Day69_SpaCy_Text_Summarization_Project

August 21, 2025

1 Extractive Text Summarization with spaCy Project :

- **Goal:** Build an extractive summarizer (selects the most important sentences from the original text).
- **Why:** Compress long documents into concise overviews for faster reading and downstream analysis.
- **Method:** Rank sentences by content importance using token statistics (term frequency), then pick the top-K.

2 Environment & Model

[1]:
```
#!pip install spacy

# 3 English pipelines exist by size/quality/speed: sm (small), md (medium), lg␣
↪(large)
#!python -m spacy download en_core_web_sm
```

3 Imports & Pipeline Configuration

- spaCy provides a processing pipeline that turns raw text → Doc with tokens, sentences, and linguistic annotations.
- We add a sentencizer (rule-based sentence boundary detector) so sentence segmentation works even if we disable heavier components (faster).
- We'll use stopword and punctuation filtering to remove low-information tokens from scoring.

[2]:
```
import spacy
from spacy.lang.en.stop_words import STOP_WORDS
from string import punctuation
from heapq import nlargest
```

[3]:
```
# Load lightweight English model
nlp = spacy.load("en_core_web_sm")

# Ensure we have a cheap, reliable sentence splitter
# (If the parser is active, it's already good; sentencizer is fast &␣
↪deterministic.)
```

```
if "sentencizer" not in nlp.pipe_names:
    nlp.add_pipe("sentencizer")
```

4 Input Text

- Any raw string can be processed. Later you'll replace this with file ingestion (TXT/PDF/DOCX).
- We keep original casing/punctuation for the final summary readability, but we lowercase for scoring to avoid case bias.

[4]:
```
text = """
There are broadly two types of extractive summarization tasks depending on what␣
↪the summarization program focuses on.
The first is generic summarization, which focuses on obtaining a generic␣
↪summary or abstract of the collection (whether documents, or sets of images,␣
↪or videos, news stories etc.).
The second is query relevant summarization, sometimes called query-based␣
↪summarization, which summarizes objects specific to a query.
Summarization systems are able to create both query relevant text summaries and␣
↪generic machine-generated summaries depending on what the user needs.
An example of a summarization problem is document summarization, which attempts␣
↪to automatically produce an abstract from a given document.
Sometimes one might be interested in generating a summary from a single source␣
↪document, while others can use multiple source documents (for example, a␣
↪cluster of articles on the same topic).
This problem is called multi-document summarization.
A related application is summarizing news articles.
Imagine a system which automatically pulls together news articles on a given␣
↪topic (from the web), and concisely represents the latest news as a summary.
"""
```

5 Tokenization & Linguistic Annotations

- nlp(text) → Doc: a container of Token objects with rich attributes (e.g., text, lemma_, is_stop).
- We'll score words using term frequency (TF). Optionally, we can score by lemma (group run/running/ran).

[5]:
```
doc = nlp(text)

# Peek at tokens (debug)
tokens_preview = [t.text for t in doc[:20]]
tokens_preview
```

[5]:
```
['\n',
 'There',
 'are',
 'broadly',
 'two',
 'types',
 'of',
 'extractive',
 'summarization',
 'tasks',
 'depending',
 'on',
 'what',
 'the',
 'summarization',
 'program',
 'focuses',
 'on',
 '.',
 '\n']
```

6 Vocabulary Pruning & Weighting

- Build a vocabulary of informative terms.
- Remove stopwords (common function words) and punctuation (non-lexical).
- Optionally include digits/symbols depending on domain (e.g., finance).

Why:

- Reduces noise. Keeps only content-bearing terms that better correlate with sentence salience.

