PAN Number Validation Project

Objective

You are required to clean and validate a dataset containing the Permanent Account Numbers (PAN) of Indian nationals. The goal is to ensure each PAN number adheres to the official format and categorize it as either **Valid** or **Invalid**. The dataset is provided in an Excel file: `PAN Number Validation Dataset.xlsx`.

Instructions

1. Data Cleaning and Preprocessing

- **Handle missing data:** Identify PAN numbers that are missing. Decide whether to remove these rows or impute values depending on the context.
- **Remove duplicates:** Ensure there are no duplicate PAN numbers. Remove any duplicates found.
- **Trim spaces:** Remove any leading or trailing spaces from PAN numbers.
- **Standardize case:** Convert all letters to uppercase.

2. PAN Format Validation

A valid PAN number must satisfy all of the following criteria:

1. **Length:** Exactly 10 characters.
2. **Format:** `AAAAA1234A`
   - **First five characters:** Uppercase alphabets (`A-Z`).
     - Adjacent letters cannot be the same (e.g., `AABCD` is invalid).
     - All five letters cannot form a sequence (e.g., `ABCDE`, `BCDEF` are invalid).
   - **Next four characters:** Numeric digits (`0-9`).
     - Adjacent digits cannot be the same (e.g., `1123` is invalid).
     - All four digits cannot form a sequence (e.g., `1234`, `2345` are invalid).
   - **Last character:** Uppercase alphabet (`A-Z`).

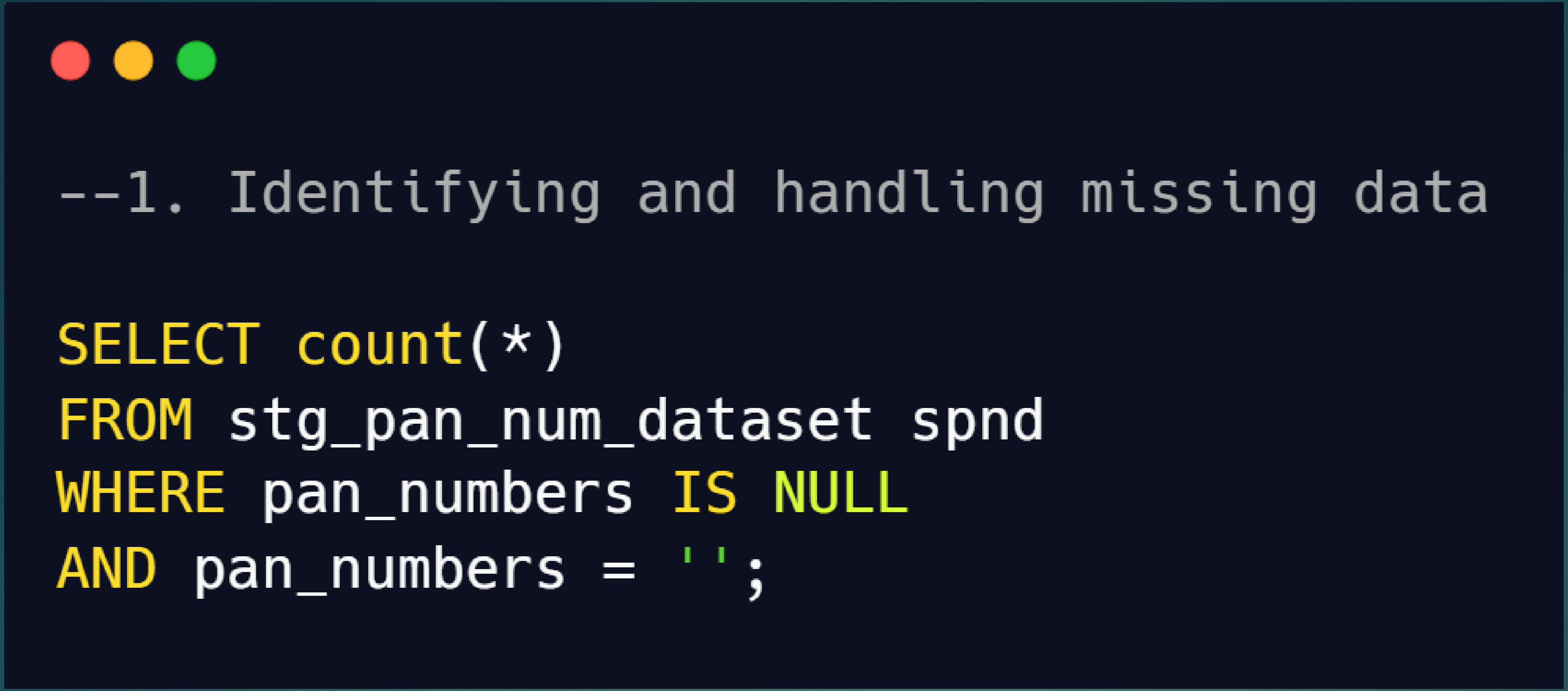
Example of a valid PAN: `AHGVE1276F`

3. Categorization

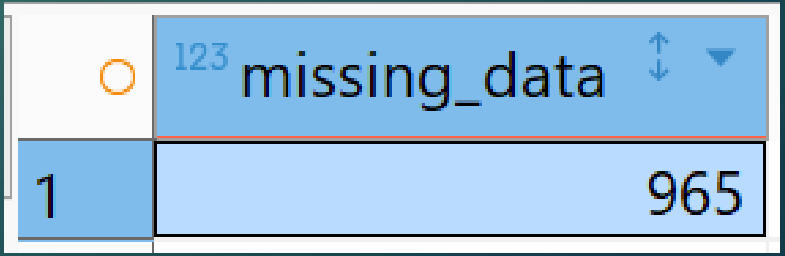
- **Valid PAN:** Meets all the criteria mentioned above.
- **Invalid PAN:** Does not meet the correct format, is incomplete, or contains non-alphanumeric characters.

