

Final Project Report

1. Project Introduction

In this project, I assume the role of a data analyst for an e-commerce company that collects feedback from multiple sources: surveys (CSV), web forms (JSON), and external platforms (XML).

The goal is to parse, clean, and integrate this data to extract product performance and customer satisfaction insights.

2. Design Process

The data was collected in three formats: CSV for survey responses, JSON for web form submissions, and XML for external reviews.

Each data source was processed separately and then integrated into SQL Server.

The database schema included the following tables:

- SurveyFeedback
- WebFeedback
- ExternalReviews

Each table stores customer_id, rating, and comments. SurveyFeedback also includes review_date.

Table Name	Attribute Name	Data Type	Key Type
SurveyFeedback	customer_id	INT	/
	name	NVARCHAR(100)	/
	email	NVARCHAR(100)	/
	region	NVARCHAR(50)	/
	rating	INT	/
	comments	NVARCHAR(255)	/
	review_date	DATE	/
WebFeedback	customer_id	INT	/
	rating	INT	/
	comments	NVARCHAR(255)	/
ExternalReviews	customer_id	INT	/
	rating	INT	/
	comments	NVARCHAR(255)	/

3. Data Integration and Cleaning

3.1 CSV (Survey Data)

- Loaded using pandas in Google Colab.
- Cleaned by dropping rows with missing customer_id, rating, or review_date.
- Converted review_date to datetime format and validated ratings to ensure values are between 1–5.
- Saved as a cleaned CSV and bulk-inserted into SQL Server.

3.2 JSON (Web Feedback)

- Parsed using Python's json module and converted to DataFrame.
- Filtered out invalid or missing customer_id and ratings.
- Saved as CSV and imported into SQL Server.

3.3 XML (External Reviews)

- Parsed using xml.etree.ElementTree.
- Extracted customer_id, rating, and comments.
- Validated ratings and logged any parsing errors.
- Cleaned data was saved and inserted into SQL Server.

3.4 Data Validation

- Duplicates were checked using GROUP BY and COUNT queries.
- Ratings were validated using range checks.
- Dates in SurveyFeedback were validated using TRY_CAST.

4. Analysis Results

4.1 Top-Rated Feedback

- Combined all ratings across sources using UNION ALL.
- Calculated average rating per customer_id.
- Identified top 10 customers by average rating.

4.2 Common Complaints

- Used SQL LIKE queries to search for keywords such as 'damaged', 'late', and 'defective' in the comments.

- Counted occurrences across all feedback tables.

4.3 Sentiment Analysis

- Classified reviews as Positive (rating ≥ 4), Neutral (rating = 3), or Negative (rating < 3).
- Applied CASE logic in SQL for classification.

4.4 Trends Over Time

- Used SurveyFeedback's review_date to group ratings by month.
- Calculated average rating and total reviews per month to monitor trends.

4.5 Product Categories

- Product category analysis is not currently possible due to lack of product_id/category data.
- Recommended collecting category info for future insights using JOINS with a Products table.

Note: Some analysis outputs appeared empty or limited because the dataset contains only 3 rows across all sources. For more meaningful insights and visualizations, a larger volume of customer feedback data is required.

Final Project Part 2

(Afnan Madi)

4. Data Parsing

1. Parse CSV File: customer_survey.csv.

This script reads customer survey data from CSV, validates dates and ratings, removes missing/invalid entries, and exports the cleaned data for database import.

```
[4] import pandas as pd

# Load CSV data
csv_path = "customer_survey.csv"
survey_df = pd.read_csv(csv_path)
```

```
0s # Preview data
print("Original CSV Data:")
print(survey_df.head())
```

Original CSV Data:

	customer_id	name	email	region	rating	\
0	1	John Doe	john.doe@example.com	North	5	
1	2	Jane Smith	jane.smith@example.com	West	4	
2	3	Emily Davis	emily.davis@example.com	South	3	

	comments	review_date
0	Great product!	2024-01-01
1	Fast delivery.	2024-01-02
2	Average quality.	2024-01-03

