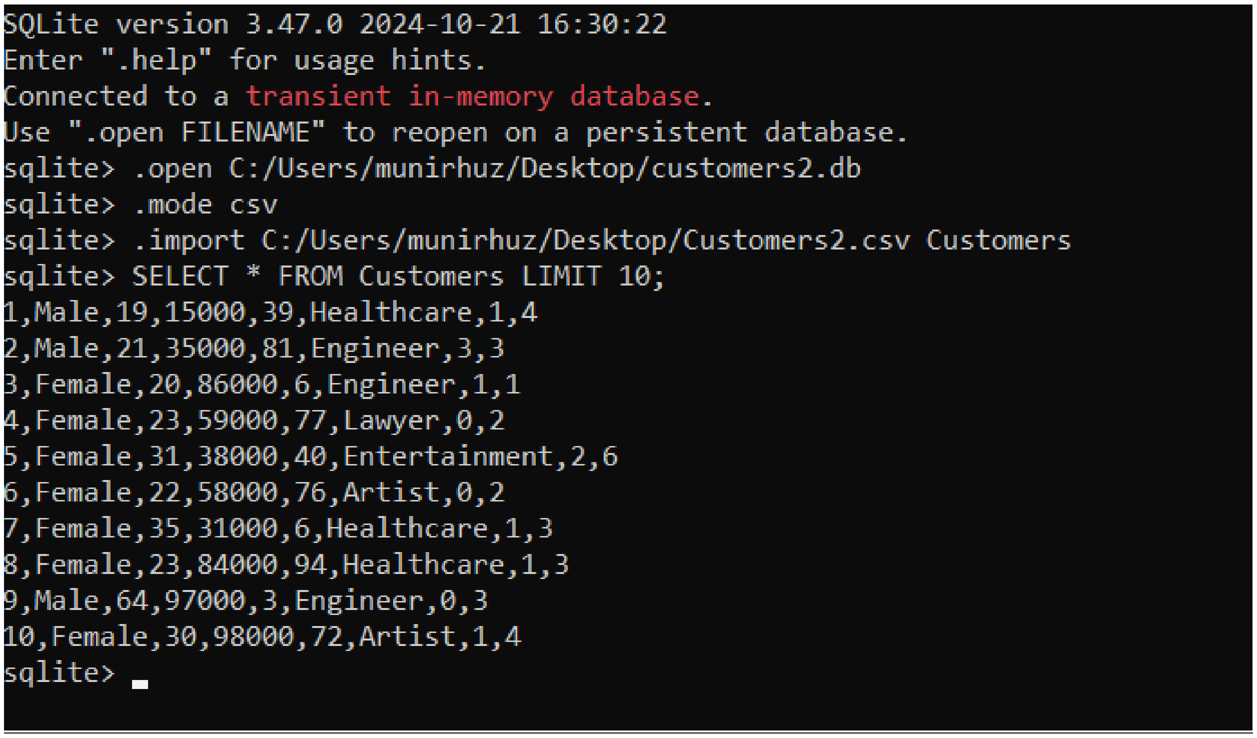
- The CSV is imported into a **table** (in this case, named Customers) inside your SQLite database (customers.db). The original CSV file (Customers1.csv) itself remains unchanged, but its data is copied into the SQLite table (Customers).



```
SQLite version 3.47.0 2024-10-21 16:30:22
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open C:/Users/munirhuz/Desktop/customers2.db
sqlite> .mode csv
sqlite> .import C:/Users/munirhuz/Desktop/Customers2.csv Customers
```

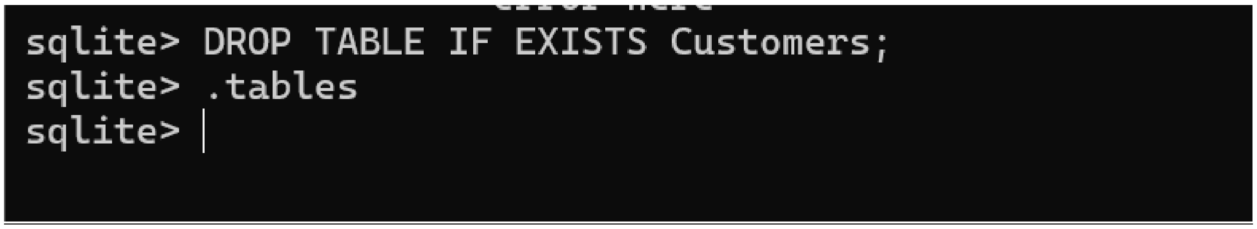
d) **Step4: Verification**

Now, verify if the above commands were successful.



