

EXPT NO: 8

MINI PROJECT – CLIMATIC CHANGES ON WEATHER EVENTS

AIM

The aim of this mini project is to analyze and visualize the effects of climate change on weather patterns using data analytics and visualization frameworks through a simple web application that presents insights in an interactive and user-friendly manner.

CODE:

```
import streamlit as st
import PyPDF2
import docx
import os
import time
from numpy.random import default_rng as rng
import pandas as pd
import plotly.express as px
import textwrap
import re
from concurrent.futures import ThreadPoolExecutor
from streamlit.elements.lib.layout_utils import Height
from streamlit_mermaid import st_mermaid
import streamlit.components.v1 as components
from mermaid import generate_mermaid

@st.cache_data
def chunk_text(text, chunk_size, chunk_overlap):
```

```

"""Dummy chunk_text function"""

ifnot text: return []

return [text[i:i+chunk_size] for i in range(0, len(text), chunk_size - chunk_overlap)]


@st.cache_data
def remove_mermaid_fences(text: str) -> str:
    """Dummy remove_mermaid_fences function"""
    return text.replace(`mermaid`, "").replace(``, "").strip()

@st.cache_data
defget_cosine_scores(chunks: list[str], categories: list[str]) -> list[dict]:
    """Dummyget_cosine_scores function to generate random data"""
    results = []
    local_rng = rng()
    ifnotchunksornot categories:
        return []
    forchunkinchunks:
        res={"text":chunk}
        forcatincategories:
            res[cat]=local_rng.random()
        results.append(res)
    return results

@st.cache_data
def get_top_n_frequencies(content: str, n: int) -> pd.DataFrame:
    """
    Performs basic tokenization, stop word removal, and frequency counting
    on the document content.

```

.....