4. Tasks

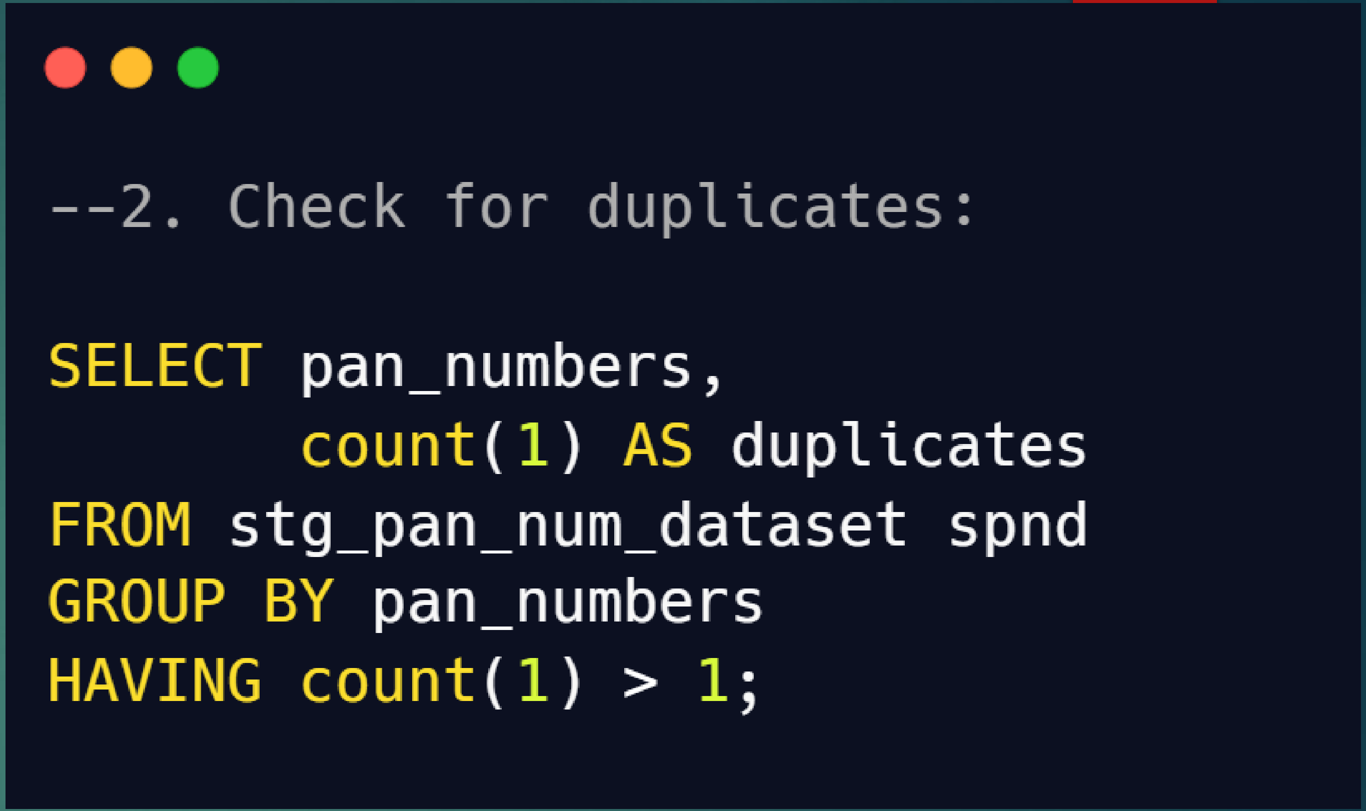
1. Validate all PAN numbers according to the rules.
2. Categorize PAN numbers into **Valid** and **Invalid**.
3. Create a summary report containing:
    - Total records processed
    - Total valid PANs
    - Total invalid PANs
    - Total missing or incomplete PANs (if any)



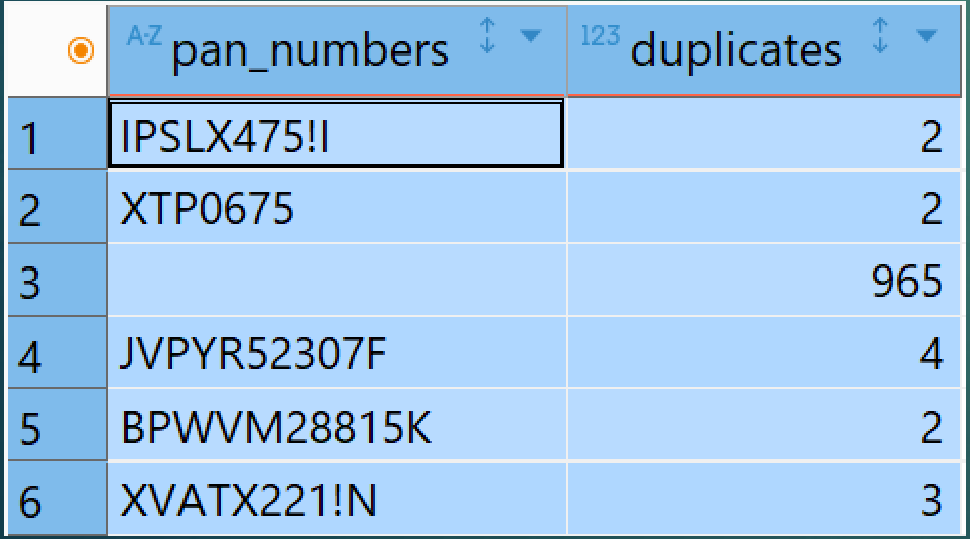
```sql
--1. Identifying and handling missing data

SELECT count(*)
FROM stg_pan_num_dataset spnd
WHERE pan_numbers IS NULL
AND pan_numbers = '';
```