Design choices:

- Use lemma to merge inflectional variants (recommended).
- Normalize TF by L∞ norm (divide by max frequency) to keep scores in 0,1

[6]:
```
stopwords = STOP_WORDS
punct_set = set(punctuation)

use_lemma = True   # switch to False to use surface forms

word_freq = {}
for token in doc:
    if token.is_space or token.is_punct:
        continue
    if token.is_stop:
        continue
```

```
    if token.text in punct_set:
        continue

    key = token.lemma_.lower() if use_lemma else token.text.lower()
    if not key or key in stopwords:
        continue
    word_freq[key] = word_freq.get(key, 0) + 1

# Normalize by max frequency (L∞ normalization)
if word_freq:
    max_f = max(word_freq.values())
    for w in word_freq:
        word_freq[w] = word_freq[w] / max_f

word_freq
```

[6]:

```
{'broadly': 0.1111111111111111,
 'type': 0.1111111111111111,
 'extractive': 0.1111111111111111,
 'summarization': 1.0,
 'task': 0.1111111111111111,
 'depend': 0.2222222222222222,
 'program': 0.1111111111111111,
 'focus': 0.2222222222222222,
 'generic': 0.3333333333333333,
 'obtain': 0.1111111111111111,
 'summary': 0.5555555555555556,
 'abstract': 0.2222222222222222,
 'collection': 0.1111111111111111,
 'document': 0.6666666666666666,
 'set': 0.1111111111111111,
 'image': 0.1111111111111111,
 'video': 0.1111111111111111,
 'news': 0.4444444444444444,
 'story': 0.1111111111111111,
 'etc': 0.1111111111111111,
 'second': 0.1111111111111111,
 'query': 0.4444444444444444,
 'relevant': 0.2222222222222222,
 'base': 0.1111111111111111,
 'summarize': 0.2222222222222222,
 'object': 0.1111111111111111,
 'specific': 0.1111111111111111,
 'system': 0.2222222222222222,
 'able': 0.1111111111111111,
 'create': 0.1111111111111111,
 'text': 0.1111111111111111,
```

```
 'machine': 0.1111111111111111,
 'generate': 0.2222222222222222,
 'user': 0.1111111111111111,
 'need': 0.1111111111111111,
 'example': 0.2222222222222222,
 'problem': 0.2222222222222222,
 'attempt': 0.1111111111111111,
 'automatically': 0.2222222222222222,
 'produce': 0.1111111111111111,
 'interested': 0.1111111111111111,
 'single': 0.1111111111111111,
 'source': 0.2222222222222222,
 'use': 0.1111111111111111,
 'multiple': 0.1111111111111111,
 'cluster': 0.1111111111111111,
 'article': 0.3333333333333333,
 'topic': 0.2222222222222222,
 'multi': 0.1111111111111111,
 'related': 0.1111111111111111,
 'application': 0.1111111111111111,
 'imagine': 0.1111111111111111,
 'pull': 0.1111111111111111,
 'web': 0.1111111111111111,
 'concisely': 0.1111111111111111,
 'represent': 0.1111111111111111,
 'late': 0.1111111111111111}
```

Note: L∞ normalization is simple and stable. Alternatives: L1 (sum to 1), TF-IDF (requires document collection; better when many docs).

7 Sentence Segmentation

- Build a list of candidate sentences to rank.
- Sentences come from Doc.sents (via sentencizer or parser).
- We'll preserve original order later for readability.

[7]:
```python
sentences = list(doc.sents)
len(sentences), sentences[:3]
```

[7]:
```
(9,
 [
  There are broadly two types of extractive summarization tasks depending on
what the summarization program focuses on.,
  The first is generic summarization, which focuses on obtaining a generic
summary or abstract of the collection (whether documents, or sets of images, or
videos, news stories etc.).,
```

```
 The second is query relevant summarization, sometimes called query-based
summarization, which summarizes objects specific to a query.])
```

8 Sentence Scoring

- Score each sentence by summing the normalized token weights of its content words.
- To reduce length bias (long sentences get larger sums), we can length-normalize by sentence token count.

Why:

- Simple additive content model approximates importance: sentences containing many high-value terms score higher.