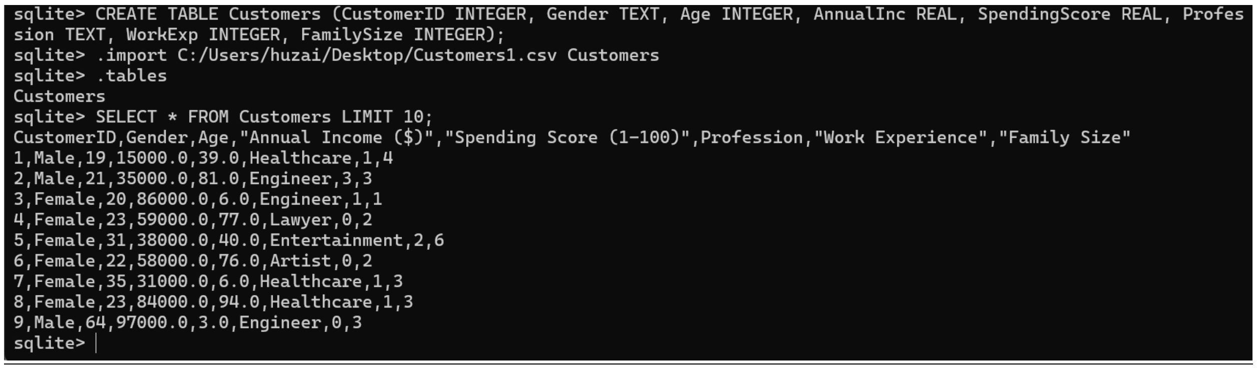
```
SQLite version 3.47.0 2024-10-21 16:30:22
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open C:/Users/munirhuz/Desktop/customers2.db
sqlite> .mode csv
sqlite> .import C:/Users/munirhuz/Desktop/Customers2.csv Customers
sqlite> SELECT * FROM Customers LIMIT 10;
1,Male,19,15000,39,Healthcare,1,4
2,Male,21,35000,81,Engineer,3,3
3,Female,20,86000,6,Engineer,1,1
4,Female,23,59000,77,Lawyer,0,2
5,Female,31,38000,40,Entertainment,2,6
6,Female,22,58000,76,Artist,0,2
7,Female,35,31000,6,Healthcare,1,3
8,Female,23,84000,94,Healthcare,1,3
9,Male,64,97000,3,Engineer,0,3
10,Female,30,98000,72,Artist,1,4
sqlite>
```

- If it doesn't work, you can try creating the table manually and then copying the data to it.
  I will first drop the table and then check if there are any tables.



```
sqlite> DROP TABLE IF EXISTS Customers;
sqlite> .tables
sqlite>
```

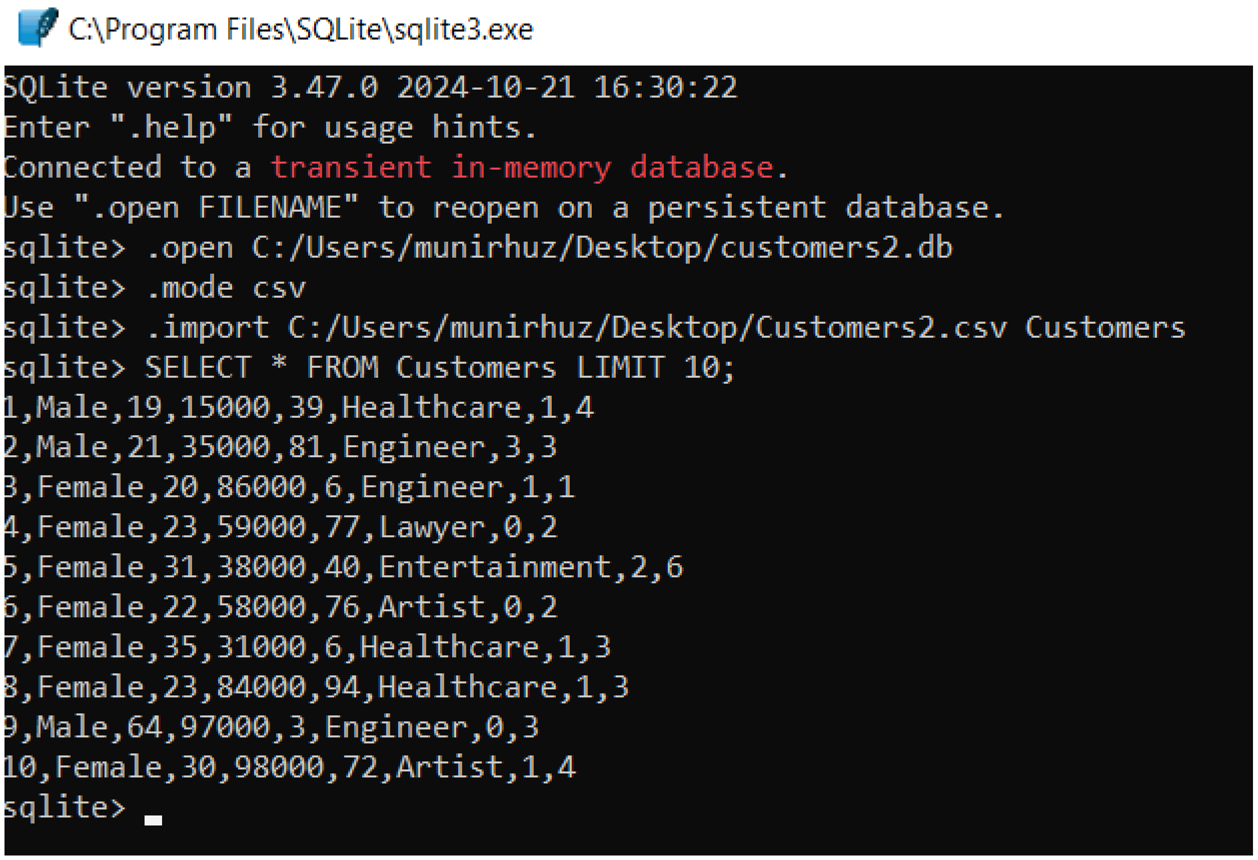
  If there were, the output would have been "Customers".
- Now, we will be manually creating the table and importing all of the data into it.
  Also, check with '.tables' command to verify.



```
sqlite> CREATE TABLE Customers (CustomerID INTEGER, Gender TEXT, Age INTEGER, AnnualInc REAL, SpendingScore REAL, Profes
sion TEXT, WorkExp INTEGER, FamilySize INTEGER);
sqlite> .import C:/Users/huzai/Desktop/Customers1.csv Customers
sqlite> .tables
Customers
sqlite> SELECT * FROM Customers LIMIT 10;
CustomerID,Gender,Age,"Annual Income ($)","Spending Score (1-100)",Profession,"Work Experience","Family Size"
1,Male,19,15000.0,39.0,Healthcare,1,4
2,Male,21,35000.0,81.0,Engineer,3,3
3,Female,20,86000.0,6.0,Engineer,1,1
4,Female,23,59000.0,77.0,Lawyer,0,2
5,Female,31,38000.0,40.0,Entertainment,2,6
6,Female,22,58000.0,76.0,Artist,0,2
7,Female,35,31000.0,6.0,Healthcare,1,3
8,Female,23,84000.0,94.0,Healthcare,1,3
9,Male,64,97000.0,3.0,Engineer,0,3
sqlite>
```