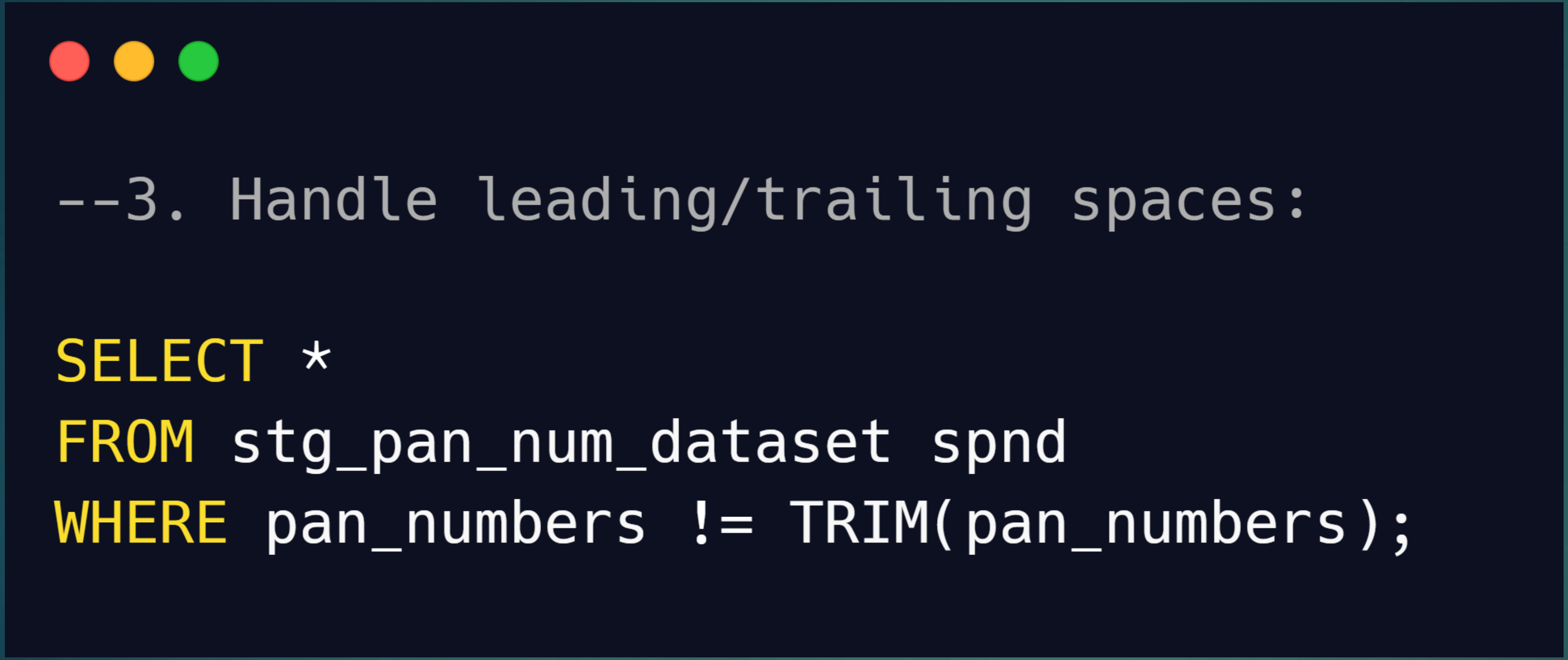
```
   missing_data
1           965
```



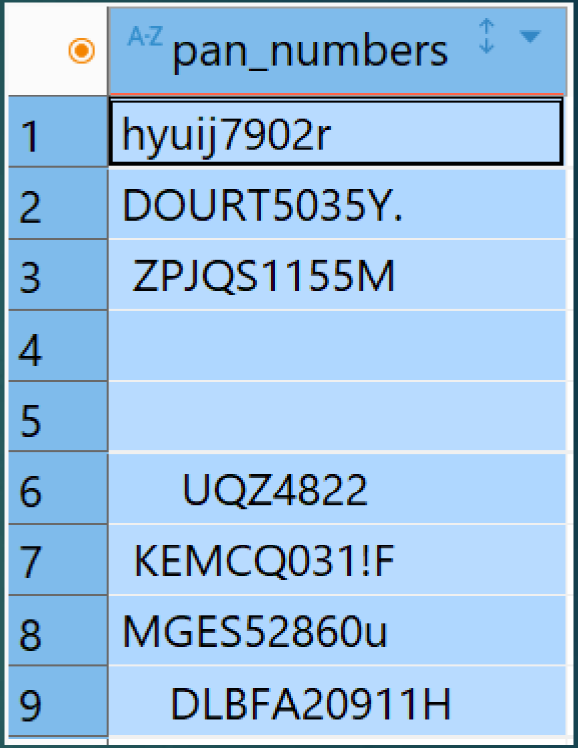
```
--2. Check for duplicates:

SELECT pan_numbers,
       count(1) AS duplicates
FROM stg_pan_num_dataset spnd
GROUP BY pan_numbers
HAVING count(1) > 1;
```



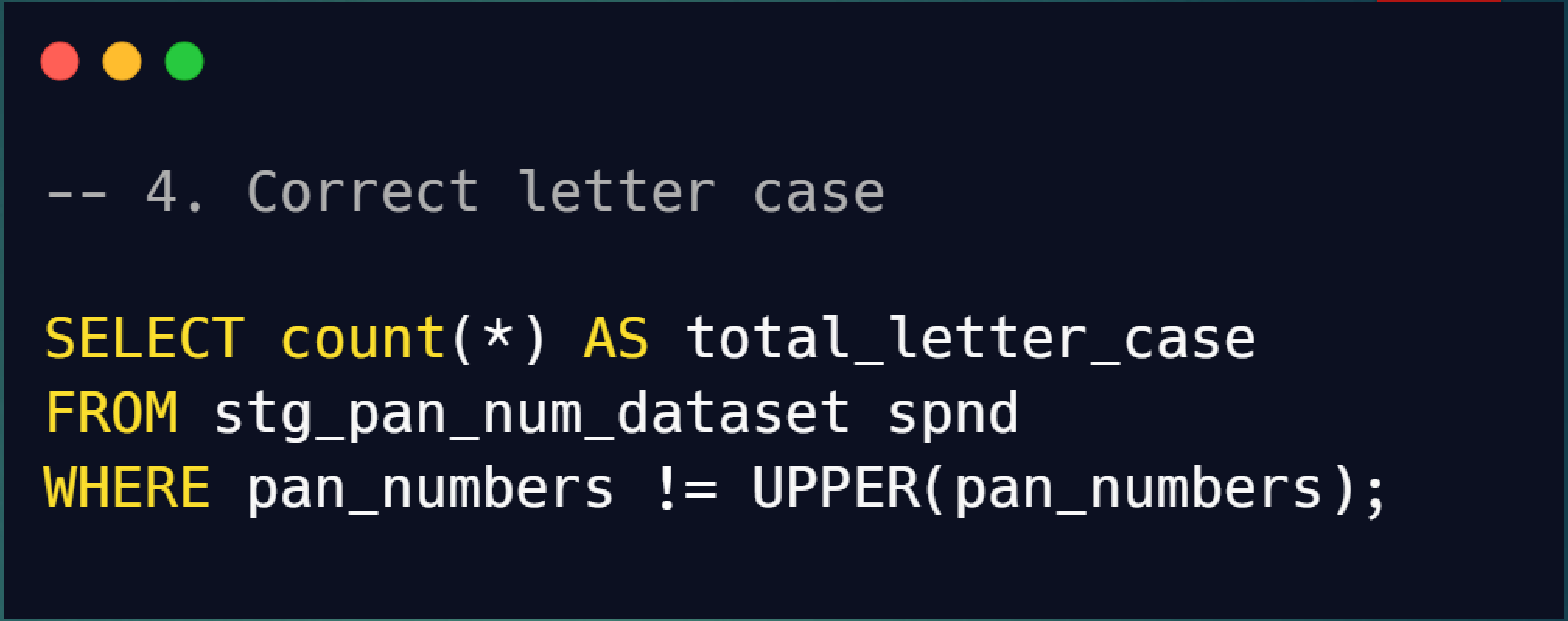
	pan_numbers	duplicates
1	IPSLX475!I	2
2	XTP0675	2
3		965
4	JVPYR52307F	4
5	BPWVM28815K	2
6	XVATX221!N	3



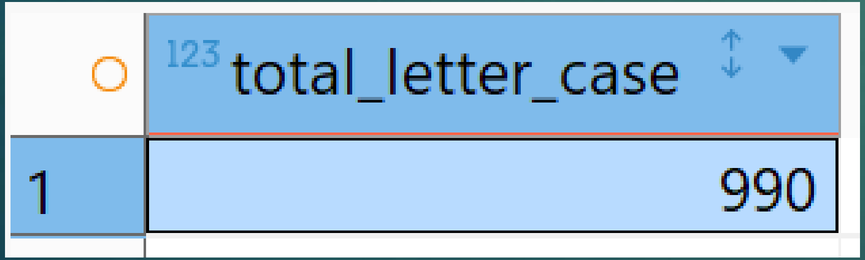
```
--3. Handle leading/trailing spaces:

SELECT *
FROM stg_pan_num_dataset spnd
WHERE pan_numbers != TRIM(pan_numbers);
```