ifnot content:

```
    return pd.DataFrame({'Term': [], 'Frequency': []})
```

```
stop_words = set([
```

```
    'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours',
    'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers',
    'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves',
    'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is',
    'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
    'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
    'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about',
    'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above',
    'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
    'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when',
    'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',
    'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own',
    'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don',
    'should', 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', 'couldn',
    'didn', 'doesn', 'hadn', 'hasn', 'haven', 'isn', 'ma', 'mightn', 'mustn', 'needn',
    'shan', 'shouldn', 'wasn', 'weren', 'won', 'wouldn', 'must', 'may', 'one',
    'would', 'could', 'might', 'time', 'like', 'get'
```

```
])
```

```
processed_text = content.lower()
```

```
processed_text = re.sub(r'[^a-z\s]', ' ', processed_text)
```

```
words = processed_text.split()
```

```
filtered_words = [word for word in words if word not in stop_words and len(word) > 2]

word_counts = {}

for word in filtered_words:
    word_counts[word] = word_counts.get(word, 0) + 1

df=pd.DataFrame(
    list(word_counts.items()),
    columns=['Term', 'Frequency']
).sort_values(by='Frequency', ascending=False)

return df.head(n)

@st.cache_data
def perform_kmeans_clustering(content: str, k: int) -> pd.DataFrame:
    """
    Simulates K-Means clustering on document embeddings.
    Generates deterministic data based on the content hash and k.
    """

    if not content or k < 1:
        return pd.DataFrame()

    content_hash = hash(content)
    seed_value = abs(content_hash)
    local_rng = rng(seed_value)

    num_points = min(200, 20 + len(content) // 100)

    data=[]
```

```
cluster_centers = local_rng.uniform(-5, 5, size=(k, 2))

for i in range(num_points):
    cluster_index = local_rng.integers(0, k)
    center = cluster_centers[cluster_index]

    dim1 = center[0] + local_rng.normal(0, 0.5)
    dim2 = center[1] + local_rng.normal(0, 0.5)

    simulated_text = f"Chunk {i+1} (approx. byte {i*100}) - Content focus in Cluster {cluster_index + 1}"

    data.append({
        'Dimension 1': dim1, 'Dimension 2':
        dim2, 'Cluster': f'Cluster {cluster_index
        + 1}', 'Size': local_rng.uniform(5, 20),
        'Text Snippet': simulated_text
    })

return pd.DataFrame(data)

@st.cache_data
def get_categories_array(text:str) -> list[str]:
    return [i.strip() for i in text.split(",")]

def wrap_text(text, width=60):
    return "<br>".join(textwrap.wrap(text, width=width))
```

```
def read_txt(file):
    """Reads and returns the content of a .txt file."""
    return file.getvalue().decode("utf-8")

defread_pdf(file):
    """Reads and returns the content of a .pdf file."""
    pdf_reader = PyPDF2.PdfReader(file)
    text= ""
    forpage in pdf_reader.pages:
        ifpage.extract_text():
            text += page.extract_text()
    return text

defread_docx(file):
    """Reads and returns the content of a .docx file."""
    doc= docx.Document(file)
    text= ""
    forpara in doc.paragraphs:
        text += para.text + "\n"
    return text

st.set_page_config(layout="wide")
st.title("📄 Document Analysis")

uploaded_file = st.sidebar.file_uploader(
    "Choose a document (TXT, PDF, or DOCX)",
    type=["txt", "pdf", "docx"]
)
```

```
content = ""

CHART_HEIGHT = 350

categories = "positive,negative,neutral"

num_clusters = 3

top_n_terms = 5

if uploaded_file is not None:

    with st.sidebar:

        with st.spinner("Processing document..."):

            file_extension = os.path.splitext(uploaded_file.name)[1]

            if file_extension == ".txt":

                content = read_txt(uploaded_file)

            elif file_extension == ".pdf":

                content = read_pdf(uploaded_file)

            elif file_extension == ".docx":

                content = read_docx(uploaded_file)

    st.sidebar.success("Document loaded successfully!")

    with st.sidebar.expander("Click to view the document's content"):

        st.sidebar.text_area("Content", content, height=200)

    st.sidebar.header("2. Analysis Controls")

    st.sidebar.subheader("Sentiment Specification")

    categories = st.sidebar.text_area(label="Enter categories (comma-separated):",
                                     value="positive,negative,neutral", height=50)

    st.sidebar.subheader("Frequency Specification")
```

```
top_n_terms = st.sidebar.number_input("Top N Terms for Frequency:", min_value=1,
max_value=20, value=5)

st.sidebar.subheader("Clustering Specification")
num_clusters = st.sidebar.number_input("Number of Clusters:", min_value=1,
max_value=10, value=3)

else:
    st.sidebar.warning("Please upload a document to get started.")

if uploaded_file is not None:
    col1,col2=st.columns(2)

    with col1:
        with st.container(border=True, height=CHART_HEIGHT + 50):
            st.subheader("📈 Sentiment Trend Analysis")

            with st.spinner("Analyzing sentiment..."):
                categories_list = get_categories_array(categories)
                chunked_content = chunk_text(text=content, chunk_size=1000,
                chunk_overlap=100)

    if not chunked_content:
        st.warning("Document appears to be empty or could not be read.")
    else:
        sentiment_scores_list = get_cosine_scores(chunked_content, categories_list)

    if not sentiment_scores_list:
        st.warning("Could not generate sentiment scores.")
    else:
```

```
df=pd.DataFrame(sentiment_scores_list)
df['doc_num'] = range(1, len(df) + 1)
df['text_wrapped'] = df['text'].apply(lambda t: wrap_text(t, width=60))

df_melted = df.melt(
    id_vars=['doc_num', 'text_wrapped'],
    value_vars=categories_list,
    var_name='sentiment',
    value_name='score'
)

fig=px.line(
    df_melted,
    x='doc_num',
    y='score',
    color='sentiment',
    markers=True,
    custom_data=['text_wrapped'],
    height=CHART_HEIGHT - 50
)

fig.update_traces(
    hovertemplate=(
        "<b>Document #%%{x}</b><br><br>" +
        "<b>Text</b>: %%{customdata[0]}<br>" +
        "<b>Sentiment</b>: %%{fullData.name}<br>" +
        "<b>Score</b>: %%{y:.2f}" +
        "<extra></extra>"
    )
)
```

```
)  
fig.update_layout(  
    title="Sentiment Score Trends",  
    xaxis_title="Document Section",  
    yaxis_title="Sentiment Score",  
    legend_title="Sentiment"  
)  
st.plotly_chart(fig, use_container_width=True)
```

with col2:

```
withst.container(border=True, height=CHART_HEIGHT + 50):  
    st.subheader("🧠 Mind Map")  
  
    withst.spinner("Generating Mind Map..."):  
        mermaid = str(generate_mermaid(content=content))  
        mermaid = remove_mermaid_fences(mermaid)  
        st.markdown(f"""  
            <div style="overflow: auto;">  
                <script type="text/javascript">  
                    {st_mermaid(mermaid)}  
                </script>  
            </div>  
        """", unsafe_allow_html=True)
```

with col1:

```
withst.container(border=True, height=CHART_HEIGHT + 50):  
    st.subheader("📊 Frequency Analysis")  
    st.caption(f"Showing the **Top {top_n_terms}** terms found in the document.")
```

```

withst.spinner(f"Calculating top {top_n_terms} terms..."):

    bar_data = get_top_n_frequencies(content, top_n_terms)

if bar_data.empty:

    st.warning("Could not generate term frequencies. Document content may be too short or too sparse.")

else:

    bar_fig = px.bar(
        bar_data,
        x='Term',
        y='Frequency',
        title=f"Top {top_n_terms} Term Frequency",
        color='Term',
        height=CHART_HEIGHT - 50
    )

    bar_fig.update_layout(xaxis={'categoryorder':'total descending'})

    st.plotly_chart(bar_fig, use_container_width=True)