[8]:
```python
length_normalize = True

sent_scores = {}
for sent in sentences:
    score = 0.0
    length = 0
    for token in sent:
        if token.is_space or token.is_punct:
            continue
        key = (token.lemma_.lower() if use_lemma else token.text.lower())
        if key in word_freq:
            score += word_freq[key]
        length += 1

    if length == 0:
        continue

    if length_normalize:
        score = score / length  # mean weight per token

    sent_scores[sent] = score

sent_scores
```

[8]:
```
{
 There are broadly two types of extractive summarization tasks depending on what
the summarization program focuses on.: 0.17647058823529413,
 The first is generic summarization, which focuses on obtaining a generic
summary or abstract of the collection (whether documents, or sets of images, or
videos, news stories etc.).: 0.16269841269841265,
 The second is query relevant summarization, sometimes called query-based
summarization, which summarizes objects specific to a query.:
0.23456790123456792,
```

```
 Summarization systems are able to create both query relevant text summaries and
generic machine-generated summaries depending on what the user needs.:
0.202020202020202,
 An example of a summarization problem is document summarization, which attempts
to automatically produce an abstract from a given document.:
0.22222222222222224,
 Sometimes one might be interested in generating a summary from a single source
document, while others can use multiple source documents (for example, a cluster
of articles on the same topic).: 0.12544802867383514,
 This problem is called multi-document summarization.: 0.2857142857142857,
 A related application is summarizing news articles.: 0.1746031746031746,
 Imagine a system which automatically pulls together news articles on a given
topic (from the web), and concisely represents the latest news as a summary.:
0.12444444444444443}
```

Alternatives:

- Positional prior: Boost early sentences (useful for news).
- Title overlap: Boost terms appearing in the document title.
- Redundancy control / Diversity: Penalize sentences that repeat selected content (see MMR below).

9 Selection (Top-K or Ratio)

- Choose K sentences (or ratio of total) with highest scores using heapq.nlargest.
- Then restore original document order for readability.

Why:

- Ranking → selection is the heart of extractive summarization.

[9]:
```
# Hyperparameters
ratio = 0.3                 # keep 30% sentences (set 0.2-0.35 for readable
 ↪summaries)
min_sentences = 3
max_sentences = 8

K = max(min_sentences, int(len(sentences) * ratio))
K = min(K, max_sentences, len(sentences))

top_sents = nlargest(K, sent_scores, key=sent_scores.get)

# Restore original order
top_sents_sorted = sorted(top_sents, key=lambda s: s.start)

summary_text = " ".join([s.text.strip() for s in top_sents_sorted])
summary_text
```

[9]: 'The second is query relevant summarization, sometimes called query-based summarization, which summarizes objects specific to a query. An example of a summarization problem is document summarization, which attempts to automatically produce an abstract from a given document. This problem is called multi-document summarization.'

10 (Optional) Redundancy Reduction with MMR (Maximal Marginal Relevance)

- Pure top-K may select near-duplicate sentences.
- MMR trades off relevance (sentence score) with novelty (dissimilarity to already chosen sentences).
- We approximate similarity with token overlap (Jaccard) for simplicity.

[10]:
```
def jaccard(a_tokens, b_tokens):
    a, b = set(a_tokens), set(b_tokens)
    if not a or not b:
        return 0.0
    return len(a & b) / len(a | b)

def select_with_mmr(sentences, scores, K, lambda_=0.7):
    # lambda_: 1.0 favors relevance only; 0.0 favors novelty only
    selected = []
    remaining = set(sentences)
    while remaining and len(selected) < K:
        if not selected:
            # pick the most relevant first
            best = max(remaining, key=lambda s: scores.get(s, 0))
            selected.append(best)
            remaining.remove(best)
            continue

        def mmr_score(s):
            sim_to_sel = max(
                jaccard([t.lemma_.lower() for t in s if t.is_alpha],
                        [t.lemma_.lower() for t in x if t.is_alpha])
                for x in selected
            ) if selected else 0.0
            return lambda_ * scores.get(s, 0) - (1 - lambda_) * sim_to_sel

        best = max(remaining, key=mmr_score)
        selected.append(best)
        remaining.remove(best)
    return sorted(selected, key=lambda s: s.start)
```

```
mmr_sents = select_with_mmr(sentences, sent_scores, K, lambda_=0.75)
mmr_summary = " ".join([s.text.strip() for s in mmr_sents])
mmr_summary
```

