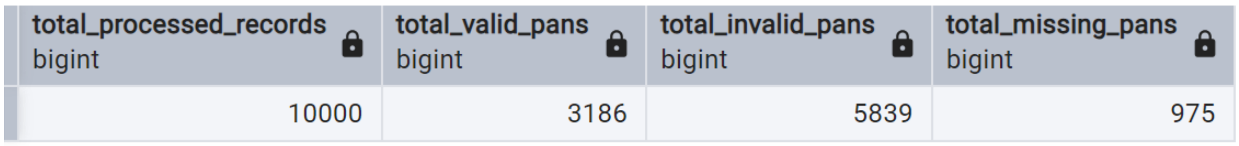
```sql
--- Pan Number Validation Project using SQL ---
CREATE OR REPLACE TABLE stg_pan_numbers_dataset
(
pan_number text
);

SELECT *
FROM stg_pan_numbers_dataset;

-- Data Cleaning and Preprocessing
---Identify and handle missing data:
SELECT *
FROM stg_pan_numbers_dataset WHERE pan_number IS NULL;

--Check for duplicates
SELECT pan_number,COUNT(*)
FROM stg_pan_numbers_dataset
GROUP BY pan_number
HAVING COUNT(*)>1;

-- Handling leading/trailing spaces
SELECT *
FROM stg_pan_numbers_dataset
WHERE pan_number <> trim(pan_number);

-- Correct letter case
SELECT *
FROM stg_pan_numbers_dataset
WHERE pan_number <> UPPER(pan_number);

-- Cleaned Pan Numbers
SELECT DISTINCT UPPER(trim(pan_number)) AS pan_number
FROM stg_pan_numbers_dataset
WHERE pan_number IS NOT NULL AND trim(pan_number) <> '';

-- Function to check if adjacent characters are the same -- ZWOVO3987M ==> ZWOVO
CREATE OR REPLACE FUNCTION fn_check_adjacent_characters(p_str text)
returns boolean
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 true; -- the characters are adjacent
        end if;
    end loop;
    return false; --  non of the characters adjacent to each other were the same
end;
$$

-- Function to check if sequential characters are used
CREATE OR REPLACE FUNCTION fn_check_sequential_characters(p_str text)
returns boolean
language plpgsql
as $$
begin
    for i in 1 .. (length(p_str) - 1)
    loop
        if ASCII(substring(p_str,i+1,1)) - ASCII(substring(p_str,i,1)) <> 1
        then
           return false; -- string does not form the sequence
        end if;
    end loop;
    return true; --  the string is forming a sequence
end;
$$

-- Regular expression to validate the pattern or structure of PAN Numbers
SELECT *
FROM stg_pan_numbers_dataset
WHERE pan_number ~ '^[A-Z]{5}[0-9]{4}[A-Z]$';

-- Valid and Invalid PAN categorization
CREATE OR REPLACE VIEW vw_valid_invalid_pans
AS
WITH cte_cleaned_pan AS (
SELECT DISTINCT UPPER(trim(pan_number)) AS pan_number
FROM stg_pan_numbers_dataset
WHERE pan_number IS NOT NULL AND trim(pan_number) <> ''),
cte_valid_pans AS (
SELECT *
FROM cte_cleaned_pan
WHERE fn_check_adjacent_characters(pan_number)= false
and fn_check_sequential_characters(substring(pan_number,1,5)) = false
and fn_check_sequential_characters(substring(pan_number,6,4)) = false
and 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_pans vld ON vld.pan_number = cln.pan_number;

-- Summary Report
WITH cte AS (
SELECT (SELECT COUNT(*) FROM stg_pan_numbers_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 vw_valid_invalid_pans)
SELECT *,
total_processed_records - (total_valid_pans + total_invalid_pans) AS total_missing_pans
FROM cte;
```



total_processed_records bigint	total_valid_pans bigint	total_invalid_pans bigint	total_missing_pans bigint
10000	3186	5839	975