```

with col2:

```

withst.container(border=True, height=CHART_HEIGHT + 50):

    st.subheader("⭐ K-Means Document Clustering")
    st.caption(f"Clustering based on simulated embeddings into **{num_clusters}** groups.")

```

withst.spinner(f"Running K-Means with k={num_clusters}..."):

```

scatter_data = perform_kmeans_clustering(content, num_clusters)

if scatter_data.empty:

    st.warning("Cannot perform clustering on empty content.")

else:

```

```
scatter_fig = px.scatter(
    scatter_data, x='Dimension 1', y='Dimension 2',
    size='Size', color='Cluster', title=f"K-Means
    Clustering (k={num_clusters})",
    hover_name='Cluster', custom_data=['Text
    Snippet'], height=CHART_HEIGHT - 50

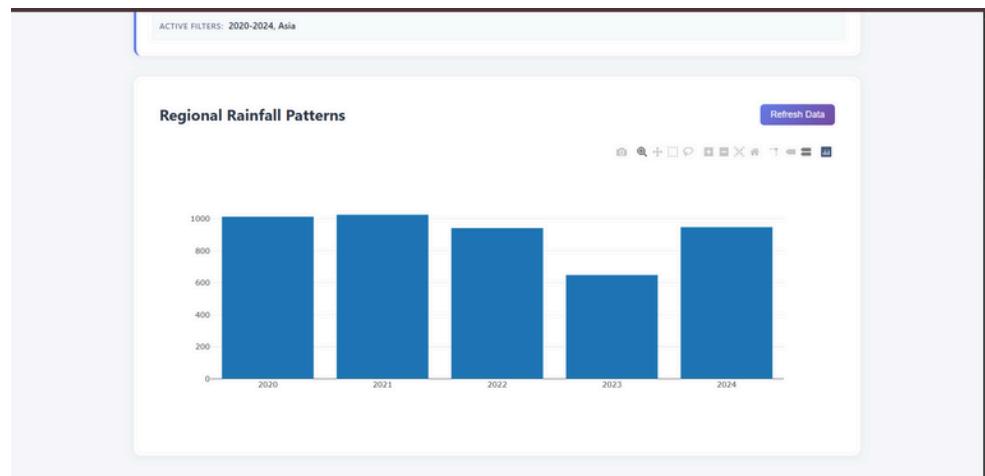
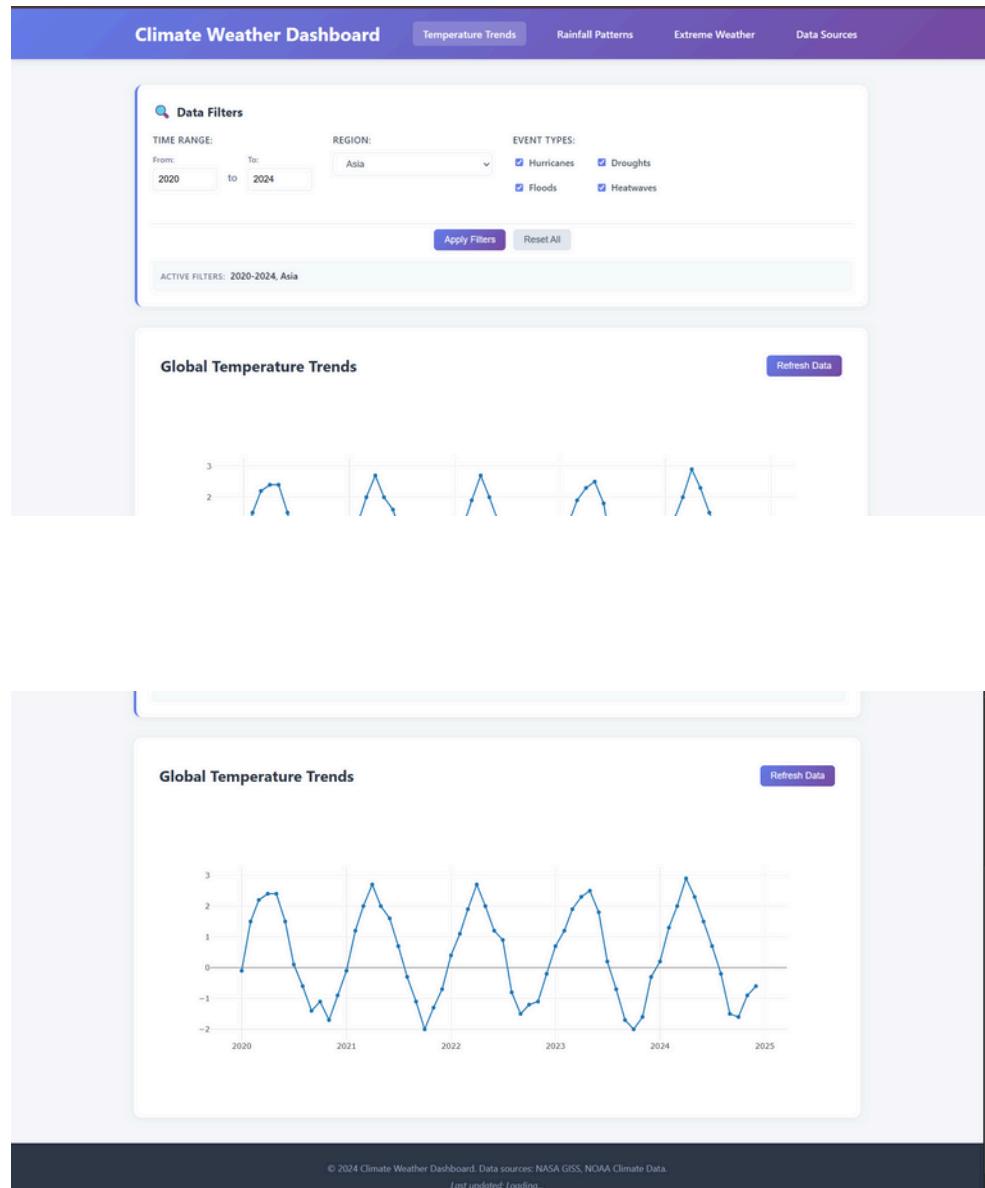