e) **<u>Step5:</u>**

Check the structure of the imported table.

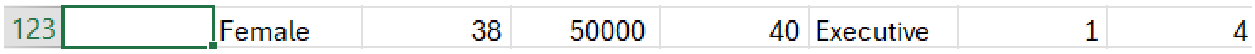


C:\Program Files\SQLite\sqlite3.exe

```
SQLite version 3.47.0 2024-10-21 16:30:22
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open C:/Users/munirhuz/Desktop/customers2.db
sqlite> .mode csv
sqlite> .import C:/Users/munirhuz/Desktop/Customers2.csv Customers
sqlite> SELECT * FROM Customers LIMIT 10;
1,Male,19,15000,39,Healthcare,1,4
2,Male,21,35000,81,Engineer,3,3
3,Female,20,86000,6,Engineer,1,1
4,Female,23,59000,77,Lawyer,0,2
5,Female,31,38000,40,Entertainment,2,6
6,Female,22,58000,76,Artist,0,2
7,Female,35,31000,6,Healthcare,1,3
8,Female,23,84000,94,Healthcare,1,3
9,Male,64,97000,3,Engineer,0,3
10,Female,30,98000,72,Artist,1,4
sqlite>
```

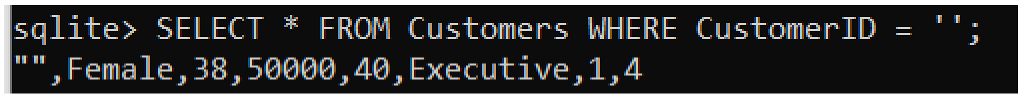
2. Issues in the database:

- **Missing Values/Blank OR Empty String**:
  - **CustomerID**: There is 1 missing value in the CustomerID column, which could disrupt data retrieval since this is the primary key and is expected to be unique and non-null.
    Example: Row 123 in csv file.

123		Female	38	50000	40	Executive	1	4

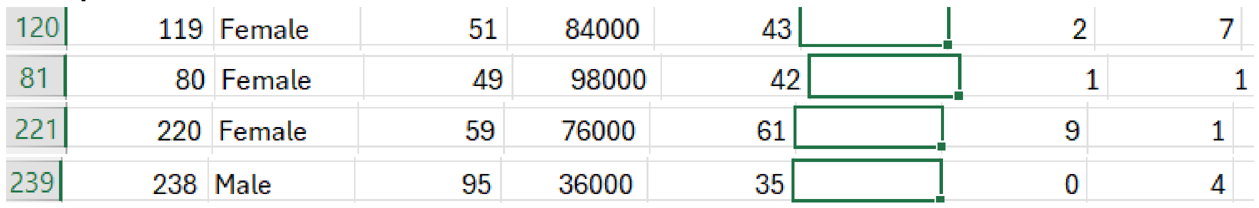
    SQL Command: SELECT * FROM Customers WHERE CustomerID = ' ';

    ```
    sqlite> SELECT * FROM Customers WHERE CustomerID = '';
    "",Female,38,50000,40,Executive,1,4
    ```

