



Hotel Review Sentiment Analyzer

**Name:** AKASH T S M

Roll Number: 23BCE2029

College Name: VELLORE INSTITUTE OF TECHNOLOGY (VELLORE)

Branch: Computer Science and Engineering (Core)

Date of Submission: 05-07-2025

Table of Contents

1. Introduction

2. Objective

3. Tools & Technologies Used

4. Methodology / Working

5. Code Snippets with Explanation

6. Screenshots / Output Results

7. Project Links

8. Challenges Faced & Solutions (Optional)

9. Conclusion

10. References

**1. Introduction**

In today’s hospitality industry, online customer reviews play a critical role in shaping a hotel’s reputation and service quality. However, manually analyzing hundreds or thousands of guest reviews can be time-consuming and inconsistent. This project leverages Generative AI to automate the process of understanding customer feedback.

Using IBM Watsonx.ai and foundation models, the system reads hotel reviews and identifies their overall sentiment—Positive, Neutral, or Negative. Additionally, it extracts key aspects such as cleanliness, food quality, staff behavior, and amenities that guests mention, offering deeper insights into customer satisfaction. This enables hotel management teams to make informed, data-driven decisions and improve guest experience.

**2. Objective**

The objective is to automate the sentiment analysis and aspect-based extraction from hotel reviews to provide actionable insights to hotel managers.

**3. Tools & Technologies Used**

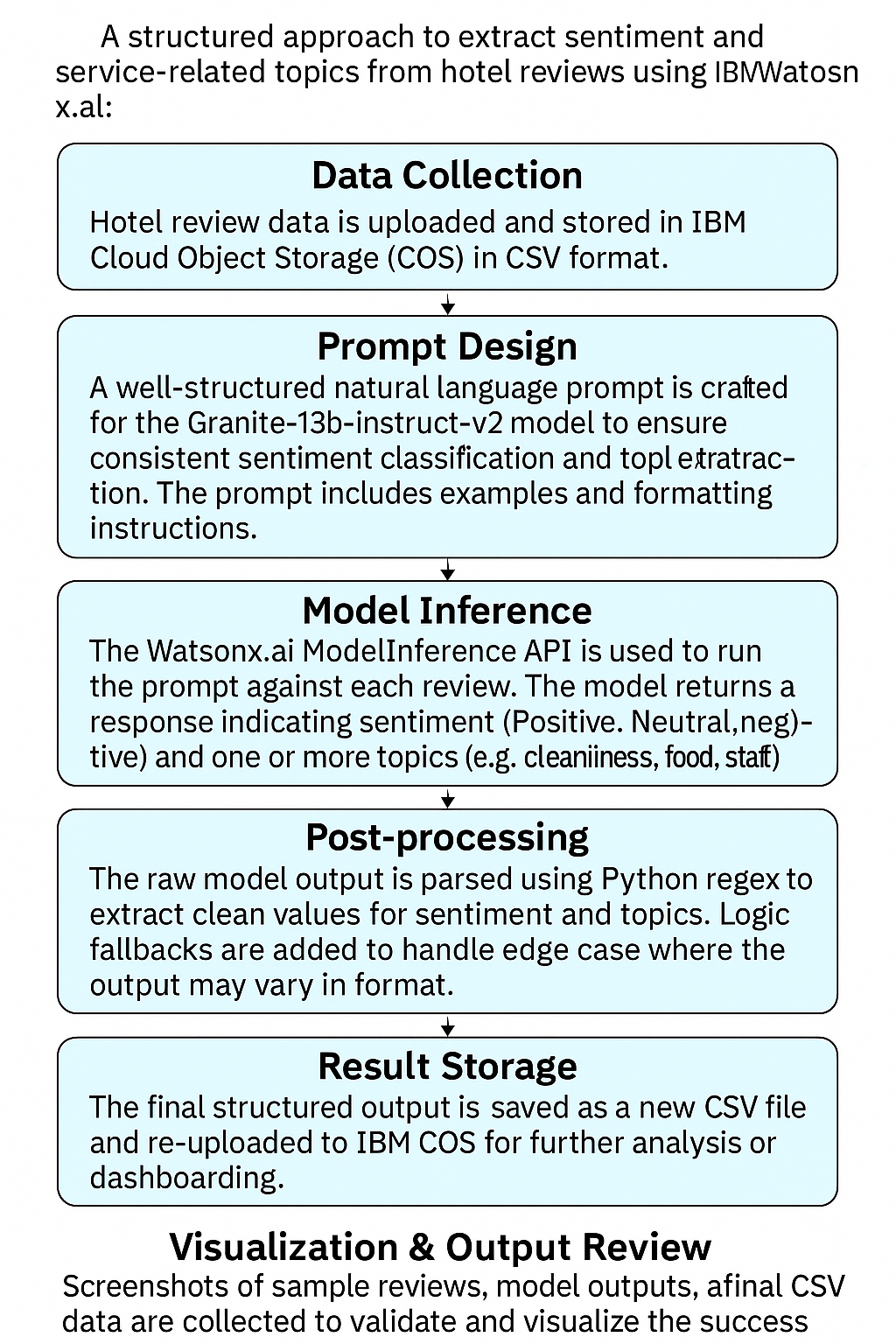
The following tools and technologies were used in the development and execution of this project:

* **IBM Watsonx.ai** – To access and prompt large foundation models like Granite-13b-instruct-v2 for text analysis tasks.
* **IBM Cloud Object Storage (COS)** – For storing the dataset (input hotel reviews) and saving the processed results (CSV output).
* **Python** – Core programming language used for data manipulation, inference, and output handling.
* **Pandas** – Python library for reading, cleaning, and managing structured data (CSV files).
* **Regular Expressions (Regex)** – To extract structured elements (sentiment and topics) from the raw model responses.
* **Jupyter Notebook**  – For developing and testing the code in an interactive notebook environment.
* **Granite-13b-instruct-v2** – The specific IBM foundation model used to interpret hotel reviews and generate insights.Top of Form

# 4. Methodology / Working

The project follows a structured approach to extract sentiment and service-related topics from hotel reviews using IBM Watsonx.ai:

1. **Data Collection**  
   Hotel review data is uploaded and stored in **IBM Cloud Object Storage (COS)** in CSV format.
2. **Prompt Design**  
   A well-structured natural language prompt is crafted for the **Granite-13b-instruct-v2** model to ensure consistent sentiment classification and topic extraction. The prompt includes examples and formatting instructions.
3. **Model Inference**  
   The **Watsonx.ai ModelInference** API is used to run the prompt against each review. The model returns a response indicating sentiment (Positive, Neutral, Negative) and one or more topics (e.g., cleanliness, food, staff).
4. **Post-processing**  
   The raw model output is parsed using **Python regex** to extract clean values for sentiment and topics. Logic fallbacks are added to handle edge cases where the output may vary in format.
5. **Result Storage**  
   The final structured output is saved as a new CSV file and re-uploaded to IBM COS for further analysis or dashboarding.
6. **Visualization & Output Review**  
   Screenshots of sample reviews, model outputs, and final CSV data are collected to validate and visualize the success of the pipeline.