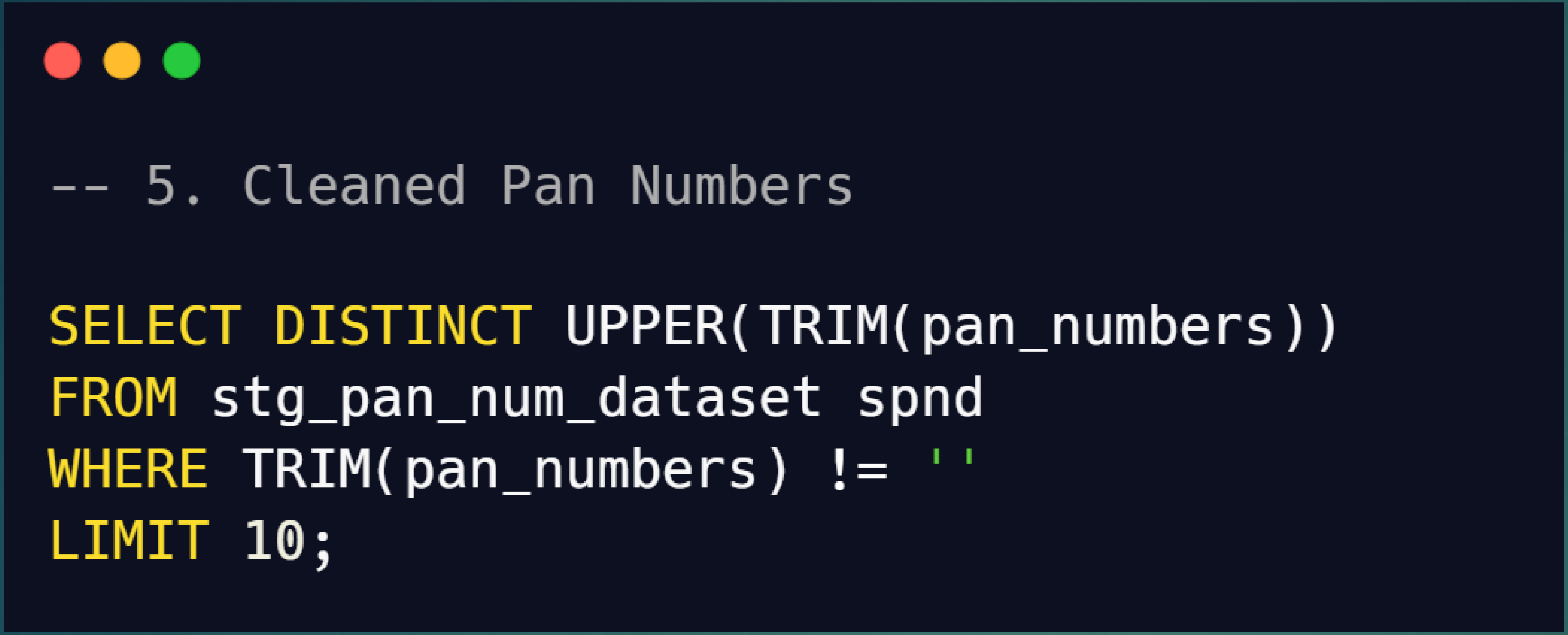
	pan_numbers
1	hyuij7902r
2	DOURT5035Y.
3	ZPJQS1155M
4	
5	
6	UQZ4822
7	KEMCQ031!F
8	MGES52860u
9	DLBFA20911H



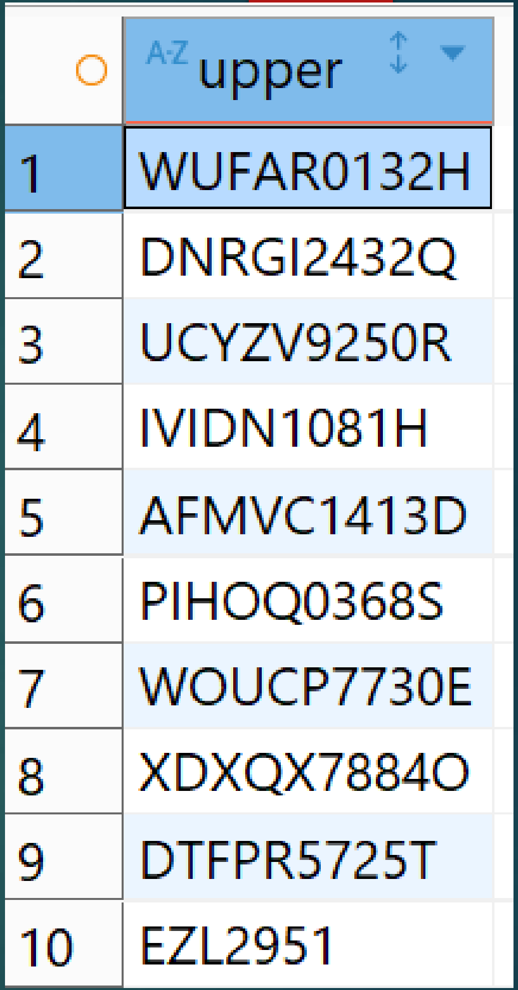
```sql
-- 4. Correct letter case

SELECT count(*) AS total_letter_case
FROM stg_pan_num_dataset spnd
WHERE pan_numbers != UPPER(pan_numbers);
```