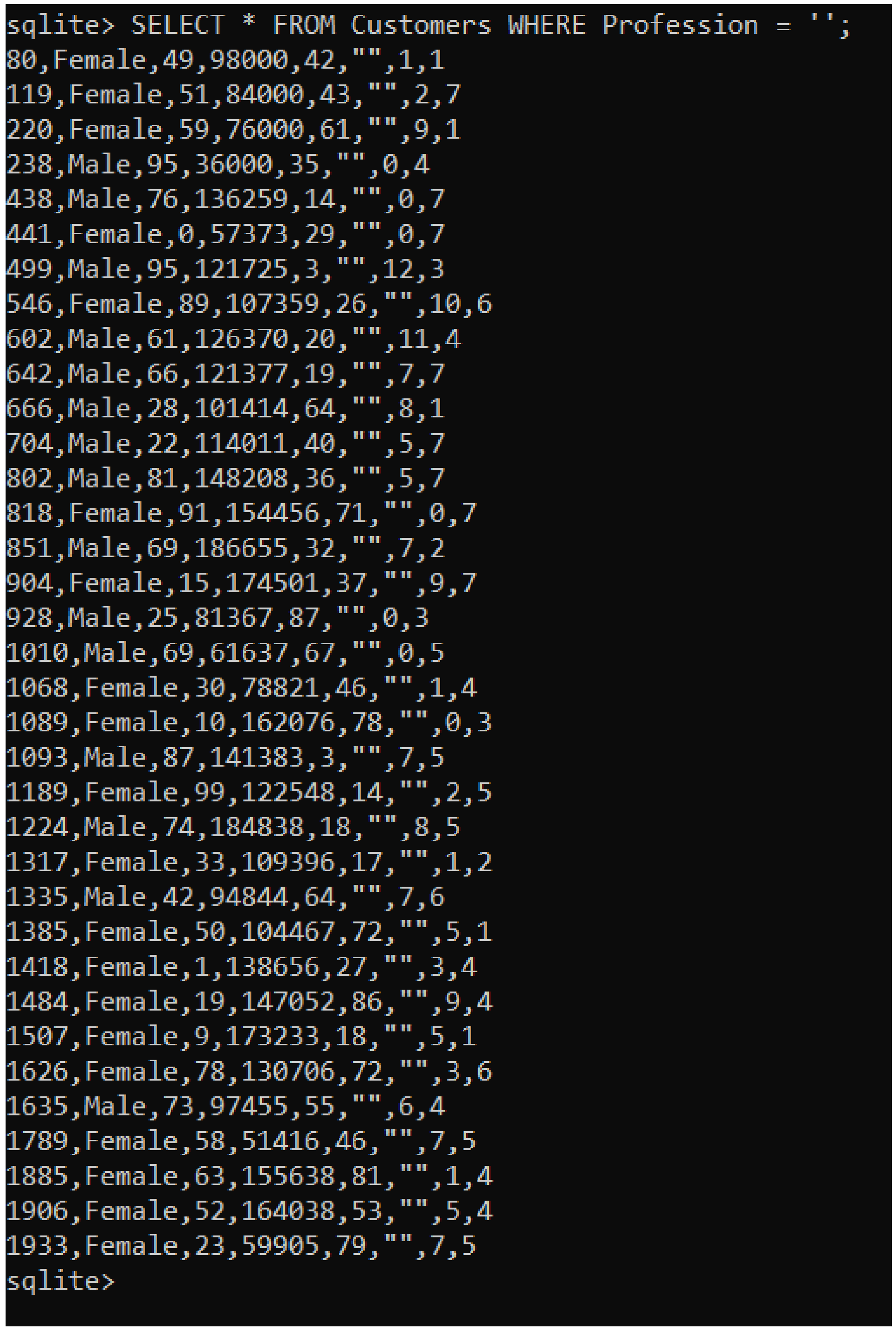
    NOTE:
    - **For NULL:**If you are running the IS NULL command i.e., **SELECT * FROM Customers WHERE CustomerID IS NULL;** and it is not running try using the above command.
    - **For Zero Values (if zero is not valid in your primary key logic):** If you are running the IS NULL command i.e., **SELECT * FROM Customers WHERE CustomerID = 0;** and it is not running try using the above command.
  - **Profession**: There are 35 missing values in the Profession column, which may cause issues if the application expects all values to be present without a NULL-handling mechanism.

Example: Row 81, 120, 221, 239 etc. in csv file.



120	119	Female	51	84000	43		2	7
81	80	Female	49	98000	42		1	1
221	220	Female	59	76000	61		9	1
239	238	Male	95	36000	35		0	4

SQL Command: SELECT * FROM Customers WHERE Profession = ' ';