+ Code

+ Text

```
0s # Drop rows with missing values in key fields
survey_df.dropna(subset=['customer_id', 'rating', 'review_date'], inplace=True)

# Convert date column to datetime
survey_df['review_date'] = pd.to_datetime(survey_df['review_date'], errors='coerce')

# Filter out invalid ratings
survey_df = survey_df[survey_df['rating'].between(1, 5)]
```

```
0s import os

# Create directory if it doesn't exist
os.makedirs('/mnt/data', exist_ok=True)

# Then save the file
clean_csv_path = "/mnt/data/cleaned_customer_survey.csv"
survey_df.to_csv(clean_csv_path, index=False)

print("\nCleaned CSV saved for bulk insert.")
```



Cleaned CSV saved for bulk insert.

2. Parse JSON File: web_feedback.json.

Parsed and flattened web feedback JSON. Ensured valid ratings, removed incomplete rows, and saved for SQL Server loading.

✓
0s



```
import json

# Load JSON data
json_path = "web_feedback.json"
with open(json_path, 'r') as file:
    web_feedback = json.load(file)

# Convert to DataFrame
web_df = pd.DataFrame(web_feedback)

# Clean data
web_df = web_df.dropna(subset=['customer_id', 'rating'])
web_df = web_df[web_df['rating'].between(1, 5)]

# Save for SQL import
clean_json_path = "/mnt/data/cleaned_web_feedback.csv"
web_df.to_csv(clean_json_path, index=False)
print("Cleaned JSON data saved as CSV for import.")
```



Cleaned JSON data saved as CSV for import.

3. Parse XML File: external_reviews.xml.

Used ElementTree to parse XML reviews, extracted and validated each review, and exported the clean data for SQL Server import.



```
import xml.etree.ElementTree as ET

# Load and parse XML
xml_path = "external_reviews.xml"
tree = ET.parse(xml_path)
root = tree.getroot()

# Extract data
xml_reviews = []
for review in root.findall('review'):
    try:
        review_data = {
            'customer_id': int(review.find('customer_id').text),
            'rating': int(review.find('rating').text),
            'comments': review.find('comments').text
        }
        if 1 <= review_data['rating'] <= 5:
            xml_reviews.append(review_data)
    except Exception as e:
        print(f"Error parsing review: {e}")

# Convert to DataFrame
xml_df = pd.DataFrame(xml_reviews)


# Save to CSV
clean_xml_path = "/mnt/data/cleaned_external_reviews.csv"
xml_df.to_csv(clean_xml_path, index=False)
print("Cleaned XML data saved as CSV.")
```

Validate parsed data

Create Tables (SurveyFeedback , WebFeedback, and ExternalReviews)

```
CREATE TABLE SurveyFeedback (  
    customer_id INT,  
    name NVARCHAR(100),  
    email NVARCHAR(100),  
    region NVARCHAR(50),  
    rating INT,  
    comments NVARCHAR(255),  
    review_date DATE  
);  
  
CREATE TABLE WebFeedback (  
    customer_id INT,  
    rating INT,  
    comments NVARCHAR(255)  
);  
  
CREATE TABLE ExternalReviews (  
    customer_id INT,  
    rating INT,  
    comments NVARCHAR(255)  
);
```

100 %

 Messages

Commands completed successfully.