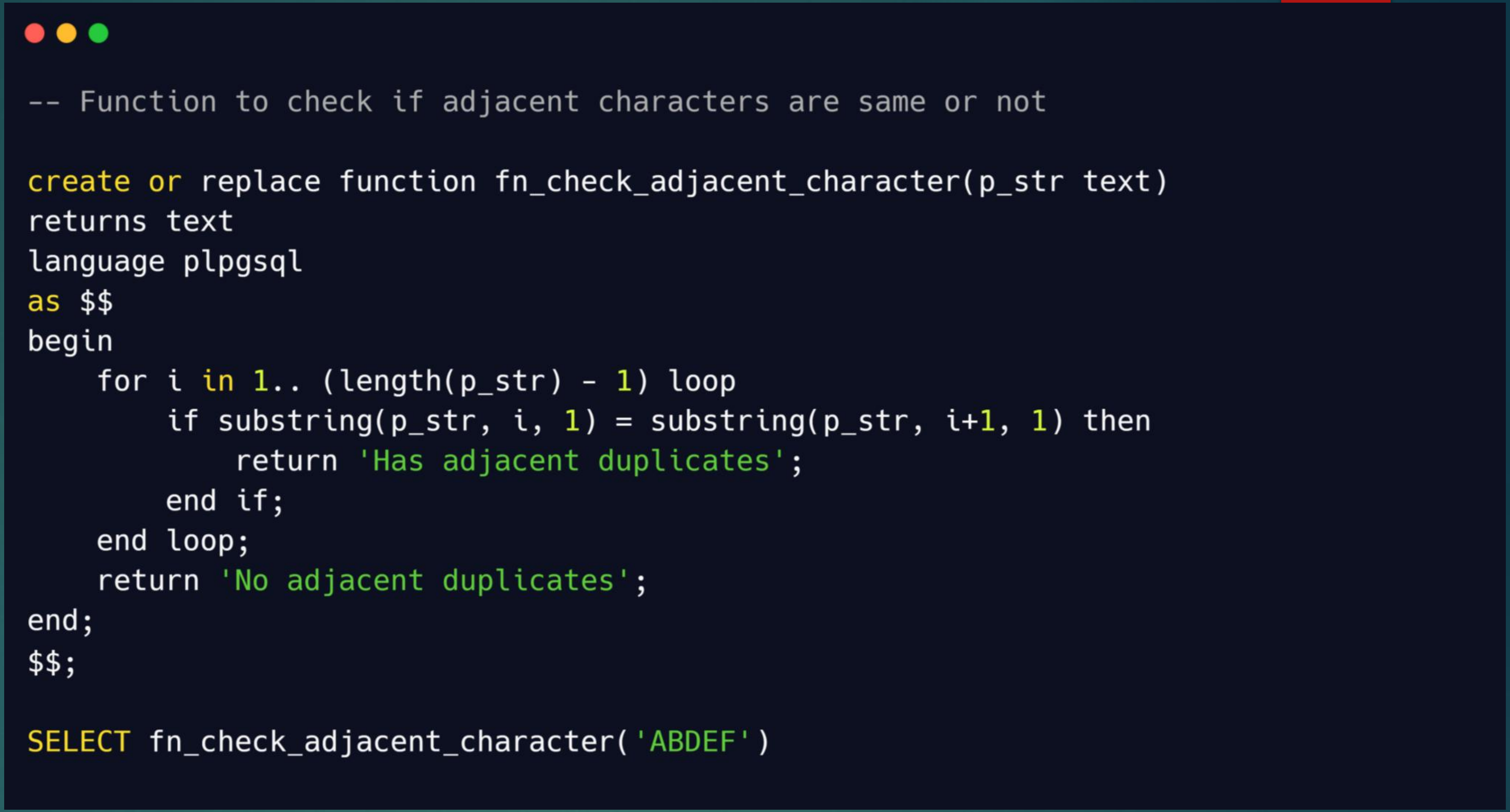
```
   total_letter_case
1                990
```



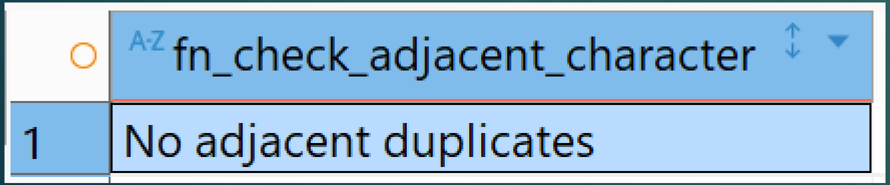
```sql
-- 5. Cleaned Pan Numbers

SELECT DISTINCT UPPER(TRIM(pan_numbers))
FROM stg_pan_num_dataset spnd
WHERE TRIM(pan_numbers) != ''
LIMIT 10;
```



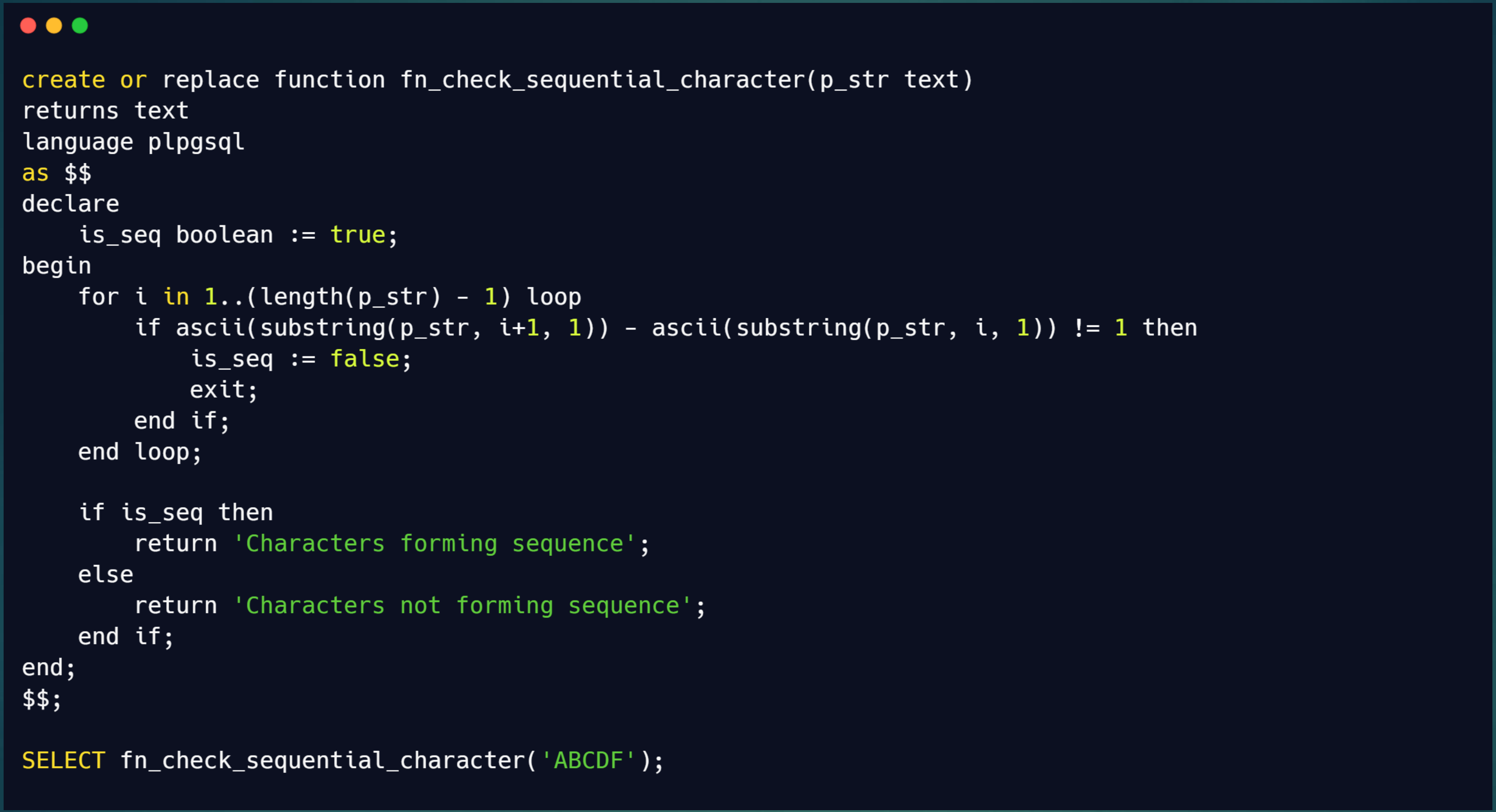
	upper
1	WUFAR0132H
2	DNRGI2432Q
3	UCYZV9250R
4	IVIDN1081H
5	AFMVC1413D
6	PIHOQ0368S
7	WOUCP7730E
8	XDXQX7884O
9	DTFPR5725T
10	EZL2951