)

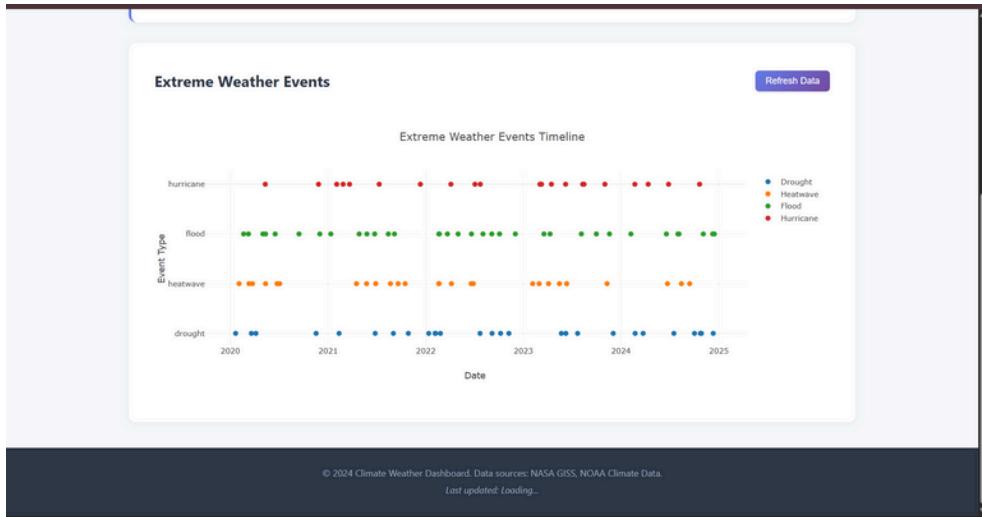
scatter_fig.update_traces(
    hovertemplate=(
        "<b>%{hovertext}</b><br><br>" +
        "<b>Snippet</b>: %{customdata[0]}<br>" +
        "Dimension 1: %{x:.2f}<br>" +
        "Dimension 2: %{y:.2f}" +
        "<extra></extra>"
    )
)

st.plotly_chart(scatter_fig, use_container_width=True)

else:
    st.info("Upload a document using the sidebar to begin analysis.") .
```

OUTPUT:





RESULT:

The developed web application successfully displays and analyzes the impact of climate change on various weather events. Users can view interactive visualizations such as line graphs, bar charts, and maps showing trends in temperature rise, rainfall variation, CO₂ levels, and frequency of natural disasters over the years.