

DocuPresenter ReadMe

Overview

DocuPresenter is a tool that allows you to create summarized and structured presentations using text extracted from Word or PDF documents. This README provides a step-by-step guide on how to set up and use DocuPresenter. **Highlighted** Sections are Variable/**Customizable**.

Setup

1. First, install the required Python packages by running the following commands:

```
# Updating the apt-get package list

!apt-get update


# Installing wkhtmltopdf for converting HTML to PDF

!apt-get install -y wkhtmltopdf


# Installing Python packages

!pip install -U google-generativeai # Generative AI functionalities from Google

!pip install googletrans==4.0.0-rc1 # Translation tasks

!pip install --upgrade google-api-python-client # Interacting with Google APIs

!pip install nltk # Natural language processing tasks

!pip install PyPDF2 # Handling and manipulating PDF files

!pip install weasyprint CairoSVG # HTML to PDF conversion and vector graphics handling

!pip install python-docx # Working with DOCX files
```

2. Import the necessary libraries and set up your API Key:

```
# Google Generative AI
```

```
import google.generativeai as palm
```

```
# Data handling and processing
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Natural Language Processing
```

```
import nltk
```

```
from nltk.tokenize import sent_tokenize
```

```
# Translation
```

```
import googletrans
```

```
from googletrans import Translator
```

```
# Regular expressions
```

```
import re
```

```
# PDF handling
```

```
import PyPDF2
```

```
# Working with DOCX files
```

```
from docx import Document
```

```
# Provides a way of using operating system dependent functionality like reading or writing to a file system
```

```
import os
```

```
# Used for manipulating text, such as formatting long strings to a specific width or indenting
```

```
import textwrap
```

```
# Set your API Key
```

```
palm.configure(api_key='YOUR_API_KEY_HERE')
```

3. Choose a model for text embedding. The following code lists available models and selects one for text embedding ('embedText')

```
models = [m for m in palm.list_models() if 'embedText' in m.supported_generation_methods]
```

```
model = models[0] # Choose the desired model
```

Retrieve Data from Drive

1. Mount Google Drive to Google Colab to access your PDF/DOCX files.

```
# Mount Google Drive to Google Colab.
```

```
from google.colab import drive
```

```
drive.mount('/content/gdrive', force_remount=True)
```

2. Extract text from PDF or DOCX files in a specific folder and create a Pandas DataFrame to store it. Ensure that your input files are stored in the folder you specify (folder_path). The code reads and compiles its text content, removing newline characters for better formatting.

```
"shortened_text = cleaned_text.encode('utf-8')[:9900].decode('utf-8', errors='ignore')"
```

At this moment, a characters limit of 9900 bytes is set since there is a maximum limit of 10000 bytes when embedding text. This processing step helps avoiding any potential errors. However, under ideal conditions, this API will be able to process a larger amount of text.

```
folder_path = "/content/gdrive/MyDrive/Test1"
```

```
documents_dict = {}

# Check if the folder exists

if os.path.exists(folder_path):

    # Iterate over files in the folder

    for filename in os.listdir(folder_path):

        file_path = os.path.join(folder_path, filename)

        text = ""

        if filename.endswith(".pdf"): # Check if it's a PDF file

            with open(file_path, 'rb') as pdf_file:

                pdf_reader = PyPDF2.PdfReader(pdf_file)

                # Extract text from each page

                for page in pdf_reader.pages:

                    page_text = page.extract_text()

                    if page_text: # Ensure there's text on the page

                        text += page_text + " " # Add space after each page's content

        elif filename.endswith(".docx"): # Check if it's a DOCX file

            doc = Document(file_path)

            # Extract text from each paragraph
```

```

for para in doc.paragraphs:

    text += para.text + " " # Add space after each paragraph's content


# Truncate text if too long for your application

shortened_text = text[:9900] if len(text) > 9900 else text


# Add the shortened text to the dictionary

documents_dict[filename] = shortened_text

else:

    print(f"Folder {folder_path} does not exist!")


# Convert the dictionary to a dataframe

df = pd.DataFrame(list(documents_dict.items()), columns=['Filename', 'Text'])


df

```

3. Generate embeddings for the text and add them to the DataFrame.

```

# Get the embeddings of each text and add to an embeddings column in the dataframe

def embed_fn(text):

    return palm.generate_embeddings(model=model, text=text)['embedding']


df['Embeddings'] = df['Text'].apply(embed_fn)


df

```

Query the Documents

1. Specify the **topic and age** group of the audience you wish to teach for your query.

```
topic = "AI and Machine Learning"
```

```
age = "23"
```

2. Create a function to find the most relevant passage related to the topic in your documents.

```
def find_best_passage(topic, dataframe):  
    """  
    Compute the distances between the query and each document in the dataframe  
    using the dot product.  
    """  
    query_embedding = palm.generate_embeddings(model=model, text=topic)  
    dot_products = np.dot(np.stack(dataframe['Embeddings']), query_embedding['embedding'])  
    idx = np.argmax(dot_products)  
    return dataframe.iloc[idx]['Text'] # Return text from index with max value
```