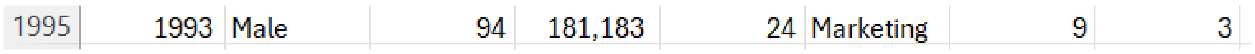


```
sqlite> SELECT * FROM Customers WHERE Profession = '';
80,Female,49,98000,42,"",1,1
119,Female,51,84000,43,"",2,7
220,Female,59,76000,61,"",9,1
238,Male,95,36000,35,"",0,4
438,Male,76,136259,14,"",0,7
441,Female,0,57373,29,"",0,7
499,Male,95,121725,3,"",12,3
546,Female,89,107359,26,"",10,6
602,Male,61,126370,20,"",11,4
642,Male,66,121377,19,"",7,7
666,Male,28,101414,64,"",8,1
704,Male,22,114011,40,"",5,7
802,Male,81,148208,36,"",5,7
818,Female,91,154456,71,"",0,7
851,Male,69,186655,32,"",7,2
904,Female,15,174501,37,"",9,7
928,Male,25,81367,87,"",0,3
1010,Male,69,61637,67,"",0,5
1068,Female,30,78821,46,"",1,4
1089,Female,10,162076,78,"",0,3
1093,Male,87,141383,3,"",7,5
1189,Female,99,122548,14,"",2,5
1224,Male,74,184838,18,"",8,5
1317,Female,33,109396,17,"",1,2
1335,Male,42,94844,64,"",7,6
1385,Female,50,104467,72,"",5,1
1418,Female,1,138656,27,"",3,4
1484,Female,19,147052,86,"",9,4
1507,Female,9,173233,18,"",5,1
1626,Female,78,130706,72,"",3,6
1635,Male,73,97455,55,"",6,4
1789,Female,58,51416,46,"",7,5
1885,Female,63,155638,81,"",1,4
1906,Female,52,164038,53,"",5,4
1933,Female,23,59905,79,"",7,5
sqlite>
```

NOTE:

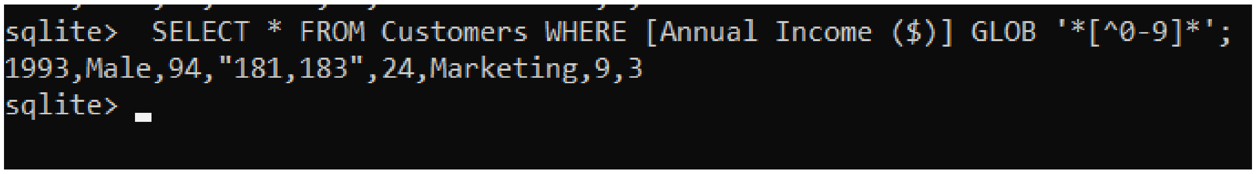
- **For NULL:**If you are running the IS NULL command i.e., **SELECT * FROM Customers WHERE Profession IS NULL;** and it is not running try using the above command.
- **For Zero Values (if zero is not valid in your primary key logic):** If you are running the IS NULL command i.e., **SELECT * FROM Customers WHERE Profession = 0;** and it is not running try using the above command.

➢ **Data Type Discrepancies**:
Annual Income ($): The Annual Income ($) column is stored as an object (string) instead of an integer, indicating potential incorrect data entries or format issues (e.g., extra spaces or symbols). This may prevent the application from processing income values correctly.
Example: Row 1995 in csv file.



1995	1993	Male	94	181,183	24	Marketing	9	3

SQL Command: SELECT * FROM Customers WHERE [Annual Income ($)] GLOB '*[^0-9]*';

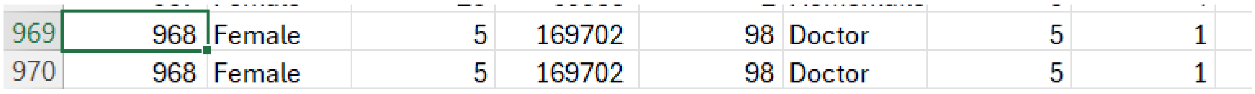


```
sqlite>  SELECT * FROM Customers WHERE [Annual Income ($)] GLOB '*[^0-9]*';
1993,Male,94,"181,183",24,Marketing,9,3
sqlite>
```

NOTE:

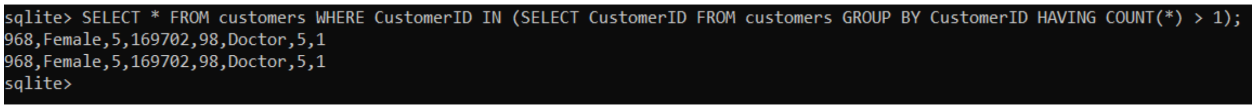
- If you are using the command SELECT * FROM Customers WHERE [Annual Income ($)] NOT LIKE '%[^0-9]%'; than this will output all of the queries in the table which is not right.
- For this issue to be resolved we use **GLOB** which is an SQLite operator that allows **pattern matching** using **Unix shell-style wildcards**.
- Also, *[^0-9]* matches any row where Annual Income ($) contains characters other than digits (0-9).

➢ **Duplicate Records**:
CustomerID: There are duplicate entries for CustomerID 968. Since CustomerID is a primary key, duplicates here could lead to conflicts in data integrity and issues when fetching specific customer records.
Example: Row 969 & 970 in csv file.



