Python Code Documentation in LaTeX

Debajyoti Maity

November 4, 2024

1 How to Create a Virtual Environment

1.1 Check Python Version

First, ensure that Python is installed on your system and check the version by running the following command:

```
python --version
```

This will display the version of Python installed. Ensure you're using the correct version of Python for your project.

1.2 Create a Virtual Environment

Use the `venv` module to create a new virtual environment. Replace `[your environment]` with the desired name of your virtual environment:

```
python3 -m venv [your environment]
```

This command will create a folder with your virtual environment files.

1.3 Activate the Virtual Environment (Windows PowerShell)

To activate your virtual environment in PowerShell, navigate to your environment folder and run the following command:

```
.\[your environment]\Scripts\Activate.Ps1
```

After activation, your prompt will change to show that the virtual environment is active.

2 How to Create a Google API Key

To use Google services, you need an API key. Follow the steps below to create one:

1. Go to the Google API Studio website by visiting: `https://aistudio.google.com/app/apikey`
2. Once you're on the site, click on the **Get API Key** button.
3. Follow the instructions on the screen to generate your API key.
4. After the key is generated, store it securely and avoid sharing it publicly.

Once you have your API key, you can store it in a `.env` file and use it in your project as described in the previous section.

3 Storing the API Key in a .env File

Instead of hardcoding sensitive information like your Google API key directly in your script, it is a good practice to store such information in a `.env` file.

1. Create a file named `.env` in the root directory of your project.
2. Inside the `.env` file, add your Google API key like this:

   ```
   GOOGLE_API_KEY="your-api-key"
   ```

3. Make sure to add the `.env` file to your `.gitignore` to prevent it from being included in version control:

   ```
   .env
   ```

After this, you can load the key into your Python project using a library like `python-dotenv`.

3.1 Loading API Key in Python

Install the `python-dotenv` package by running:

```
pip install python-dotenv
```

Then, in your Python script, load the key as follows:

```
from dotenv import load_dotenv
import os

load_dotenv()

api_key = os.getenv('GOOGLE_API_KEY')
```

Python Code with Explanations

```python
import PyPDF2
# This line imports the PyPDF2 library, which is a Python library used to read and write PDF files. It allows you to extract text from PDFs, merge PDFs, and more.

import os
# This line imports the os module, which provides a way of using operating system-dependent functionality like reading or writing to the file system, fetching environment variables, and handling file paths.

from dotenv import load_dotenv
# This line imports the load_dotenv function from the dotenv package. The dotenv package is used to load environment variables from a .env file into your Python script, which is useful for managing configuration settings or sensitive information like API keys.

import google.generativeai as genai
# This line imports the google.generativeai module as genai. This module is assumed to be a Python client library for accessing Google's Generative AI services, such as the Gemini Pro API.

# Load environment variables
load_dotenv()
```

```
# This line calls the load_dotenv() function, which reads the .env file in the current
    directory (or the directory specified) and loads the environment variables found there
    into the environment for this script. This makes it possible to use these variables in
    your Python code.

import base64
# This line imports the base64 module, which provides functions for encoding binary data to
    base64-encoded strings and decoding base64-encoded strings back into binary data. Base64
     encoding is commonly used when you need to encode binary data, especially when that
    data needs to be stored and transferred over media that are designed to deal with
    textual data.

from PIL import Image
# This line imports the Image class from the PIL package, which stands for Python Imaging
    Library (now known as Pillow). Image is a class that provides a set of methods to
    manipulate images, such as opening, modifying, and saving various image file formats.
print(os.getenv("GOOGLE_API_KEY"))  # This should print your actual API key if loaded
    correctly
# This line prints the value of the environment variable GOOGLE_API_KEY to the console. The
    os.getenv function is used to get the value of an environment variable. If the
    GOOGLE_API_KEY was successfully loaded from the .env file by load_dotenv(), it will
    print out the actual API key. If not, it will print None.

genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))
# This line calls the configure method from the genai module, passing it the GOOGLE_API_KEY
    environment variable as the api_key parameter. This configures the genai client to use
    the specified API key for authenticating requests to Google's Generative AI services.

genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))
# This line calls the configure method from the genai module, passing it the GOOGLE_API_KEY
    environment variable as the api_key parameter. This configures the genai client to use
    the specified API key for authenticating requests to Google's Generative AI services.


# Function to preprocess and extract text from PDF
def pdf_file_preprocessing(pdf_path, selected_pages):
    try:
        with open(pdf_path, 'rb') as pdfFileObject:
            pdfReader = PyPDF2.PdfReader(pdfFileObject)
            text = ""
            for i in selected_pages:
                page = pdfReader.pages[i - 1]
                page_text = page.extract_text()
                if page_text:
                    text += page_text + "\n"
            return text
    except Exception as e:
        print(f"Error processing PDF file: {e}")
        return ""
```

Note: The following explanations correspond to specific lines of code in the listing above.