3. Query the documents to find the best passage.

```
passage = find_best_passage(topic, df)
```

4. Create a prompt using the found passage and topic.

```
def make_prompt(topic, age, relevant_passage):  
    escaped = relevant_passage.replace("","").replace("'", "").replace("\n", " ")  
    prompt = textwrap.dedent("""\n  
    You are a helpful and informative bot that creates presentations using text from the  
reference passage included below.  
  
    I am a teacher for a group of '{age}'-year-old students, please output markdown scripts.  
  
    If the passage is irrelevant to the presentation, you may ignore it.  
  
    Topic: '{topic}'
```

```
PASSAGE: '{relevant_passage}'

ANSWER:

"".format(topic=topic, age=age, relevant_passage=escaped)

return prompt

prompt = make_prompt(topic, age, passage)

print(prompt)
```

Generate a Presentation

1. Choose a text generation model and set parameters like temperature.

```
text_models = [m for m in palm.list_models() if 'generateText' in
m.supported_generation_methods]
```

```
text_model = text_models[0]
```

```
temperature = 0.5
```

2. Generate a presentation using the prompt and the selected text generation model.

```
answer = palm.generate_text(prompt=prompt,

                             model=text_model,

                             temperature=temperature,

                             max_output_tokens=1000)
```

```
llm_output= answer.result
```

Translation

1. Provide a function for translating text from one language to another using a translation library.

If the translation fails for any reason, it prints an error message and returns the original text.

```
def translate_text(text, dest_language):  
  
    translator = Translator()  
  
    try:  
  
        translation = translator.translate(text, dest=dest_language)  
  
        return translation.text  
  
    except Exception as e:  
  
        print(f"Error during translation: {e}")  
  
        return text # Return the original text if translation fails
```

2. Set the (Google Translate supported) **language**.

```
lang = "English"
```

3. Translate the given text to the specified language using the translation function.

```
translated_llm_output = translate_text(llm_output, lang)  
  
translated_llm_output
```

Conversion to PDF

1. Convert the generated Markdown content to a PDF and save it to Google Drive.

```
from weasyprint import HTML  
  
import markdown  
  
# Ensure llm_output is a string and strip unnecessary characters if present.  
  
translated_llm_output = translated_llm_output.strip("``").strip()
```



```

# Split the content into slides based on '##' and insert page breaks before headers

slides = re.split(r'\n## ', translated_llm_output)

for i, slide in enumerate(slides):

    if i > 0:

        slides[i] = f"\n\n<div style=\"page-break-before: always;\"></div>\n{slide}"

# Replace all heading levels with bold headings

for i, slide in enumerate(slides):

    for heading_level in range(2, 7):

        heading_pattern = f"^{('#' * heading_level)} (.*)"

        replacement_pattern = f"\1 **\2**"

        slide = re.sub(heading_pattern, replacement_pattern, slide, flags=re.MULTILINE)

    slides[i] = slide

# Convert each slide to HTML

html_slides = [markdown.markdown("# {}".format(slide),
extensions=['markdown.extensions.extra']) for slide in slides]

# HTML and CSS for the presentation-like format

presentation_html = ""

```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <meta charset="utf-8">
```

```
  <title>Presentation</title>
```

```
  <style>
```

```
    @page {{
```

```
      size: A4 landscape;
```

```
      margin: 0mm;
```

```
    }}
```

```
    body {{
```

```
      font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
```

```
      margin: 0;
```

```
      padding: 0;
```

```
      display: block;
```

```
    }}
```

```
    section {{
```

```
      width: 80%;
```

```
      max-width: 1280px;
```

```
      margin: 1cm auto;
```

```
      page-break-after: always;
```

```
      page-break-inside: avoid;
```

```
      display: block;
```

```
    }}

    h1, h2, h3, h4 {{

        text-align: center;

        margin-top: 0.5cm;

        font-weight: bold;

    }}

    p, li {{

        font-size: 24px;

        line-height: 1.5;

        text-align: left;

        margin-left: 10%;

        margin-right: 10%;

    }}

    ul, ol {{

        padding-left: 20px;

    }}

</style>

</head>

<body>

    {}

</body>

</html>

"".format("\n".join(html_slides))
```

```
# Set the output file name
```

```
output_file_name = f"{topic}_{lang}_{age}.pdf"
```

```
# Set the path to save the PDF file (modify as needed)
```

```
pdf_file_path = f"/content/gdrive/MyDrive/Test/{output_file_name}"
```

```
# Generate the PDF from the HTML string and save it to the specified path
```

```
HTML(string=presentation_html).write_pdf(pdf_file_path)
```

```
print(f"The presentation PDF has been created and saved to {pdf_file_path}.")
```