[10]:
```
'The second is query relevant summarization, sometimes called query-based
summarization, which summarizes objects specific to a query. An example of a
summarization problem is document summarization, which attempts to automatically
produce an abstract from a given document. This problem is called multi-document
summarization.'
```

11 Packaging as a Reusable Function

- Encapsulate the pipeline for reuse (backend API / UI).
- Parameters expose trade-offs: ratio, lemma use, normalization, MMR, limits.

[11]:
```
def summarize_text_spacy(
    text: str,
    nlp,
    ratio: float = 0.3,
    min_sentences: int = 3,
    max_sentences: int = 8,
    use_lemma: bool = True,
    length_normalize: bool = True,
    use_mmr: bool = False,
    mmr_lambda: float = 0.75
) -> str:
    if not text or not text.strip():
        return ""

    doc = nlp(text)
    sentences = list(doc.sents)
    if not sentences:
        return text.strip()

    stopwords = STOP_WORDS
    punct_set = set(punctuation)

    word_freq = {}
    for token in doc:
        if token.is_space or token.is_punct:
            continue
        if token.is_stop:
            continue
        if token.text in punct_set:
            continue
        key = token.lemma_.lower() if use_lemma else token.text.lower()
```

```
        if not key or key in stopwords:
            continue
        word_freq[key] = word_freq.get(key, 0) + 1

    if not word_freq:
        # fallback: return lead sentences
        keep = max(min_sentences, int(len(sentences) * ratio))
        keep = min(keep, max_sentences, len(sentences))
        return " ".join([s.text.strip() for s in sentences[:keep]])

    max_f = max(word_freq.values())
    for w in word_freq:
        word_freq[w] = word_freq[w] / max_f  # L∞ normalization

    sent_scores = {}
    for sent in sentences:
        score = 0.0
        length = 0
        for token in sent:
            if token.is_space or token.is_punct:
                continue
            key = token.lemma_.lower() if use_lemma else token.text.lower()
            if key in word_freq:
                score += word_freq[key]
            length += 1
        if length == 0:
            continue
        if length_normalize:
            score /= length
        sent_scores[sent] = score

    K = max(min_sentences, int(len(sentences) * ratio))
    K = min(K, max_sentences, len(sentences))

    if use_mmr:
        chosen = select_with_mmr(sentences, sent_scores, K, lambda_=mmr_lambda)
    else:
        top = nlargest(K, sent_scores, key=sent_scores.get)
        chosen = sorted(top, key=lambda s: s.start)

    return " ".join([s.text.strip() for s in chosen])

# Demo
summary_text = summarize_text_spacy(text, nlp, ratio=0.3, use_mmr=True)
summary_text
```

[11]: 'The second is query relevant summarization, sometimes called query-based summarization, which summarizes objects specific to a query. An example of a summarization problem is document summarization, which attempts to automatically produce an abstract from a given document. This problem is called multi-document summarization.'

12 Complexity & Behavior

- Time:
  - Token pass (vocab build): O(N) tokens
  - Sentence scoring: O(N) tokens
  - Selection: O(S log K) with heap (S=sentences)
- Space: vocabulary O(V), sentence scores O(S)
- Determinism: deterministic given same text & parameters (no randomness).
- Biases: favors content-dense sentences; length normalization mitigates long-sentence bias; MMR mitigates redundancy.