Completion time: 2025-05-19T11:14:08.6629691-04:00

Bulk Insert Cleaned Data into SQL Server.

```
--Bulk Insert Cleaned Data into SQL Server
BULK INSERT SurveyFeedback
FROM 'C:\Users\user\Downloads\Willis college\ETL\Assignments\finle ptojects\2\cleaned_customer_survey.csv'
WITH (
    FIRSTROW = 2,
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    CODEPAGE = '65001'
);
```

100 %

Messages

(0 rows affected)

Completion time: 2025-05-19T11:31:09.8171762-04:00

```
-- Bulk Insert Cleaned CSV
BULK INSERT WebFeedback
FROM 'C:\Users\user\Downloads\Willis college\ETL\Assignments\finle ptojects\2\cleaned_web_feedback.csv'
WITH (
    FIRSTROW = 2,
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    CODEPAGE = '65001'
);

-- Bulk Insert Cleaned CSV
BULK INSERT ExternalReviews
FROM 'C:\Users\user\Downloads\Willis college\ETL\Assignments\finle ptojects\2\cleaned_external_reviews.csv'
WITH (
    FIRSTROW = 2,
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    CODEPAGE = '65001'
);
```

100 %

Messages

(0 rows affected)

Completion time: 2025-05-19T11:38:10.1403538-04:00

Validate Parsed Data in SQL.

-SurveyFeedback.

Check for Duplicates:

```
--Validate Parsed Data in SQL.
--SurveyFeedback.

SELECT customer_id, COUNT(*) AS count
FROM SurveyFeedback
GROUP BY customer_id
HAVING COUNT(*) > 1;
```

100 %

Results Messages

customer_id	count
-------------	-------

Validate Ratings.

```
-- Validate Ratings
SELECT * FROM SurveyFeedback WHERE rating NOT BETWEEN 1 AND 5;
```

100 %

Results Messages

customer_id	name	email	region	rating	comments	review_date
-------------	------	-------	--------	--------	----------	-------------

Validate Dates.

```
--Validate Dates
SELECT * FROM SurveyFeedback WHERE TRY_CAST(review_date AS DATE) IS NULL;
```

100 %

Results Messages

customer_id	name	email	region	rating	comments	review_date
-------------	------	-------	--------	--------	----------	-------------

VALIDATION FOR WebFeedback & ExternalReviews

Check for Duplicates .

The screenshot shows the SQL Server Enterprise Manager interface. The query editor displays the following SQL code:

```
--Check for Duplicates
-- WebFeedback
SELECT customer_id, COUNT(*) AS count
FROM WebFeedback
GROUP BY customer_id
HAVING COUNT(*) > 1;

-- ExternalReviews
SELECT customer_id, COUNT(*) AS count
FROM ExternalReviews
GROUP BY customer_id
HAVING COUNT(*) > 1;
```

Below the query editor, the 'Results' tab is active, showing a table with two columns: 'customer_id' and 'count'. The table is currently empty.

customer_id	count
-------------	-------

Validate Rating Range.

```
-- Validate Rating Range
-- WebFeedback
SELECT * FROM WebFeedback
WHERE rating NOT BETWEEN 1 AND 5;

-- ExternalReviews
SELECT * FROM ExternalReviews
WHERE rating NOT BETWEEN 1 AND 5;
```

%

Results Messages

customer_id	rating	comments
-------------	--------	----------

```
-- ExternalReviews
SELECT * FROM ExternalReviews
WHERE rating NOT BETWEEN 1 AND 5;
```

%

Results Messages

customer_id	rating	comments
-------------	--------	----------

Part 5: Analysis and Presentation.

Top-Rated Products:

This query merges all ratings and calculates an average per customer , sorted by highest ratings.

```
-- Top 10 highest-rated customer feedback (assuming each customer reviewed a product)
SELECT TOP 10
    customer_id,
    AVG(rating) AS avg_rating
FROM (
    SELECT customer_id, rating FROM SurveyFeedback
    UNION ALL
    SELECT customer_id, rating FROM WebFeedback
    UNION ALL
    SELECT customer_id, rating FROM ExternalReviews
) AS CombinedRatings
GROUP BY customer_id
ORDER BY avg_rating DESC;
```

100 %

Results Messages

customer_id	avg_rating
-------------	------------

Common Complaints Using Keywords.

