1. Selecting the Right Database

```
USE portfolio_projects;
```

Sets the active database to portfolio_projects, ensuring all queries run within this database and making sure not performing anything in the wrong database

---

2. Creating a Staging Table

```
CREATE TABLE IF NOT EXISTS users_staging LIKE users;
```

Creates a duplicate of the users table named users_staging. This staging table allows safe data cleaning without modifying the original data.

3. Checking Table Structure

```
DESC users;
```

```
DESC users_staging;
```

Displays the structure of both tables to ensure they match.

4. Copying Data to Staging Table

```
INSERT INTO users_staging
```

```
SELECT * FROM users;
```

Copies all data from users into users_staging for further cleaning and analysis.

5. Checking Data Integrity

```
SELECT * FROM users AS t1
```

```
LEFT JOIN users_staging AS t2
```

```
ON t1.user_id = t2.user_id
```

```
WHERE t2.user_id IS NULL;
```

Checking no data is missing in the staging table by checking for any records in users that are not present in users_staging.

---

6. Data Cleaning

Adding Constraints to user_id

```
ALTER TABLE users_staging
MODIFY COLUMN user_id INT NOT NULL,
ADD PRIMARY KEY(user_id);
```

Making user_id as a unique identifier by making it a primary key and preventing NULL values.

7. Standardizing total_balance Column

```
ALTER TABLE users_staging
MODIFY COLUMN total_balance DECIMAL(8,2);
```

Ensures financial accuracy by setting total_balance to DECIMAL(8,2), preventing unrealistic values.

8. Identifying Duplicate Records

```
WITH duplicate_rows AS (
SELECT *, ROW_NUMBER() OVER(PARTITION BY user_id, first_name, last_name, email, gender, age, contact_info, location, total_balance, transaction_count, total_spent, account_age ORDER BY user_id) AS row_count
FROM users_staging)
SELECT * FROM duplicate_rows WHERE row_count > 1;
```

Identifies duplicate records based on all columns using CTEs.

---

9. Cleaning first_name and last_name column

```
UPDATE users_staging
SET first_name = CONCAT(UCASE(LEFT(first_name, 1)), LCASE(SUBSTRING(first_name, 2)));
```

Standardizes capitalization in first_name.

```
SELECT COUNT(*) FROM users_staging
WHERE first_name = CONCAT(UCASE(LEFT(first_name, 1)), LCASE(SUBSTRING(first_name, 2)));
```

Verifies whether all names are properly formatted.

10.. Standardizing Email Format

```
UPDATE users_staging
SET email = INSERT(email, LOCATE('.', email), 1, '');
```

Removes unnecessary dots from emails.

```
UPDATE users_staging
SET email = CONCAT(LEFT(email, LOCATE('@', email)), LCASE(RIGHT(email, LENGTH(email) - LOCATE('@', email))));
```

Ensures email domains are in lowercase.

```
UPDATE users_staging
SET email = REPLACE(email, '.in', '.com')
WHERE email LIKE '%.in';
```

Corrects incorrect domain extensions.

---

11. Cleaning gender Column

```
UPDATE users_staging
SET gender = CASE
  WHEN gender = 'M' THEN 'Male'
  WHEN gender = 'F' THEN 'Female'
  ELSE gender
END;
```

Standardizes gender values for consistency.

---

12. Cleaning location Column

```
UPDATE users_staging
SET location = CASE
  WHEN location = 'Pokhara123Chitwan' THEN 'Chitwan'
  WHEN location = 'Biratn@gar' THEN 'Biratnagar'
  WHEN location = 'Kathm#$ndu' OR location = 'Balen City' THEN 'Kathmandu'
```

```
  WHEN location = 'Pokahara' OR location = 'P0khara' THEN 'Pokhara'
  WHEN location = 'lalitpur4443' THEN 'Lalitpur'
  WHEN location = 'Raasuwa' OR location = 'rasuwa' THEN 'Rasuwa'
  ELSE location
END;
```

Corrects misspelled and inconsistent location names.

---

13. Cleaning age Column

```
DELETE FROM users_staging
WHERE age < 16;
```

Removes records where age is less than 16, ensuring valid data.

```
ALTER TABLE users_staging
ADD CONSTRAINT age_check CHECK (age >= 16);
```

Prevents future invalid age entries.

---

14. Cleaning contact_info Column

```
SELECT contact_info, CHAR_LENGTH(contact_info) AS tendigits
FROM users_staging
WHERE CHAR_LENGTH(contact_info) != 10;
```

Identifies phone numbers that do not have exactly 10 digits.

---

15. Validating Financial Columns

```
SELECT DISTINCT total_balance FROM users_staging ORDER BY 1;
```

Ensures all balance values are within expected limits.

```
SELECT DISTINCT transaction_count FROM users_staging ORDER BY 1;
```

Identifies anomalies in transaction counts.

---

16. Cleaning account_age Column

```
UPDATE users_staging
SET account_age_in_day = CASE
  WHEN account_age_in_day < 0 THEN account_age_in_day * -1
  ELSE account_age_in_day
END;
```

Converts negative account age values to positive.

Conclusion

This project ensures a clean and standardized dataset for analysis by:

- Creating a staging table for safe data cleaning.
- Identifying and removing duplicate values.
- Standardizing name, email, and location formats.
- Validating numerical fields and financial constraints.
- Fixing inconsistencies in categorical columns such as gender and location.

Now this data is ready for Exploratory Data Analysis and dashboard