13 Limitations & Upgrades

- Extractive only: doesn't paraphrase or compress internally (no "abstractive" fluency improvements).
- Vocabulary mismatch: rare words can be overweighted; consider TF-IDF over a corpus.
- Domain sensitivity: customize stopwords, keep numbers/symbols if domain requires (finance, science).
- Improvements:
  - TextRank (graph centrality)
  - Supervised extractive models
  - Abstractive LLMs for fluent summaries (cost/latency/safety trade-offs)
  - ROUGE for evaluation against reference summaries

14 Frontend Code (Streamlit App)

Save this as app.py:

```
import streamlit as st
import spacy
from spacy.lang.en.stop_words import STOP_WORDS
from string import punctuation
from heapq import nlargest
```

```python
# ==============================
# Summarizer Function
# ==============================
def summarize_text_spacy(
    text: str,
    nlp,
    ratio: float = 0.3,
    min_sentences: int = 3,
    max_sentences: int = 8
) -> str:
    if not text or not text.strip():
        return ""

    doc = nlp(text)
    sentences = list(doc.sents)
    if not sentences:
        return text.strip()

    stopwords = STOP_WORDS
    punct_set = set(punctuation)

    word_freq = {}
    for token in doc:
        if token.is_space or token.is_punct:
            continue
        if token.is_stop:
            continue
        if token.text in punct_set:
            continue
        key = token.lemma_.lower()
        if not key or key in stopwords:
            continue
        word_freq[key] = word_freq.get(key, 0) + 1

    if not word_freq:
        return " ".join([s.text.strip() for s in sentences[:min_sentences]])

    max_f = max(word_freq.values())
    for w in word_freq:
        word_freq[w] = word_freq[w] / max_f

    sent_scores = {}
    for sent in sentences:
        score = 0.0
        length = 0
        for token in sent:
            if token.is_space or token.is_punct:
                continue
```

```
            key = token.lemma_.lower()
            if key in word_freq:
                score += word_freq[key]
            length += 1
        if length > 0:
            score /= length
            sent_scores[sent] = score

    K = max(min_sentences, int(len(sentences) * ratio))
    K = min(K, max_sentences, len(sentences))

    top = nlargest(K, sent_scores, key=sent_scores.get)
    chosen = sorted(top, key=lambda s: s.start)

    return " ".join([s.text.strip() for s in chosen])

# ==============================
# Streamlit Frontend
# ==============================
st.set_page_config(page_title="Text Summarizer", page_icon=" ", layout="wide")
st.title("  Extractive Text Summarizer")
st.write("Upload a file or paste text below to generate a summary.")

# Load spaCy model once
@st.cache_resource
def load_model():
    return spacy.load("en_core_web_sm")

nlp = load_model()

# Sidebar controls
st.sidebar.header("  Settings")
ratio = st.sidebar.slider("Summary Length (ratio of original)", 0.1, 0.9, 0.3, 0.05)
min_sents = st.sidebar.number_input("Minimum Sentences", 1, 10, 3)
max_sents = st.sidebar.number_input("Maximum Sentences", 1, 20, 8)

# Input section
input_method = st.radio("Choose Input Method:", ["Paste Text", "Upload File"])

input_text = ""
if input_method == "Paste Text":
    input_text = st.text_area("Enter your text here:", height=200)
elif input_method == "Upload File":
    uploaded_file = st.file_uploader("Upload a .txt, .pdf, or .docx file", type=["txt", "pdf",
    if uploaded_file:
        ext = uploaded_file.name.split(".")[-1].lower()
        if ext == "txt":
            input_text = uploaded_file.read().decode("utf-8")
```

```
        elif ext == "pdf":
            import pdfplumber
            with pdfplumber.open(uploaded_file) as pdf:
                input_text = "\n".join(page.extract_text() or "" for page in pdf.pages)
        elif ext == "docx":
            from docx import Document
            doc = Document(uploaded_file)
            input_text = "\n".join(p.text for p in doc.paragraphs)

# Run summarization
if st.button("Generate Summary"):
    if input_text.strip():
        summary = summarize_text_spacy(input_text, nlp, ratio=ratio, min_sentences=min_sents, 
        st.subheader("  Original Text")
        st.write(input_text)
        st.subheader("  Generated Summary")
        st.success(summary)
    else:
        st.warning("Please provide text or upload a file first.")
```