This shows how often certain complaint words appear in feedback, helping to pinpoint recurring issues.

```
-- Search for common complaint keywords
SELECT
    'SurveyFeedback' AS Source,
    COUNT(*) AS Count,
    'damaged' AS Keyword
FROM SurveyFeedback
WHERE comments LIKE '%damaged%'

UNION

SELECT
    'WebFeedback' AS Source,
    COUNT(*),
    'late'
FROM WebFeedback
WHERE comments LIKE '%late%'

UNION

SELECT
    'ExternalReviews' AS Source,
    COUNT(*),
    'defective'
FROM ExternalReviews
WHERE comments LIKE '%defective%';
```

100 %

Results Messages

	Source	Count	Keyword
1	ExternalReviews	0	defective
2	SurveyFeedback	0	damaged
3	WebFeedback	0	late

Customer Sentiment Classification.

This query helps in tagging each review with a sentiment label based on rating.

```
-- Classify feedback sentiment
SELECT
    customer_id,
    rating,
    comments,
    CASE
        WHEN rating >= 4 THEN 'Positive'
        WHEN rating = 3 THEN 'Neutral'
        ELSE 'Negative'
    END AS sentiment
FROM SurveyFeedback;
```

100 %

Results Messages

customer_id	rating	comments	sentiment
-------------	--------	----------	-----------

SUMMARY REPORTS.

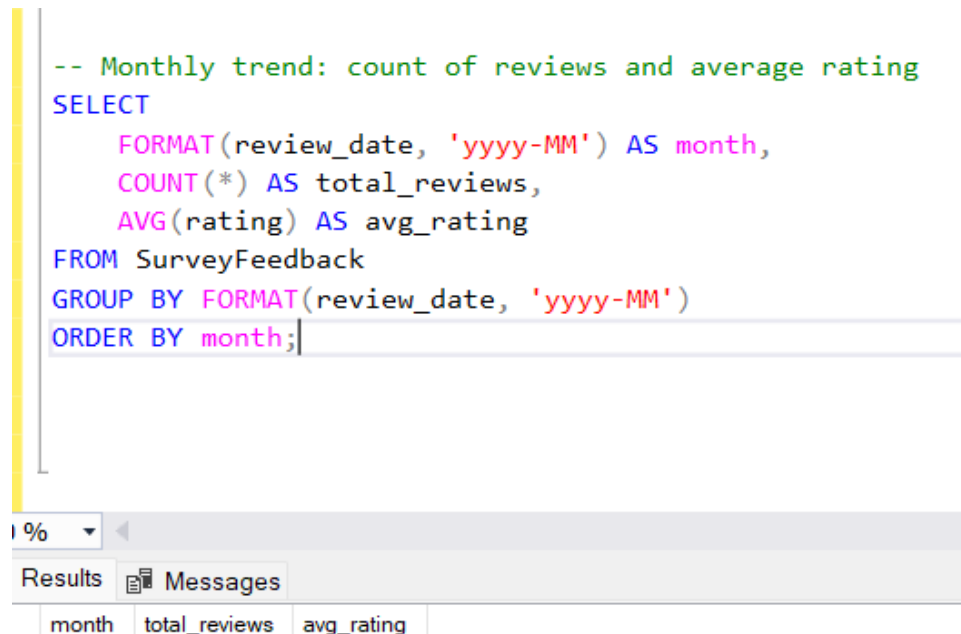
Trends Over Time in Ratings and Reviews.

These queries help visualize trends in:

- Total reviews submitted per month.
- How average satisfaction scores are changing over time

Group by Month:

```
-- Monthly trend: count of reviews and average rating
SELECT
    FORMAT(review_date, 'yyyy-MM') AS month,
    COUNT(*) AS total_reviews,
    AVG(rating) AS avg_rating
FROM SurveyFeedback
GROUP BY FORMAT(review_date, 'yyyy-MM')
ORDER BY month;
```



month	total_reviews	avg_rating
-------	---------------	------------

Product Categories with Highest Satisfaction Scores.

Currently, product category-level analysis is not possible due to the absence of category or product identifiers in the available feedback sources. For more granular insight, future data collection should include product-level metadata.