5. Code Snippets with Explanation

STEP 1: Install SDK

# !pip install -q ibm-watsonx-ai ibm-boto3

🔹 This command installs the IBM Watsonx.ai and IBM Cloud Object Storage SDKs in your Python environment. These are required to interact with IBM's foundation models and cloud storage services.

**STEP 2: Imports**

import os, types, re

import pandas as pd

import ibm\_boto3

from botocore.client import Config

from ibm\_watsonx\_ai import Credentials

from ibm\_watsonx\_ai.foundation\_models import ModelInference

🔹 These import essential libraries:

* pandas for data handling
* re for regular expressions
* ibm\_boto3 and botocore for IBM COS access
* ibm\_watsonx\_ai for working with Watsonx foundation models

### **STEP 3: IBM COS Setup**

cos\_client = ibm\_boto3.client(...)

...

df = pd.read\_csv(body)

🔹 This block initializes a connection to **IBM Cloud Object Storage**, downloads the hotel review CSV file, and loads it into a Pandas DataFrame (df). This becomes the input dataset for your project.

**STEP 4: Watsonx.ai Setup**

creds = Credentials(...)

model = ModelInference(...)

🔹 This step authenticates your project with **IBM Watsonx.ai**, loads the Granite-13b-instruct-v2 foundation model, and sets inference parameters like temperature and max\_new\_tokens.

**STEP 5: Prompt Template**

def generate\_prompt(review):

return f\"""...

🔹 A custom prompt template is defined to instruct the model:

* Classify review sentiment (Positive/Neutral/Negative)
* Extract 2–3 key topics from the review  
  It includes clear formatting instructions to improve model consistency.

**STEP 6: Inference and Extraction**

for i, review in enumerate(df["review"][:20]):

...

🔹 This loop runs inference on the first 20 reviews:

* Sends the review to the model using the prompt
* Handles output formatting issues using regex and fallback logic
* Extracts sentiment and topic values
* Appends them to separate lists (sentiments, topics)

### **STEP 7: Save Output**

df\_out = df[:20].copy()

...

df\_out.to\_csv(...)

🔹 A new DataFrame (df\_out) is created containing original reviews along with predicted sentiment and topics. It is saved as a CSV file for further use or reporting.

**STEP 8: Upload to COS**

cos\_client.upload\_file(...)

🔹 The final CSV file is uploaded back to **IBM Cloud Object Storage**, ensuring the results are saved securely and can be accessed later or shared with stakeholders.

Complete Code:

# STEP 1: Install SDK

# !pip install -q ibm-watsonx-ai ibm-boto3

# STEP 2: Imports

import os, types, re

import pandas as pd

import ibm\_boto3

from botocore.client import Config

from ibm\_watsonx\_ai import Credentials

from ibm\_watsonx\_ai.foundation\_models import ModelInference

# STEP 3: IBM COS Setup

cos\_client = ibm\_boto3.client(

service\_name='s3',

ibm\_api\_key\_id='gzDTJC05ZcAFf5TqF5XaGezjFvJTRYcIfb\_GLwkpltpF',

ibm\_auth\_endpoint="https://iam.cloud.ibm.com/oidc/token",

config=Config(signature\_version='oauth'),

endpoint\_url='https://s3.us-south.cloud-object-storage.appdomain.cloud'

)

bucket = 'abucket-bg77cl645y40cp5'

object\_key = 'hotel\_reviews\_large.csv'

body = cos\_client.get\_object(Bucket=bucket, Key=object\_key)['Body']

if not hasattr(body, "\_iter"):

body.iter\_ = types.MethodType(lambda self: 0, body)

df = pd.read\_csv(body)

# STEP 4: watsonx.ai Setup

creds = Credentials(

url="https://us-south.ml.cloud.ibm.com",

api\_key="gzDTJC05ZcAFf5TqF5XaGezjFvJTRYcIfb\_GLwkpltpF"

)

model = ModelInference(

model\_id="ibm/granite-13b-instruct-v2",

credentials=creds,

project\_id="37992c12-d89d-4fa1-82c9-4aa6e9d9a89f",

params={"temperature": 0.2, "decoding\_method": "greedy", "max\_new\_tokens": 150}

)

# STEP 5: Prompt Template

def generate\_prompt(review):

return f"""

You are an AI assistant that analyzes hotel customer reviews.

Your tasks:

1. Classify the review's sentiment as one of: Positive, Neutral, or Negative.

2. Identify exactly 2–3 specific topics mentioned in the review (e.g., food, service, cleanliness, location, check-in, room, staff).

⚠ Very important: Follow \*this exact response format\*, no deviations:

Sentiment: <Positive|Neutral|Negative>

Topics: <comma-separated list of 2–3 relevant hotel-related keywords>

Example Output:

Sentiment: Positive

Topics: staff, cleanliness, room

Now analyze the following review:

'''{review}'''

Your Response:

"""

# STEP 6: Inference and Extraction (Fully Fixed Version)

sentiments = []

topics = []

for i, review in enumerate(df["review"][:20]):

prompt = generate\_prompt(review)

try:

response = model.generate(prompt=prompt)

output = response["results"][0]["generated\_text"].strip()

# Fix formatting if missing 'Sentiment:'

if "Sentiment:" not in output:

output\_lines = output.splitlines()

sentiment\_line = ""

topic\_line = ""

for line in output\_lines:

if any(word in line.lower() for word in ["positive", "negative", "neutral"]) and not sentiment\_line:

sentiment\_line = f"Sentiment: {line.strip()}"

elif any(kw in line.lower() for kw in ["staff", "room", "clean", "food", "check", "location", "spa", "wifi", "pool", "bed"]):

topic\_line = f"Topics: {line.strip()}"

output = f"{sentiment\_line}\n{topic\_line}".strip()

print(f"\n--- Output for Review {i+1} ---\n{output}\n--------------------------")

sentiment = "Unknown"

topic\_list = "Unknown"