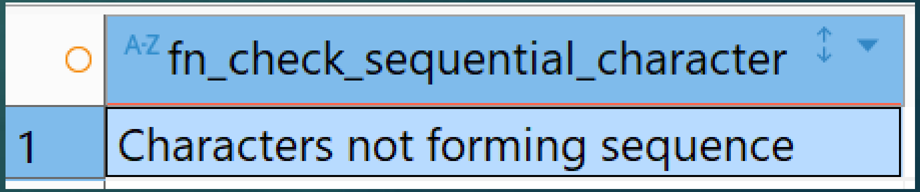
```
-- Function to check if adjacent characters are same or not

create or replace function fn_check_adjacent_character(p_str text)
returns text
language plpgsql
as $$
begin
    for i in 1.. (length(p_str) - 1) loop
        if substring(p_str, i, 1) = substring(p_str, i+1, 1) then
            return 'Has adjacent duplicates';
        end if;
    end loop;
    return 'No adjacent duplicates';
end;
$$;

SELECT fn_check_adjacent_character('ABDEF')
```



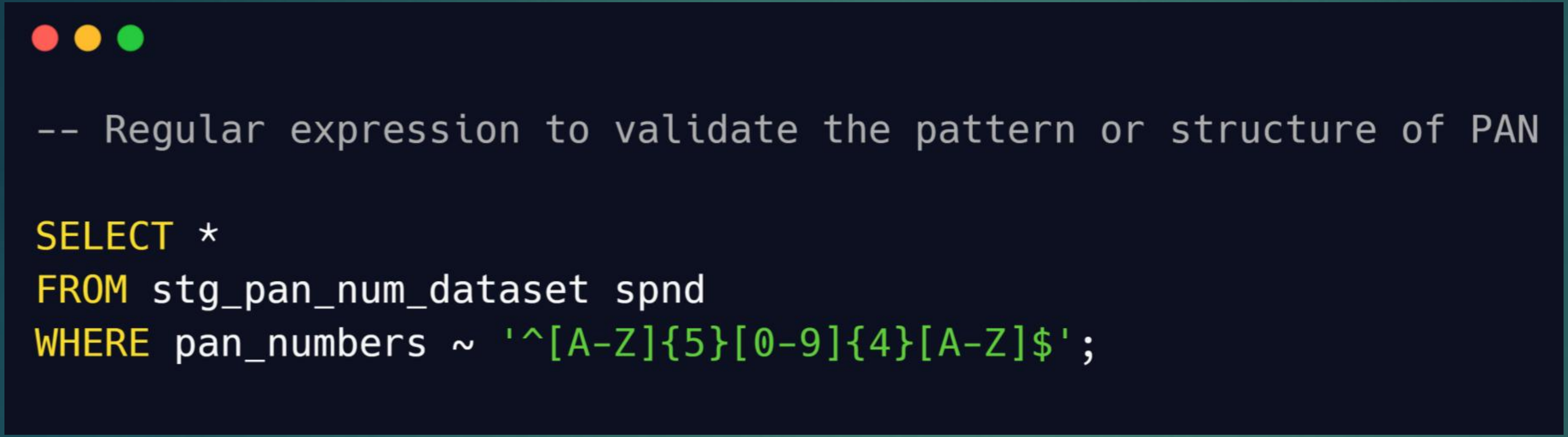
	fn_check_adjacent_character
1	No adjacent duplicates



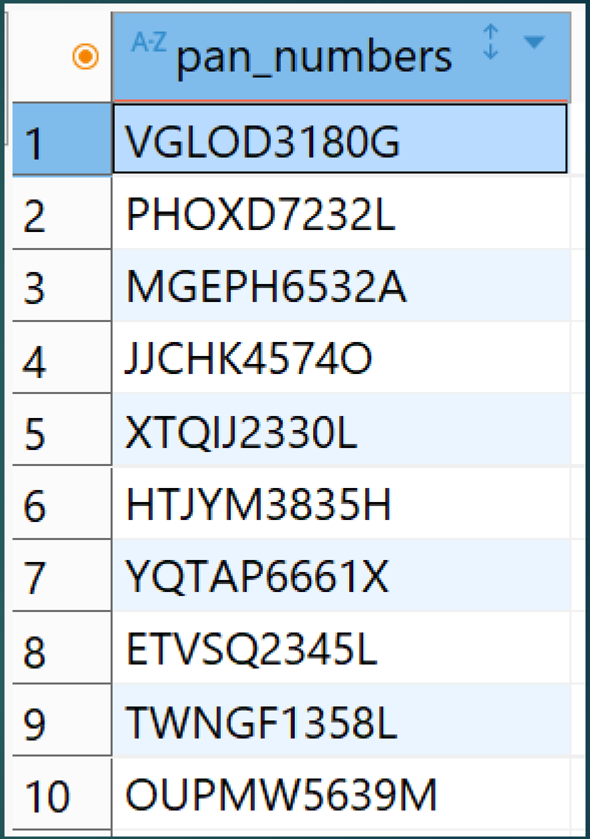
```
create or replace function fn_check_sequential_character(p_str text)
returns text
language plpgsql
as $$
declare
    is_seq boolean := true;
begin
    for i in 1..(length(p_str) - 1) loop
        if ascii(substring(p_str, i+1, 1)) - ascii(substring(p_str, i, 1)) != 1 then
            is_seq := false;
            exit;
        end if;
    end loop;

    if is_seq then
        return 'Characters forming sequence';
    else
        return 'Characters not forming sequence';
    end if;
end;
$$;

SELECT fn_check_sequential_character('ABCDF');
```