- `pdfReader = PyPDF2.PdfReader(pdfFileObject)`: This line creates an instance of the PdfReader class from the PyPDF2 library, passing the pdfFileObject as an argument. The PdfReader object represents the PDF file and allows access to its pages and other content.

- `model = genai.GenerativeModel("gemini-1.5-pro")`: This line creates an instance of the GenerativeModel class from the genai module, specifying "gemini-1.5-pro" as the model version. This is presumably a reference to a specific version of a generative AI model provided by an API.

- `base64_pdf = base64.b64encode(f.read()).decode('utf-8')`: This line reads the entire content of the file f, encodes it into base64 (a format suitable for embedding binary data in HTML), and then decodes the base64-encoded bytes into a UTF-8 string. The result is stored in the variable base64_pdf.

- `.decode('utf-8')`: Since the result of base64.b64encode(...) is a bytes object, it needs to be decoded to a string before it can be used in a text-based context (like an HTML document). The decode method converts the bytes object (which contains base64-encoded text) into a Python string, using UTF-8 as the character encoding.

- The final line returns an HTML iframe element as a string. The src attribute of the iframe uses a data URI with the base64-encoded PDF content. This allows the PDF to be displayed directly in the Jupyter Notebook.

base_text_prompt = """ This is a multi-line string that defines a template for a text prompt that could be used in a question-answering system. The template includes instructions for an AI assistant, emphasizing the need for clear, concise, and relevant responses. It also includes placeholders () for the context and the question that will be provided later. The idea is that you would format this string with actual context and question text when you want to use it to generate an answer from an AI model or service. """

Note: The following explanations correspond to specific lines of code in the listing above.

- `genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))`: This line calls the configure method from the genai module, passing it the GOOGLE_API_KEY environment variable as the api_key parameter. This configures the genai client to use the specified API key for authenticating requests to Google's Generative AI services.
  - `configure`: To set up the API client with necessary configuration such as API keys, endpoints, and other settings.
  - `generate_content`: To generate content based on a prompt or other input parameters.
  - `create_model`: To instantiate a new model object that can be used for making predictions or generating content.
  - `list_models`: To retrieve a list of available models that you can use with the API.
  - `get_model`: To retrieve details about a specific model by its identifier.
  - `update_model`: To update settings or parameters for a specific model.
  - `delete_model`: To remove a model from your account or the API service.
- `import base64`: This line imports the base64 module, which provides functions for encoding binary data to base64-encoded strings and decoding base64-encoded strings back into binary data.
- `from PIL import Image`: This line imports the Image class from the PIL package, which stands for Python Imaging Library (now known as Pillow). Image is a class that provides a set of methods to manipulate images.
- `base_text_prompt`: This is a multi-line string that defines a template for a text prompt that could be used in a question-answering system.

Note: The following explanations correspond to specific lines of code in the listing above.

- `pdfReader = PyPDF2.PdfReader(pdfFileObject)`: This line creates an instance of the PdfReader class from the PyPDF2 library, passing the pdfFileObject as an argument. The PdfReader object represents the PDF file and allows access to its pages and other content.
- `model = genai.GenerativeModel("gemini-1.5-pro")`: This line creates an instance of the GenerativeModel class from the genai module, specifying "gemini-1.5-pro" as the model version. This is presumably a reference to a specific version of a generative AI model provided by an API.
- `base64_pdf = base64.b64encode(f.read()).decode('utf-8')`: This line reads the entire content of the file f, encodes it into base64 (a format suitable for embedding binary data in HTML), and then decodes the base64-encoded bytes into a UTF-8 string. The result is stored in the variable base64_pdf.
- `.decode('utf-8')`: Since the result of base64.b64encode(...) is a bytes object, it needs to be decoded to a string before it can be used in a text-based context (like an HTML document). The decode method converts the bytes object (which contains base64-encoded text) into a Python string, using UTF-8 as the character encoding.

- The final line returns an HTML iframe element as a string. The src attribute of the iframe uses a data URI with the base64-encoded PDF content. This allows the PDF to be displayed directly in the Jupyter Notebook.