15 How to Run

1. Save as app.py.
2. Install dependencies:

```
pip install streamlit spacy pdfplumber python-docx
python -m spacy download en_core_web_sm
```

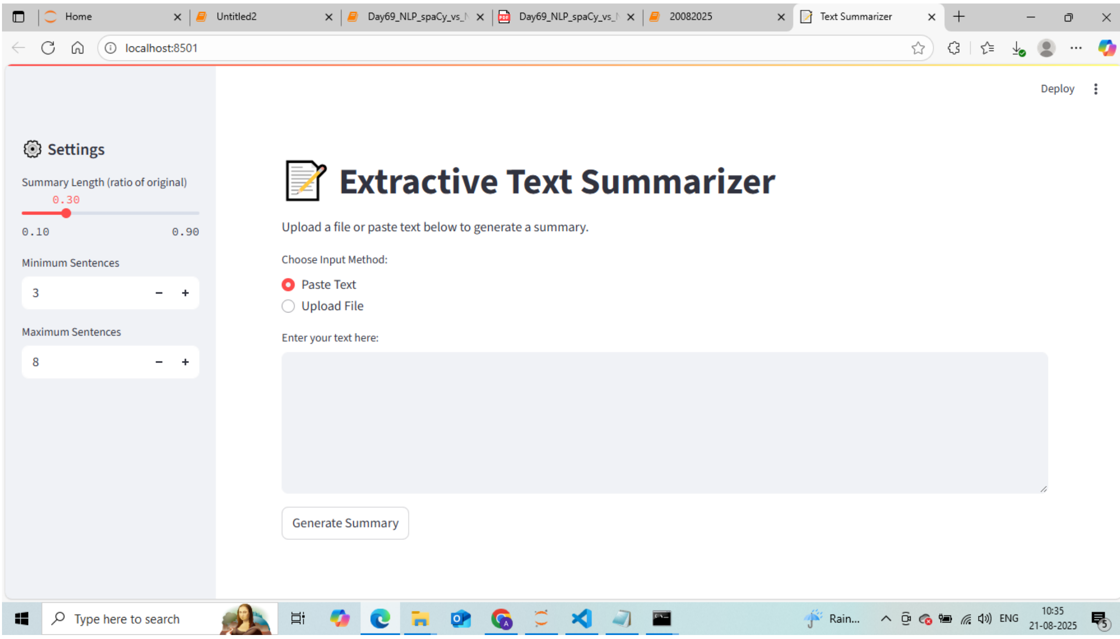
3. Run:

```
streamlit run app.py
```

4. Browser will open → paste text or upload file → see summary.

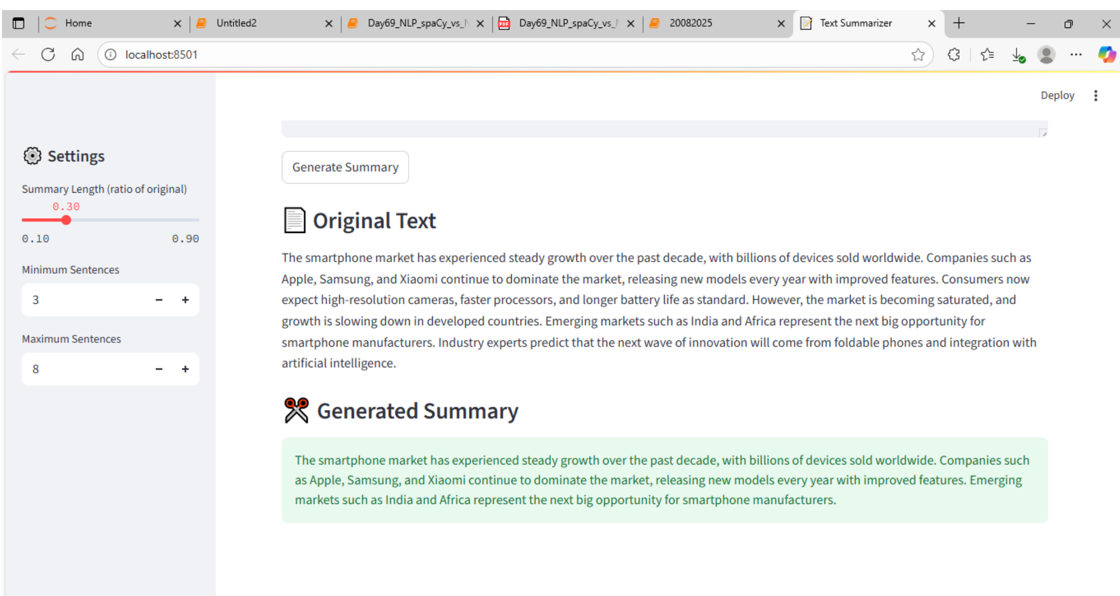
15.1 Overall App Look

This shows the complete UI of our summarizer app with background image and controls.