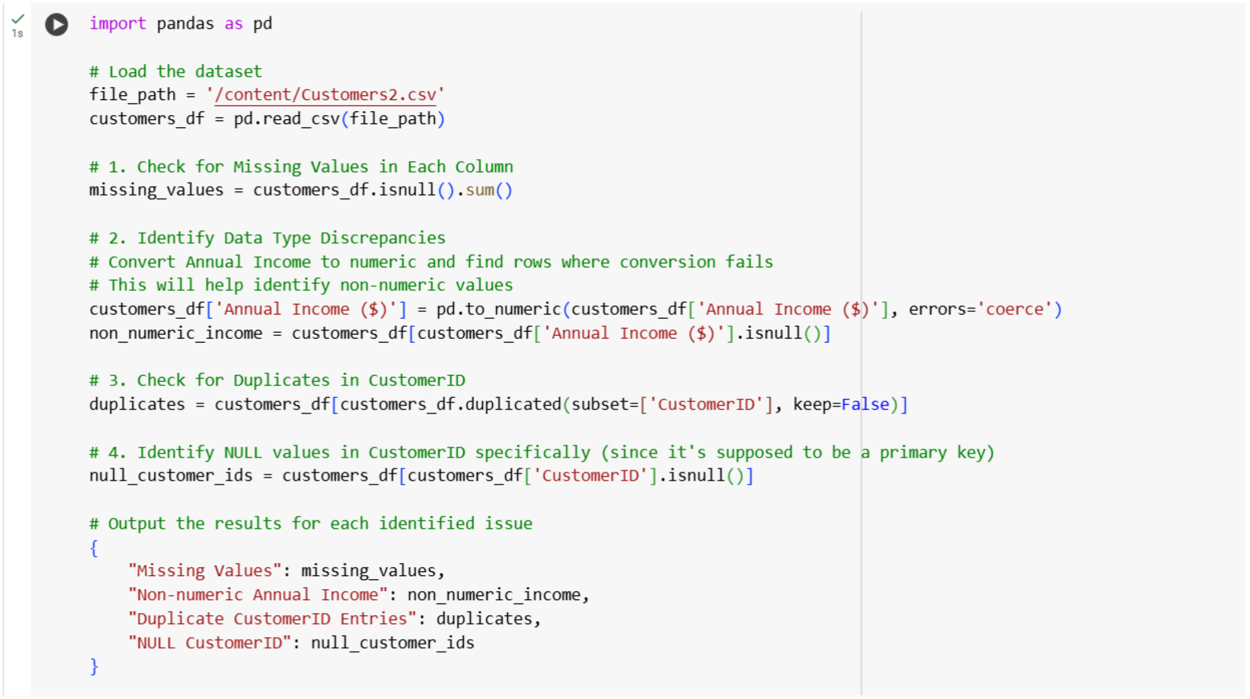
969	968	Female	5	169702	98	Doctor	5	1
970	968	Female	5	169702	98	Doctor	5	1

SQL Command: SELECT * FROM customers WHERE CustomerID IN (SELECT CustomerID FROM customers GROUP BY CustomerID HAVING COUNT(*) > 1);



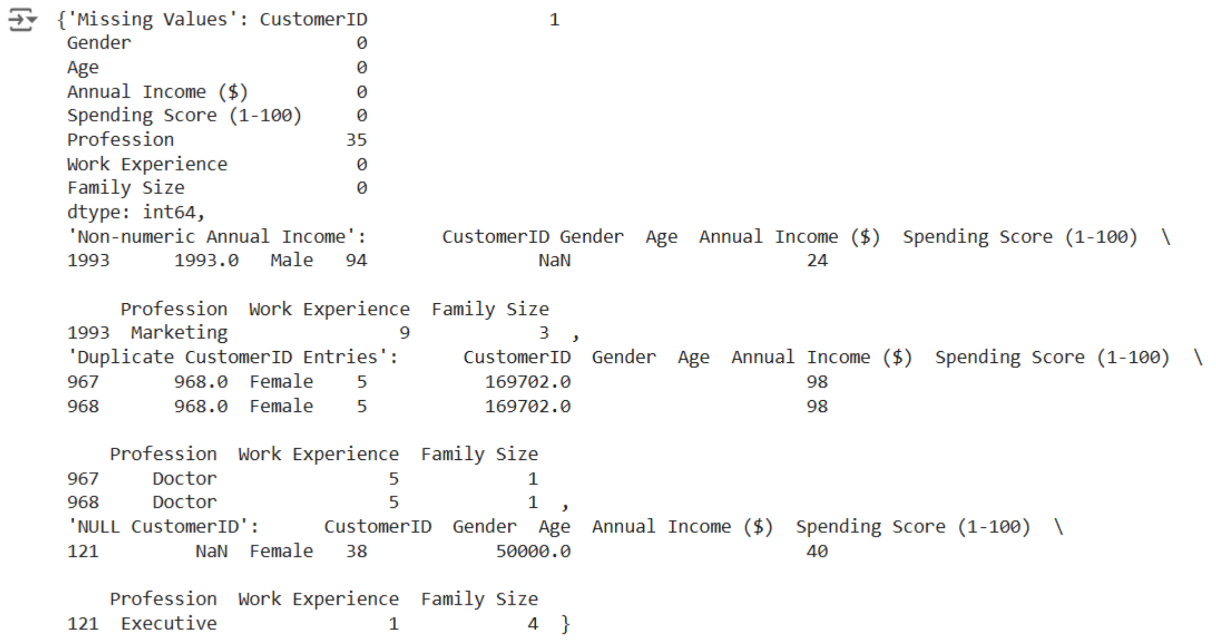
```
sqlite> SELECT * FROM customers WHERE CustomerID IN (SELECT CustomerID FROM customers GROUP BY CustomerID HAVING COUNT(*) > 1);
968,Female,5,169702,98,Doctor,5,1
968,Female,5,169702,98,Doctor,5,1
sqlite>
```