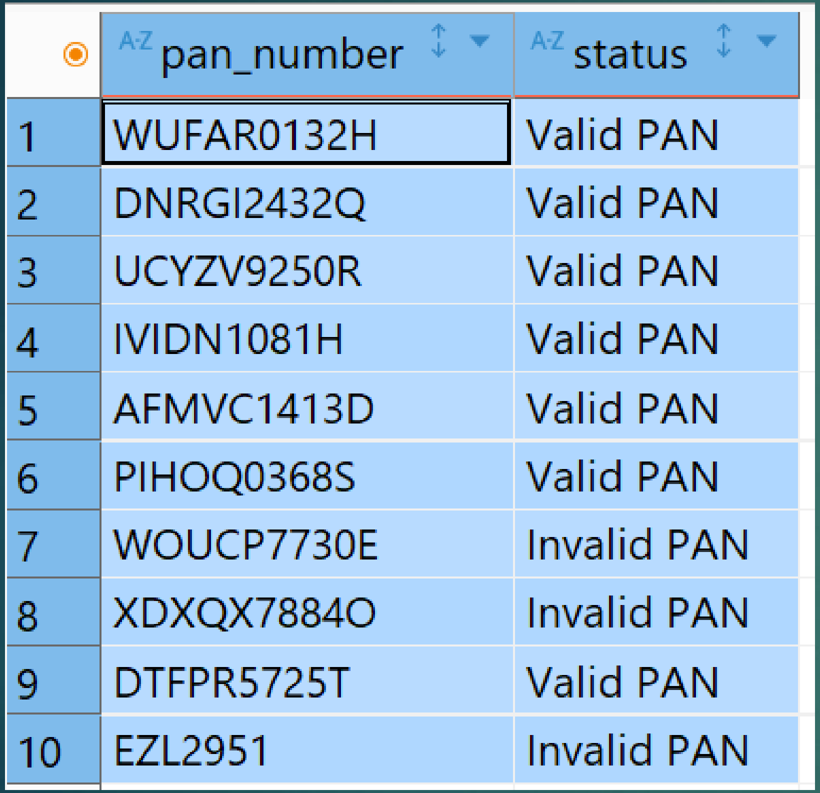
	fn_check_sequential_character
1	Characters not forming sequence



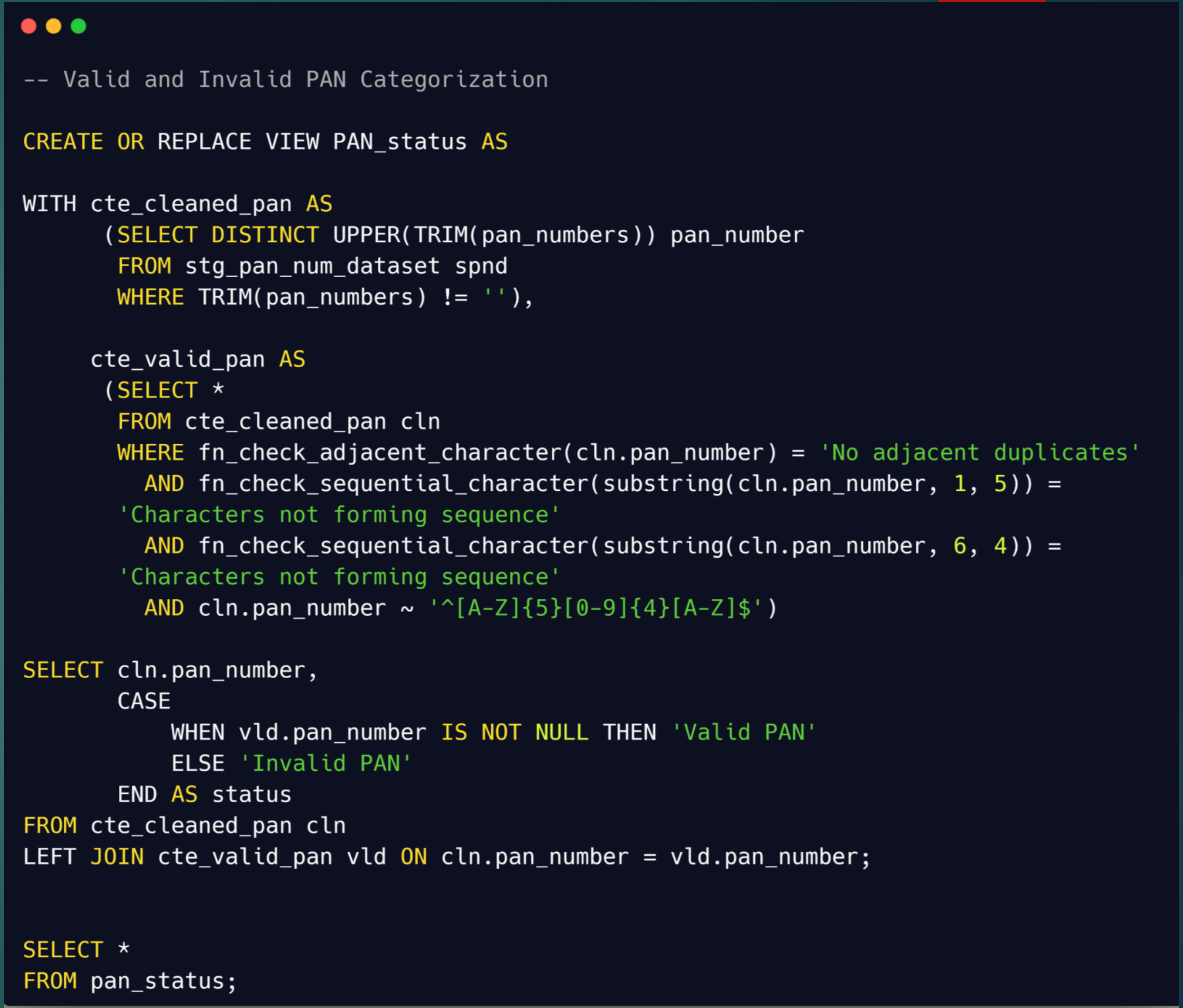
```sql
-- Regular expression to validate the pattern or structure of PAN

SELECT *
FROM stg_pan_num_dataset spnd
WHERE pan_numbers ~ '^[A-Z]{5}[0-9]{4}[A-Z]$';
```