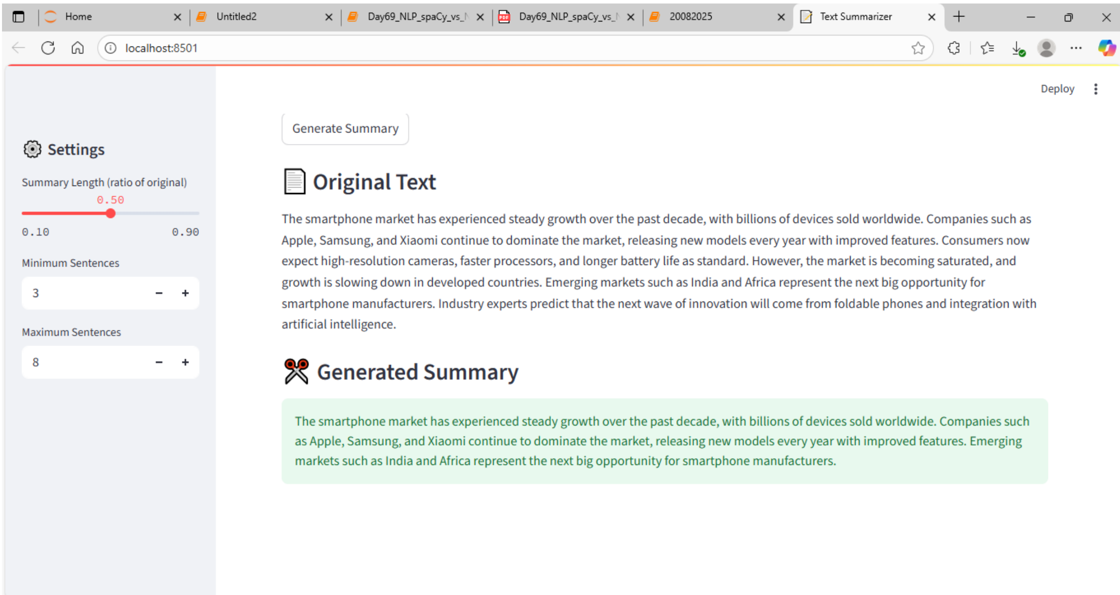
15.2 Upload Text File & Generate Summary

Here we uploaded a .txt file, extracted its content, and generated a summary.



15.3 Summary Length = 50%

When we set the summary ratio to 50%, the app extracted half of the original text into a concise summary.



15.4 Minimum Sentences = 2 Lines

When the minimum sentences parameter was set to 2, the app ensured at least two lines were returned as summary.