3. Python Code for Identifying Issues:



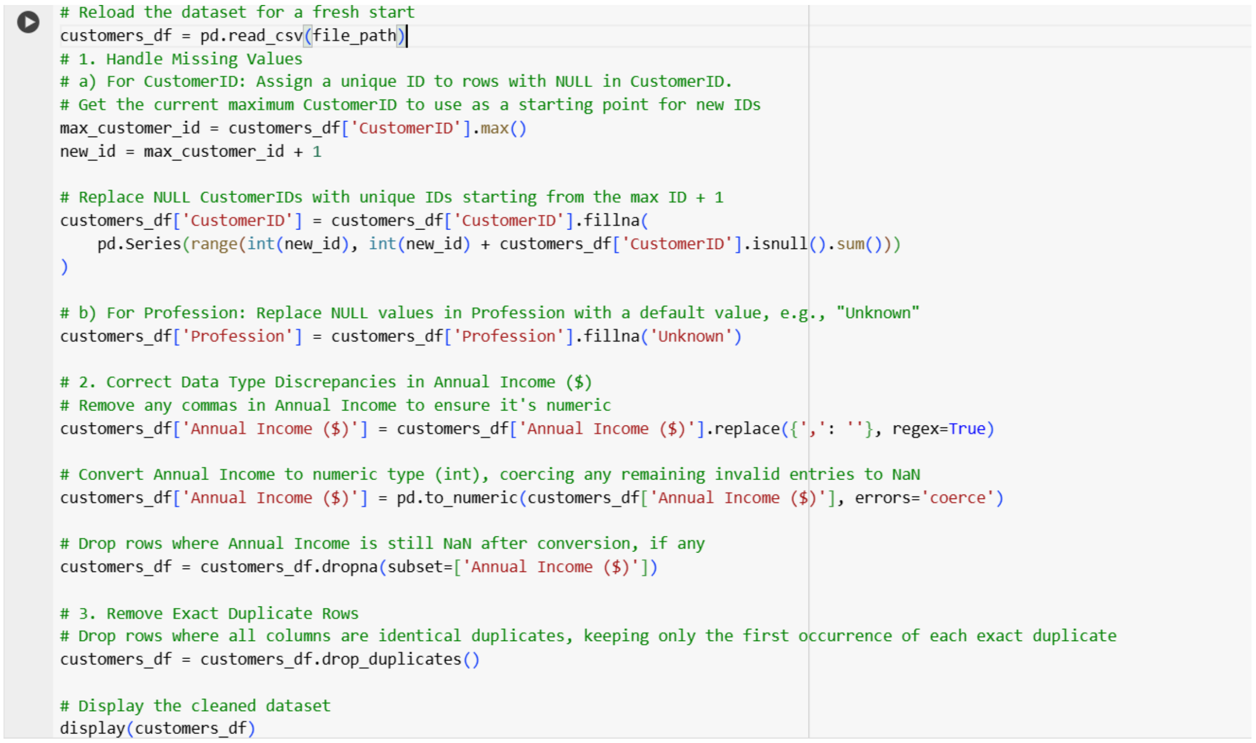
```python
import pandas as pd

# Load the dataset
file_path = '/content/Customers2.csv'
customers_df = pd.read_csv(file_path)

# 1. Check for Missing Values in Each Column
missing_values = customers_df.isnull().sum()

# 2. Identify Data Type Discrepancies
# Convert Annual Income to numeric and find rows where conversion fails
# This will help identify non-numeric values
customers_df['Annual Income ($)'] = pd.to_numeric(customers_df['Annual Income ($)'], errors='coerce')
non_numeric_income = customers_df[customers_df['Annual Income ($)'].isnull()]

# 3. Check for Duplicates in CustomerID
duplicates = customers_df[customers_df.duplicated(subset=['CustomerID'], keep=False)]

# 4. Identify NULL values in CustomerID specifically (since it's supposed to be a primary key)
null_customer_ids = customers_df[customers_df['CustomerID'].isnull()]

# Output the results for each identified issue
{
    "Missing Values": missing_values,
    "Non-numeric Annual Income": non_numeric_income,
    "Duplicate CustomerID Entries": duplicates,
    "NULL CustomerID": null_customer_ids
}
```

Output:



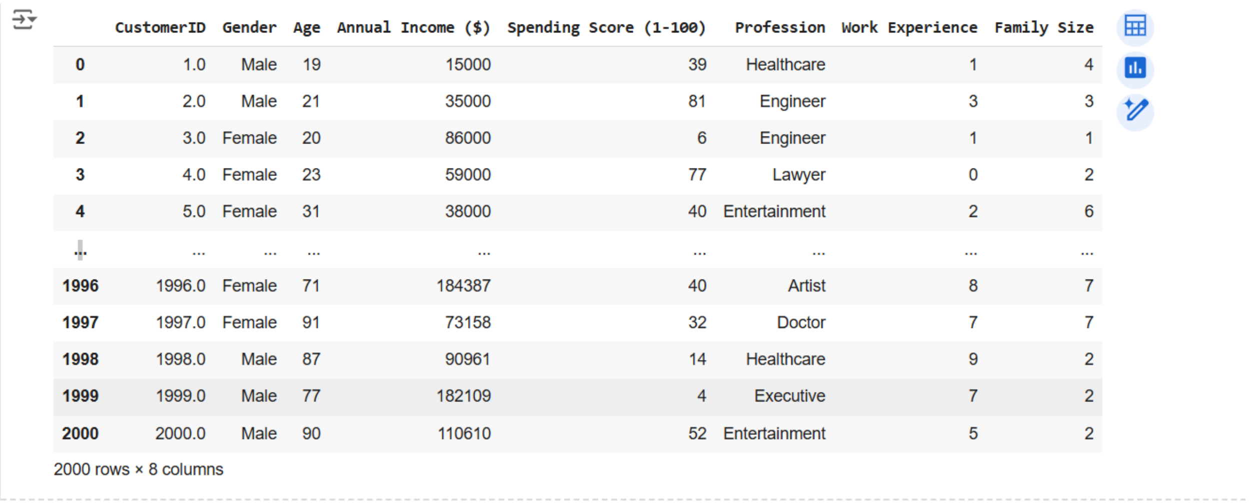
```
{'Missing Values': CustomerID                  1
 Gender                     0
 Age                        0
 Annual Income ($)          0
 Spending Score (1-100)     0
 Profession                35
 Work Experience            0
 Family Size                0
 dtype: int64,
 'Non-numeric Annual Income':       CustomerID Gender  Age  Annual Income ($)  Spending Score (1-100)  \
 1993       1993.0    Male   94                NaN                      24

      Profession  Work Experience  Family Size
 1993  Marketing                9            3  ,
 'Duplicate CustomerID Entries':       CustomerID  Gender  Age  Annual Income ($)  Spending Score (1-100)  \
 967        968.0   Female    5           169702.0                      98
 968        968.0   Female    5           169702.0                      98

     Profession  Work Experience  Family Size
 967      Doctor                5            1
 968      Doctor                5            1  ,
 'NULL CustomerID':       CustomerID  Gender  Age  Annual Income ($)  Spending Score (1-100)  \
 121          NaN  Female   38            50000.0                      40

     Profession  Work Experience  Family Size
 121  Executive                1            4  }
```

4. Python Code to Resolve the Issues:



```python
# Reload the dataset for a fresh start
customers_df = pd.read_csv(file_path)
# 1. Handle Missing Values
# a) For CustomerID: Assign a unique ID to rows with NULL in CustomerID.
# Get the current maximum CustomerID to use as a starting point for new IDs
max_customer_id = customers_df['CustomerID'].max()
new_id = max_customer_id + 1

# Replace NULL CustomerIDs with unique IDs starting from the max ID + 1
customers_df['CustomerID'] = customers_df['CustomerID'].fillna(
    pd.Series(range(int(new_id), int(new_id) + customers_df['CustomerID'].isnull().sum()))
)

# b) For Profession: Replace NULL values in Profession with a default value, e.g., "Unknown"
customers_df['Profession'] = customers_df['Profession'].fillna('Unknown')

# 2. Correct Data Type Discrepancies in Annual Income ($)
# Remove any commas in Annual Income to ensure it's numeric
customers_df['Annual Income ($)'] = customers_df['Annual Income ($)'].replace({',': ''}, regex=True)

# Convert Annual Income to numeric type (int), coercing any remaining invalid entries to NaN
customers_df['Annual Income ($)'] = pd.to_numeric(customers_df['Annual Income ($)'], errors='coerce')

# Drop rows where Annual Income is still NaN after conversion, if any
customers_df = customers_df.dropna(subset=['Annual Income ($)'])

# 3. Remove Exact Duplicate Rows
# Drop rows where all columns are identical duplicates, keeping only the first occurrence of each exact duplicate
customers_df = customers_df.drop_duplicates()

# Display the cleaned dataset
display(customers_df)
```

Output:



	CustomerID	Gender	Age	Annual Income ($)	Spending Score (1-100)	Profession	Work Experience	Family Size
0	1.0	Male	19	15000	39	Healthcare	1	4
1	2.0	Male	21	35000	81	Engineer	3	3
2	3.0	Female	20	86000	6	Engineer	1	1
3	4.0	Female	23	59000	77	Lawyer	0	2
4	5.0	Female	31	38000	40	Entertainment	2	6
...	...	...	...	...	...	...	...	...
1996	1996.0	Female	71	184387	40	Artist	8	7
1997	1997.0	Female	91	73158	32	Doctor	7	7
1998	1998.0	Male	87	90961	14	Healthcare	9	2
1999	1999.0	Male	77	182109	4	Executive	7	2
2000	2000.0	Male	90	110610	52	Entertainment	5	2

2000 rows × 8 columns

5. Conclusion:

In conclusion, the analysis revealed issues such as missing values, data type inconsistencies, and duplicate records in the Customers dataset. These issues, including NULLs in critical fields, non-numeric entries in Annual Income, and duplicate CustomerIDs, can disrupt data retrieval and display in the application. Addressing these with SQL queries to clean and standardize the data will enhance data integrity, ensuring smoother and more reliable form functionality.