# 1. Try to extract with proper format

sentiment\_match = re.search(r"(?i)sentiment\s\*:\s\*(positive|neutral|negative)", output)

topics\_match = re.search(r"(?i)topics\s\*:\s\*([^\n\r]+)", output)

if sentiment\_match:

sentiment = sentiment\_match.group(1).capitalize()

if topics\_match:

topic\_list = topics\_match.group(1).strip()

# 2. Fallback: check for simple comma-separated output

if sentiment == "Unknown" and topic\_list == "Unknown":

parts = [p.strip() for p in output.split(",")]

if parts:

first = parts[0].lower()

if first in {"positive", "negative", "neutral"}:

sentiment = first.capitalize()

topic\_list = ", ".join(parts[1:]) if len(parts) > 1 else "General"

else:

sentiment = "Unknown"

topic\_list = ", ".join(parts)

# 3. Final fallback: if sentiment present but no topic

if sentiment in {"Positive", "Negative", "Neutral"} and (topic\_list == "Unknown" or topic\_list.strip() == ""):

topic\_list = "General"

# 4. Handle cases where both are missing

if sentiment == "Unknown" and (topic\_list == "Unknown" or topic\_list.strip() == ""):

topic\_list = "General"

except Exception as e:

print(f"⚠ Error in Review {i+1}: {e}")

sentiment = "Unknown"

topic\_list = "General"

sentiments.append(sentiment)

topics.append(topic\_list)

# STEP 7: Save Output

df\_out = df[:20].copy()

df\_out['Predicted\_Sentiment'] = sentiments

df\_out['Predicted\_Topics'] = topics

output\_file = "hotel\_review\_sentiment\_output.csv"

df\_out.to\_csv(output\_file, index=False)

