# **Visualization of Retrospective Data from Confluence**

### **Objective**

1. Extract retrospective data from Confluence pages for a specific sprint.
2. Perform sentiment analysis on the extracted data to classify text into categories like *Teamwork*, *Communication*, *Timeline*, *Efficiency*, etc.
3. Visualize the positive and negative scores for these categories using bar graphs.

### **Requirements**

#### **Tools and Libraries**

**Python Libraries**:

* atlassian-python-api: To interact with Confluence.
* NLTK or TextBlob: For basic sentiment analysis.
* matplotlib or seaborn: For visualization.
* pandas: For data processing.

#### **Access Requirements**

* Access to the Confluence instance where retrospective data is stored.
* Confluence account credentials (username and API token) with read access to retrospective pages.

#### **Tools for Setup**

* Python installed on the local machine or server.
* An integrated development environment (IDE) like VS Code or PyCharm.

### 

### 

### 

### 

### **Steps to Develop the Script**

#### **1. Setup Confluence API**

* Authenticate and connect to the Confluence instance using the atlassian-python-api library.

from textblob import TextBlob

import re

import matplotlib.pyplot as plt

from atlassian import Confluence

import re

# Authentication

confluence = Confluence(

url="https://your-confluence-instance.atlassian.net",

username="your-email@example.com",

password="your-api-token"

)

# Fetch Data

def get\_retro\_data(page\_id):

page = confluence.get\_page\_by\_id(page\_id, expand='body.storage')

content = page['body']['storage']['value']

text = re.sub('<[^<]+?>', '', content) # Strip HTML tags

return text

#### **2. Retrieve Data**

* Identify the page or pages containing the retrospective data using a specific sprint or label.
* Extract the text content of the page(s).

#### **3. Process Data**

* Tokenize and categorize text into parameters: *Teamwork*, *Communication*, *Timeline*, *Efficiency*.
* Use regular expressions or keywords to classify text snippets.

def process\_data(content):

go\_well\_data = {category: 0 for category in categories}

not\_go\_well\_data = {category: 0 for category in categories}

achievements = []

# Process each line or sentence in the content

for line in content.splitlines():

blob = TextBlob(line)

sentiment = blob.sentiment.polarity # Range: [-1, 1]

# Check if each keyword is present in line, adjust score accordingly

for category, words in keywords.items():

if any(re.search(rf"\b{word}\b", line, re.IGNORECASE) for word in words):

if sentiment > 0:

go\_well\_data[category] += sentiment \* 10 # Scale positive points

elif sentiment < 0:

not\_go\_well\_data[category] += abs(sentiment \* 10) # Scale negative points

# Collect notable achievements or positive highlights

if sentiment > 0.5:

achievements.append(line.strip())

achievements\_text = "Achievements: " + "; ".join(achievements)

return go\_well\_data, not\_go\_well\_data, achievements\_text

#### **4. Sentiment Analysis**

* Apply sentiment analysis to calculate positive and negative scores for each category.
* **Positive Sentiment**: Polarity > 0
* **Negative Sentiment**: Polarity < 0

#### **5. Visualization**

* Use matplotlib or seaborn to create bar graphs for each category, showing positive and negative scores.

Plotting the data

def plot\_sentiment\_data(go\_well\_data, not\_go\_well\_data):

labels = list(go\_well\_data.keys())

go\_well\_values = list(go\_well\_data.values())

not\_go\_well\_values = list(not\_go\_well\_data.values())

x = range(len(labels)) # Positions for each category

# Create a bar graph with two sets of bars for each category

plt.figure(figsize=(10, 6))

plt.bar(x, go\_well\_values, width=0.4, label='Go Well', color='green', align='center')

plt.bar(x, not\_go\_well\_values, width=0.4, label='Not Go Well', color='red', align='edge')

# Labeling the graph

plt.xlabel('Categories')

plt.ylabel('Sentiment Score')

plt.title('Sentiment Analysis by Category')

plt.xticks(ticks=x, labels=labels) # Adding category labels to x-axis

plt.legend()

plt.show()

#### 

#### **6. Save or Display Output**

* Save graphs as PNG images or display them interactively.

content = """

Teamwork was good.

Communication was bad.

Deadline adherence was good overall.

Achieved faster build times, which improved project speed.

Efficiency was very good.

Better code quality this time.

"""