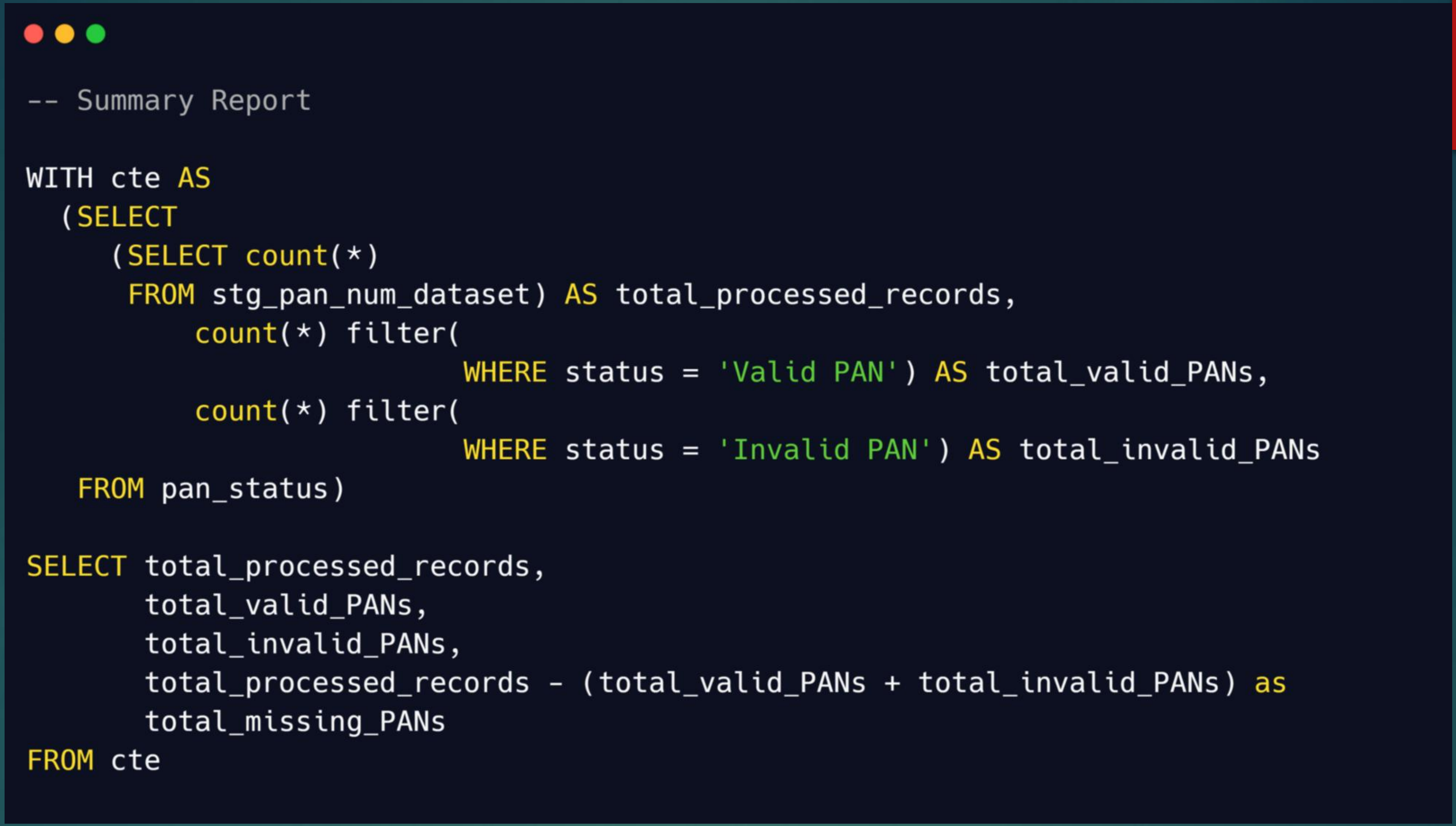
	pan_numbers
1	VGLOD3180G
2	PHOXD7232L
3	MGEPH6532A
4	JJCHK4574O
5	XTQIJ2330L
6	HTJYM3835H
7	YQTAP6661X
8	ETVSQ2345L
9	TWNGF1358L
10	OUPMW5639M



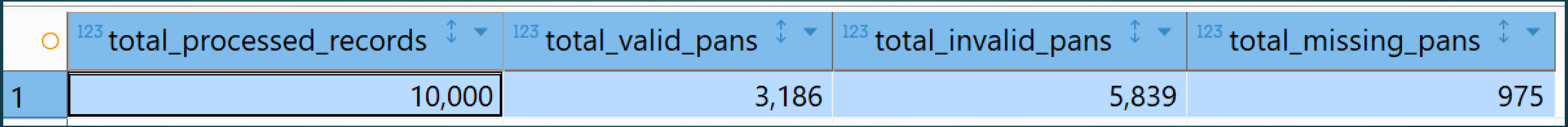
	pan_number	status
1	WUFAR0132H	Valid PAN
2	DNRGI2432Q	Valid PAN
3	UCYZV9250R	Valid PAN
4	IVIDN1081H	Valid PAN
5	AFMVC1413D	Valid PAN
6	PIHOQ0368S	Valid PAN
7	WOUCP7730E	Invalid PAN
8	XDXQX7884O	Invalid PAN
9	DTFPR5725T	Valid PAN
10	EZL2951	Invalid PAN



```
-- Valid and Invalid PAN Categorization

CREATE OR REPLACE VIEW PAN_status AS

WITH cte_cleaned_pan AS
      (SELECT DISTINCT UPPER(TRIM(pan_numbers)) pan_number
       FROM stg_pan_num_dataset spnd
       WHERE TRIM(pan_numbers) != ''),

     cte_valid_pan AS
      (SELECT *
       FROM cte_cleaned_pan cln
       WHERE fn_check_adjacent_character(cln.pan_number) = 'No adjacent duplicates'
         AND fn_check_sequential_character(substring(cln.pan_number, 1, 5)) =
       'Characters not forming sequence'
         AND fn_check_sequential_character(substring(cln.pan_number, 6, 4)) =
       'Characters not forming sequence'
         AND cln.pan_number ~ '^[A-Z]{5}[0-9]{4}[A-Z]$')

SELECT cln.pan_number,
       CASE
           WHEN vld.pan_number IS NOT NULL THEN 'Valid PAN'
           ELSE 'Invalid PAN'
       END AS status
FROM cte_cleaned_pan cln
LEFT JOIN cte_valid_pan vld ON cln.pan_number = vld.pan_number;


SELECT *
FROM pan_status;
```



```sql
-- Summary Report

WITH cte AS
  (SELECT
     (SELECT count(*)
      FROM stg_pan_num_dataset) AS total_processed_records,
          count(*) filter(
                         WHERE status = 'Valid PAN') AS total_valid_PANs,
          count(*) filter(
                         WHERE status = 'Invalid PAN') AS total_invalid_PANs
   FROM pan_status)

SELECT total_processed_records,
       total_valid_PANs,
       total_invalid_PANs,
       total_processed_records - (total_valid_PANs + total_invalid_PANs) as
       total_missing_PANs
FROM cte
```



	total_processed_records	total_valid_pans	total_invalid_pans	total_missing_pans
1	10,000	3,186	5,839	975

Analysis Summary

The dataset consisted of **10,000 PAN records**, and the validation process produced the following results:

- **Total Records Processed:** 10,000
- **Total Valid PANs:** 3,186 (31.86%)
- **Total Invalid PANs:** 5,839 (58.39%)
- **Total Missing or Incomplete PANs:** 975 (9.75%)

Conclusion

The project successfully implemented a **comprehensive PAN validation system**, combining data preprocessing with multi-layered rule-based checks using SQL, regular expressions, and custom functions.

The results reveal **significant data quality issues**: only about one-third of the records were valid, while the majority were either invalid or incomplete. This highlights the **critical importance of validation processes** to maintain data integrity before using the dataset for analytical or official purposes.

Additionally, the creation of the `PAN_status` **SQL view** provides a **reusable and efficient tool** for ongoing validation, enabling systematic monitoring and management of PAN data quality in the future.