# STEP 8: Upload to COS

cos\_client.upload\_file(

Filename=output\_file,

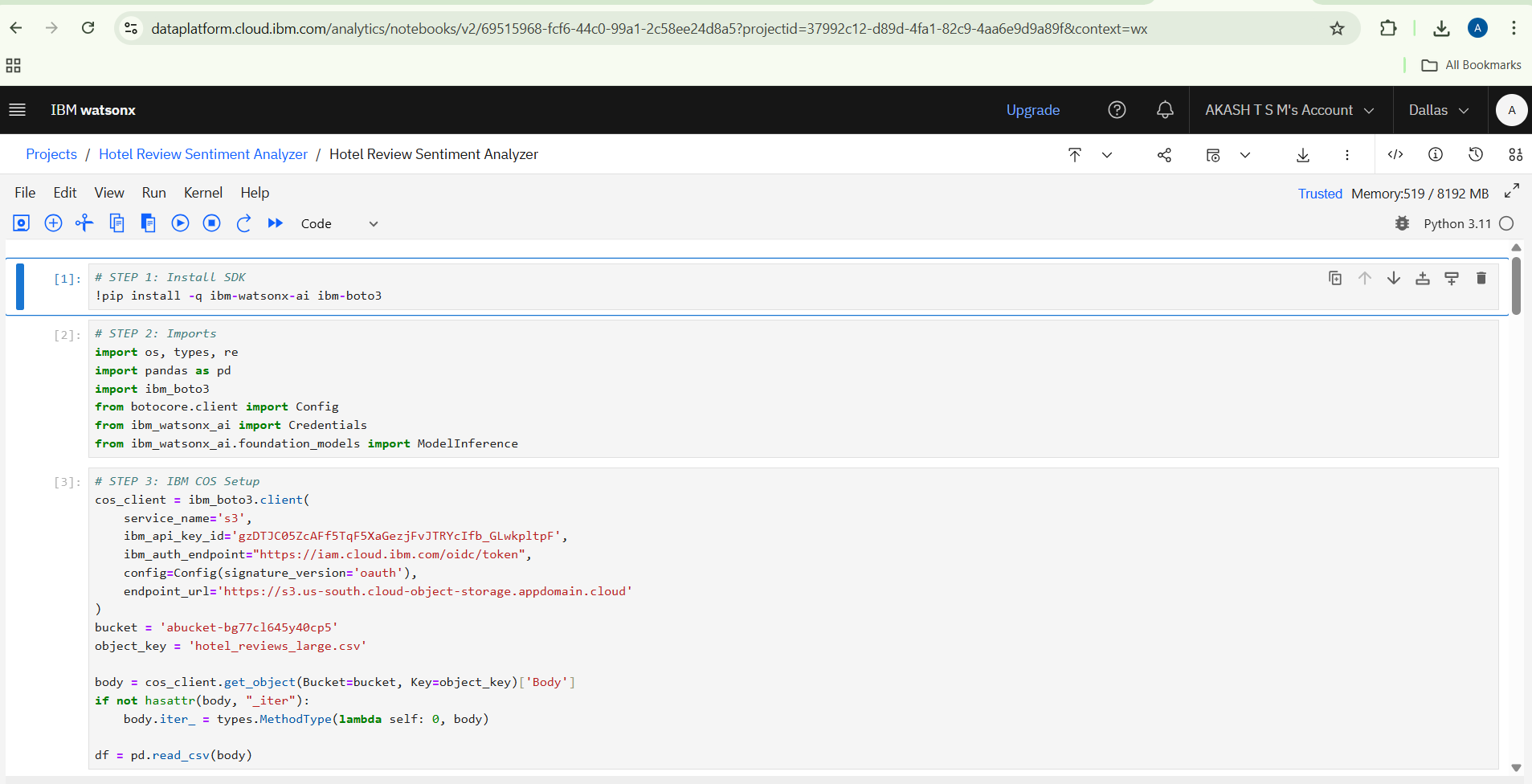
Bucket=bucket,

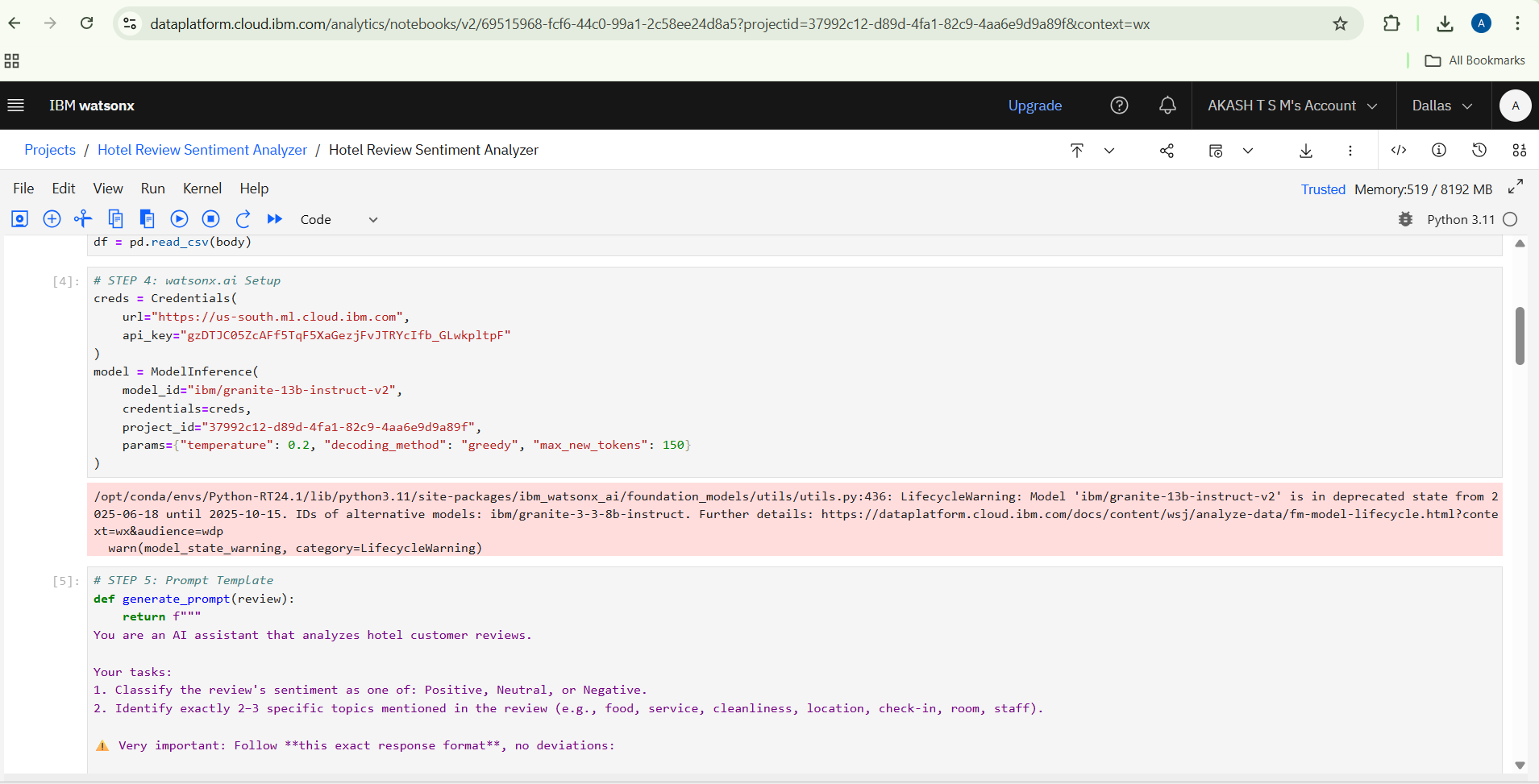
Key=output\_file

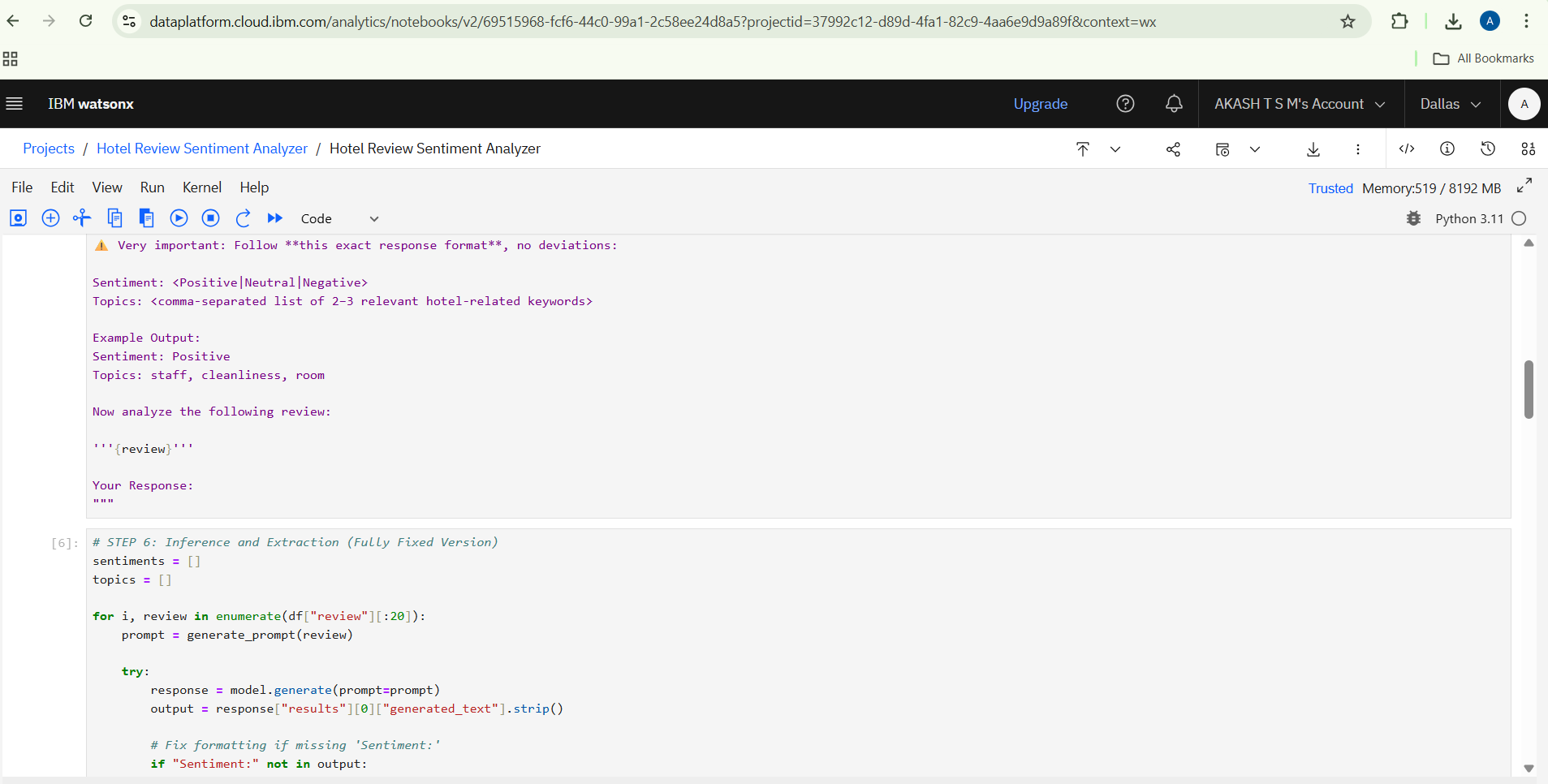
)

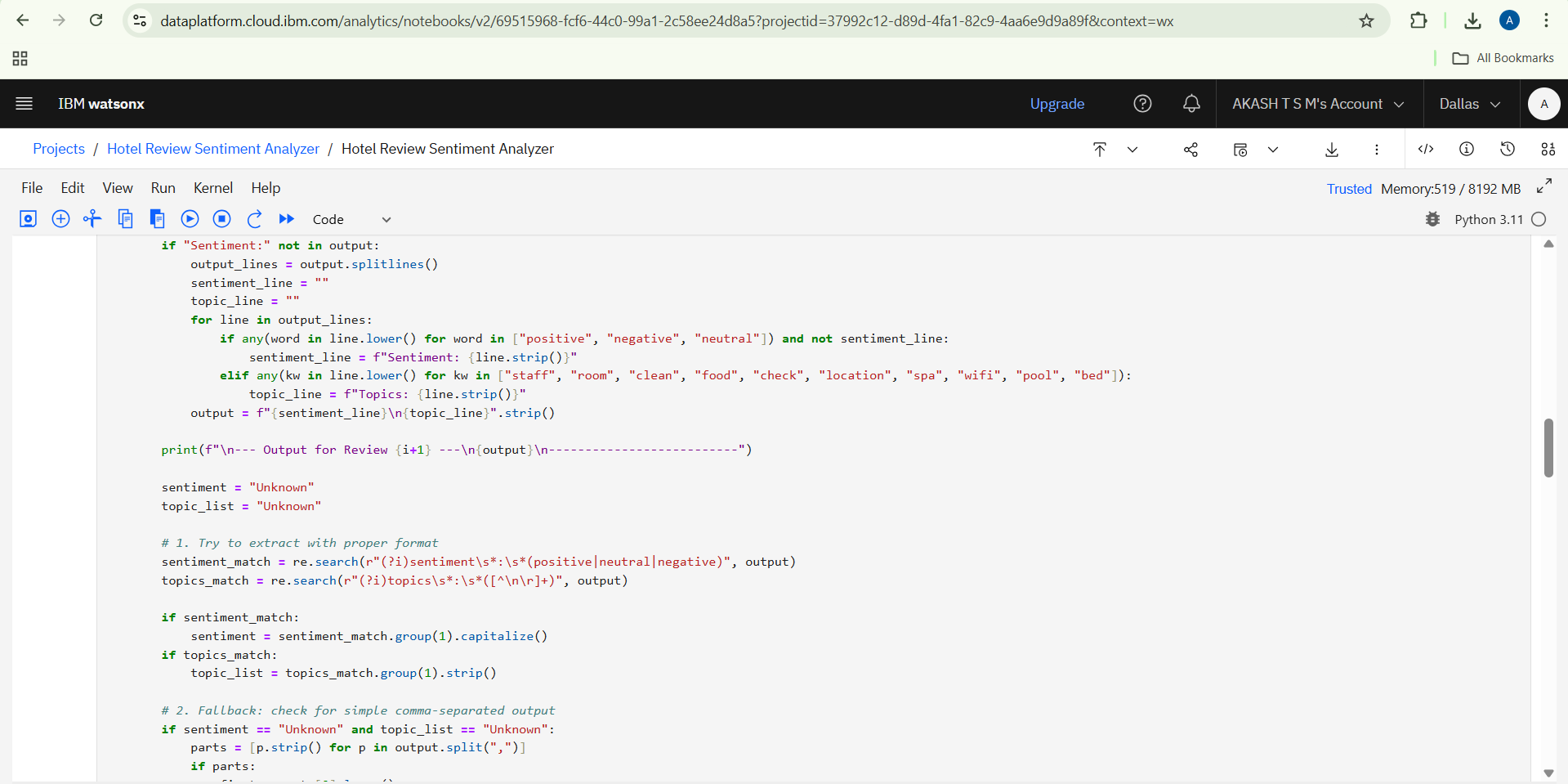
print("\n✅ Final CSV saved and uploaded to COS successfully.")

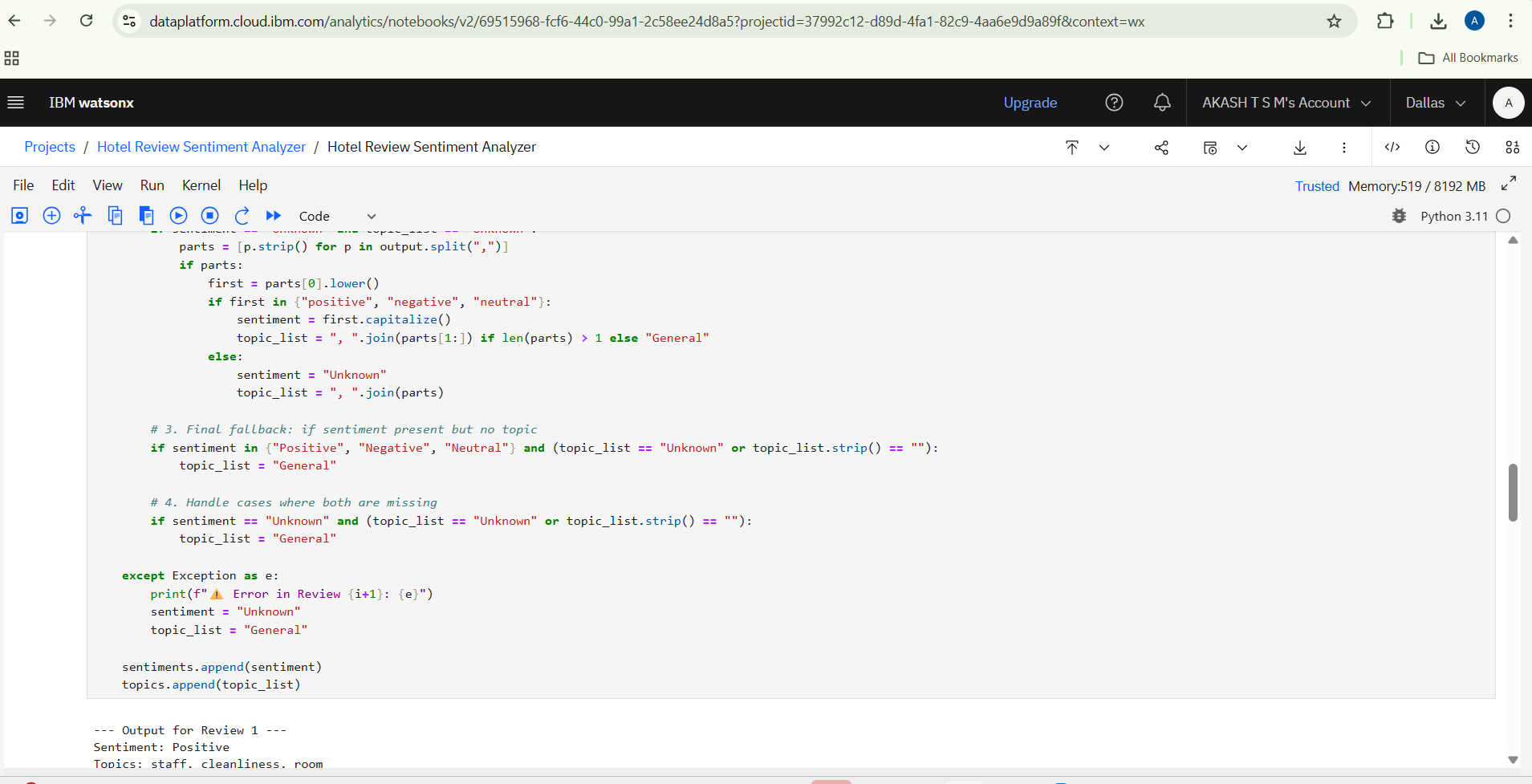
**6.Screenshots / Output Results**

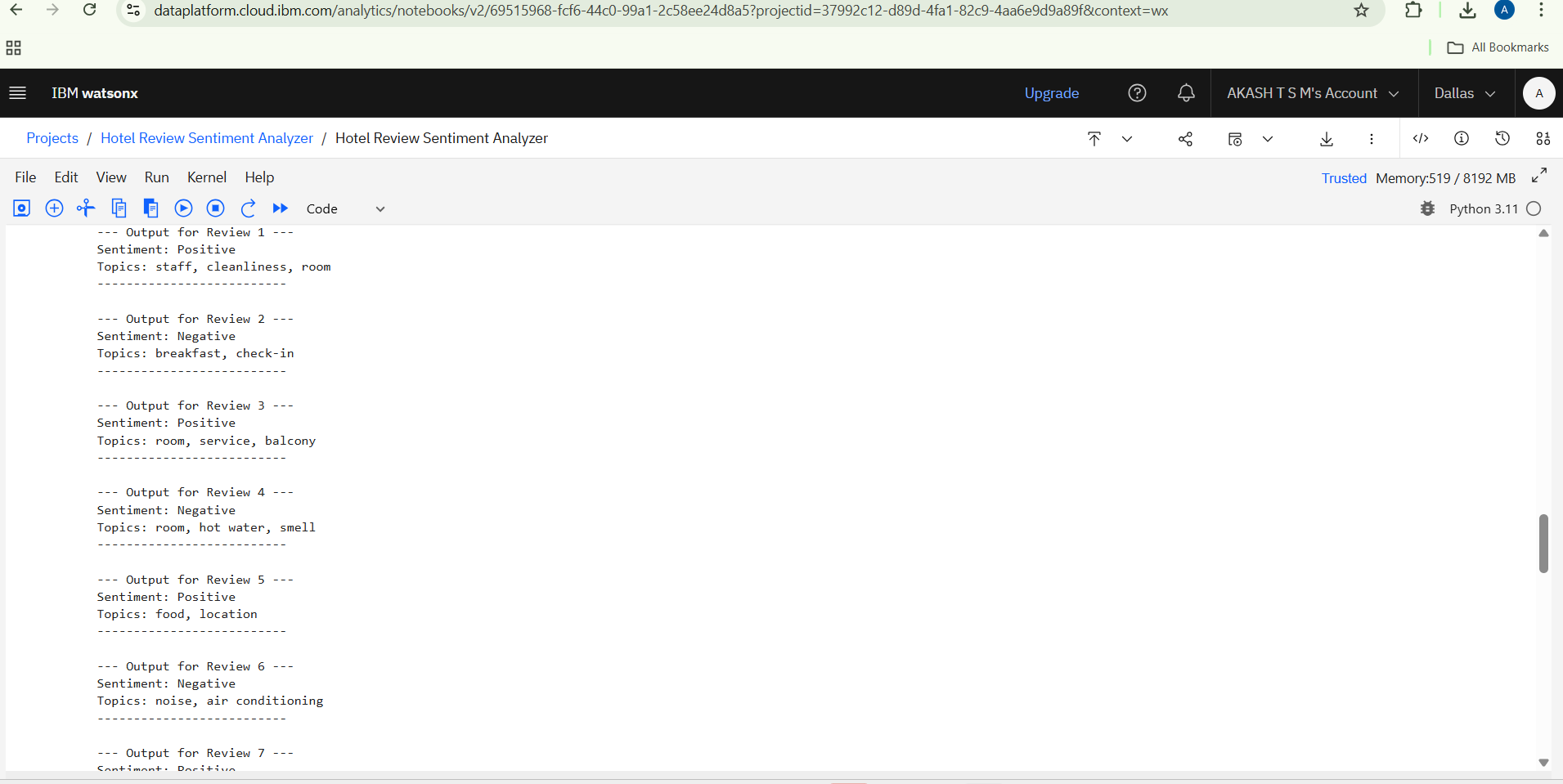
****

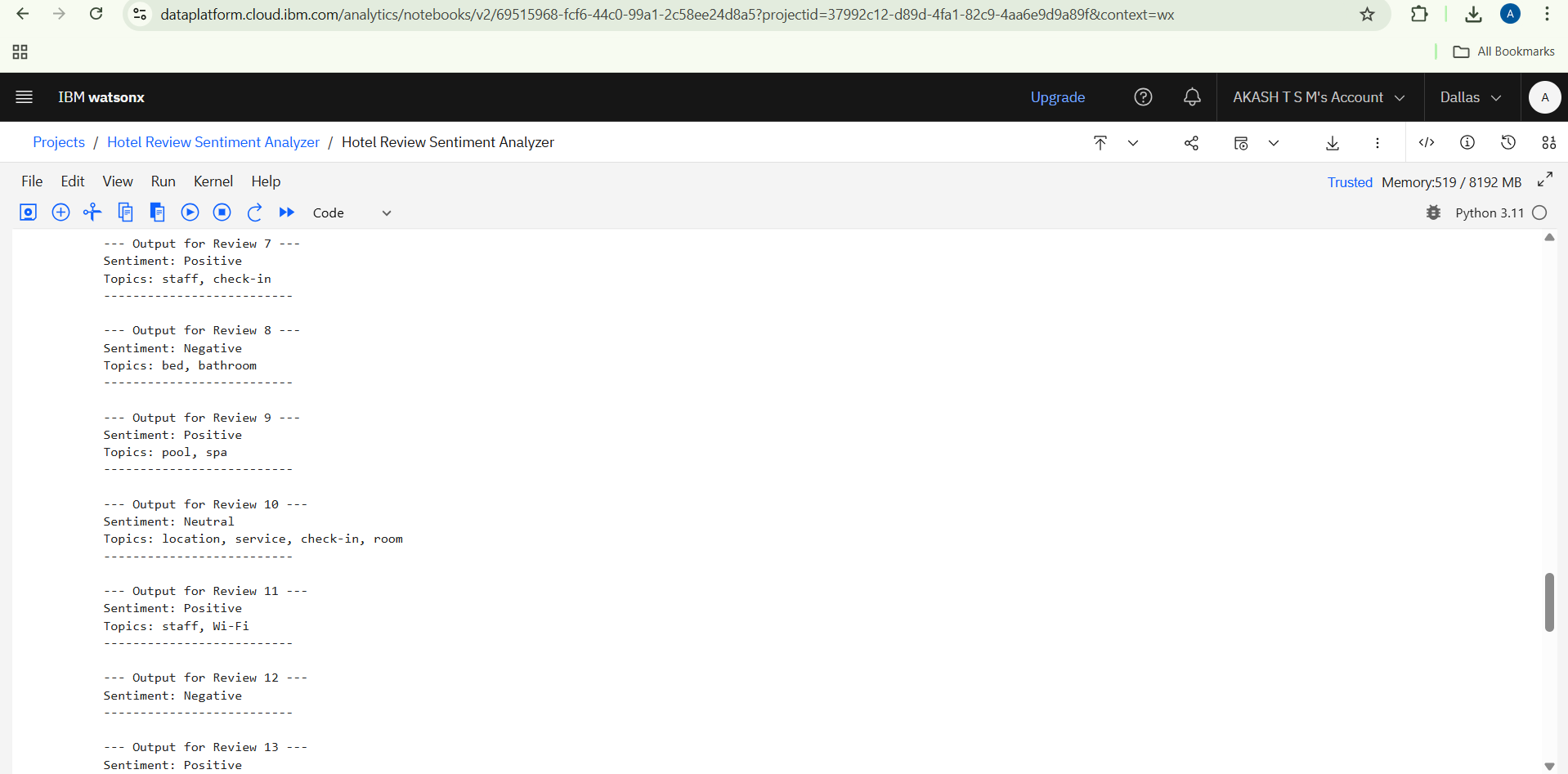
****

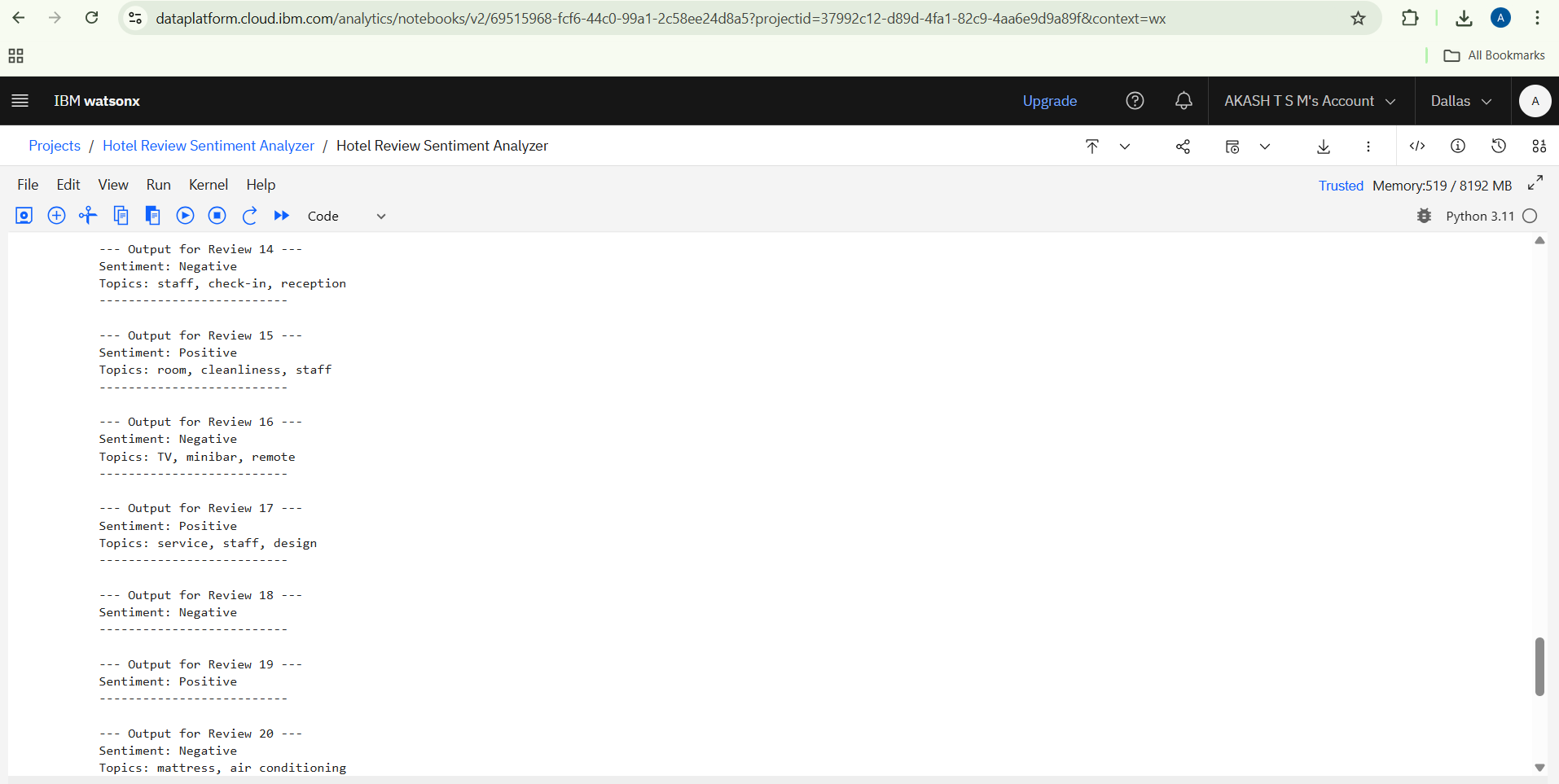
****

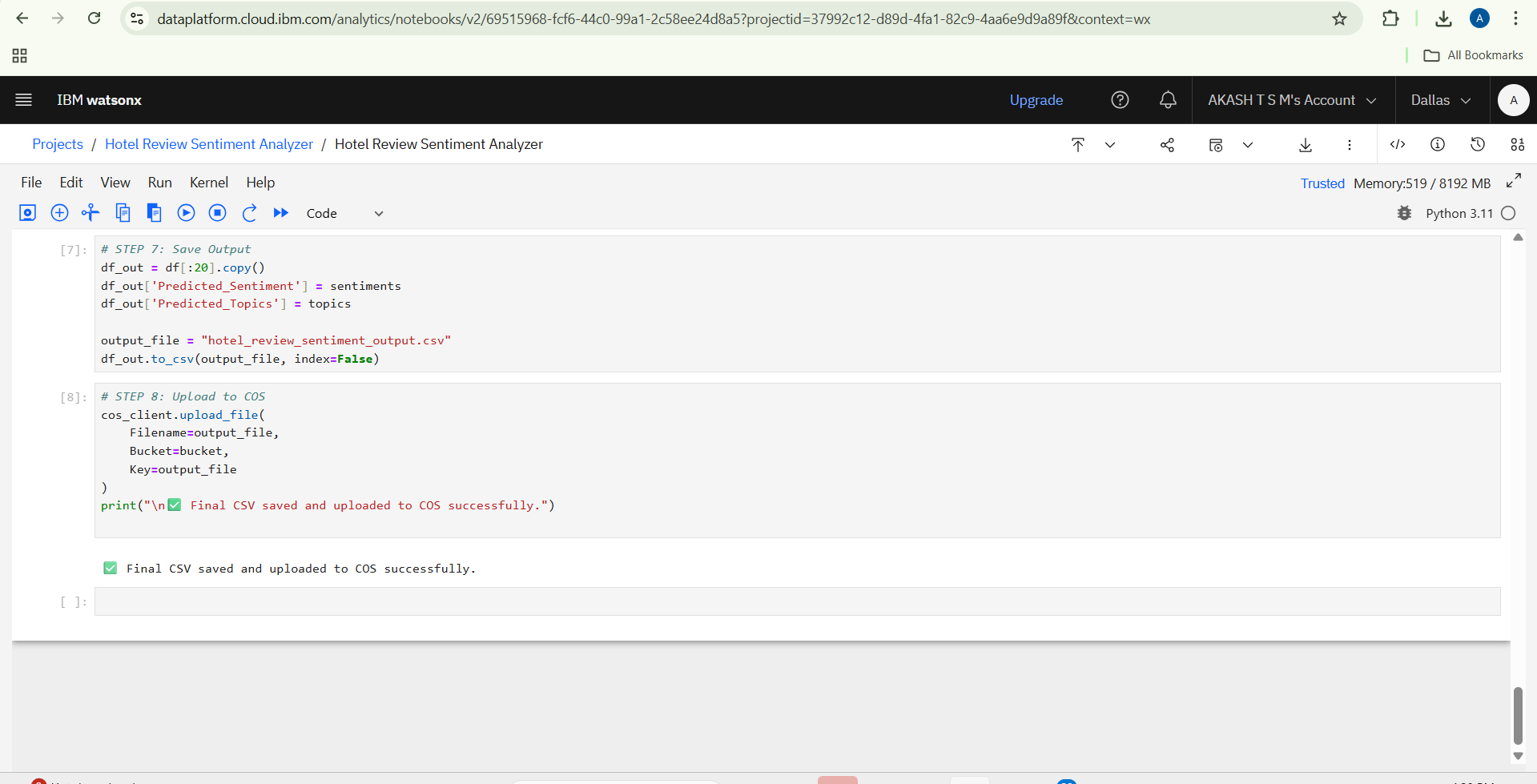
****

****

****

****

****

****

Input and Output Files :





1. **Project Links:**

[**https://github.com/ak3tsm7/hotel-review-sentiment-analyzer**](https://github.com/ak3tsm7/hotel-review-sentiment-analyzer)

**8. Challenges Faced & Solutions**

**🔹 Challenge 1: Inconsistent Model Output Format**

* **Description: The Watsonx.ai model sometimes returned responses without strict formatting (missing labels like Sentiment: or Topics:), which made it difficult to parse results directly.**
* **Solution: Added regex-based post-processing and fallback logic in the code to extract sentiment and topics even when the output format was inconsistent.**

**🔹 Challenge 2: Incorrect Topic Extraction**

* **Description: The model occasionally identified irrelevant or generic words as topics (e.g., "hotel", "place").**
* **Solution: Fine-tuned the prompt with explicit instructions and examples to guide the model toward selecting only service-related keywords like “cleanliness”, “staff”, etc.**

**🔹 Challenge 3: API Limitations with Large Data**

* **Description: Processing large datasets via Watsonx inference can be slow or exceed token limits.**
* **Solution: Limited processing to a subset of 20 reviews for demonstration and future optimization. Batched requests can be used in production for scalability.**

**🔹 Challenge 4: File Handling via IBM COS**

* **Description: Reading and uploading files using IBM Cloud Object Storage required additional setup and configuration.**
* **Solution: Successfully integrated ibm\_boto3 and tested with custom methods to download and upload CSVs programmatically.**

**9. Conclusion**

**This project successfully demonstrates the application of Generative AI using IBM Watsonx.ai to perform sentiment analysis and aspect-based extraction on hotel customer reviews. By analyzing real-world feedback, the system can classify the tone of a review (Positive, Neutral, Negative) and identify key topics such as service, cleanliness, and location.**

**The integration of IBM Cloud Object Storage (COS) with Watsonx.ai's Granite foundation model enabled a seamless pipeline from data ingestion to AI inference and result generation. Through prompt engineering and post-processing techniques, the model output was structured into actionable insights for hotel managers.**

**Overall, this project highlights the power of foundation models in automating text analytics, improving customer experience, and supporting decision-making in the hospitality industry.**

**10. References**

1. **IBM Watsonx.ai Documentation  
   https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/fm-overview.html**
2. **Granite Model on Hugging Face – granite-13b-instruct-v2  
   https://huggingface.co/ibm/granite-13b-instruct-v2**
3. **IBM Cloud Object Storage Python SDK (ibm\_boto3)**[**https://github.com/IBM/ibm-cos-sdk-python**](https://github.com/IBM/ibm-cos-sdk-python)
4. **Pandas Documentation (for data handling)  
   https://pandas.pydata.org/docs/**
5. **Python Regular Expressions – re module**[**https://docs.python.org/3/library/re.html**](https://docs.python.org/3/library/re.html)

Bottom of Form