Project Documentation

Scripts Overview

Entity Data Extraction

1. content_extract.ipynb

Purpose

This script aims to extract structured data from PDF documents using Google Cloud Document AI. It utilizes optical character recognition (OCR) to identify entities like names, ranks, and regiments from the provided document.

Dependencies

- **google-cloud-documentai:** For interfacing with Google Cloud Document AI.
- **google-cloud-storage:** For storing documents.
- **aspose-words:** For breaking pdfs into pages.

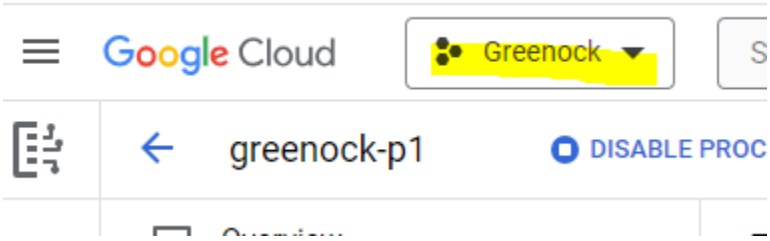
Functionality

1. **Data Extraction:** Extracts structured data such as last name, first name, rank, regiment, and page number from the provided PDF document.
2. **CSV Output:** Saves the extracted data into a CSV file.

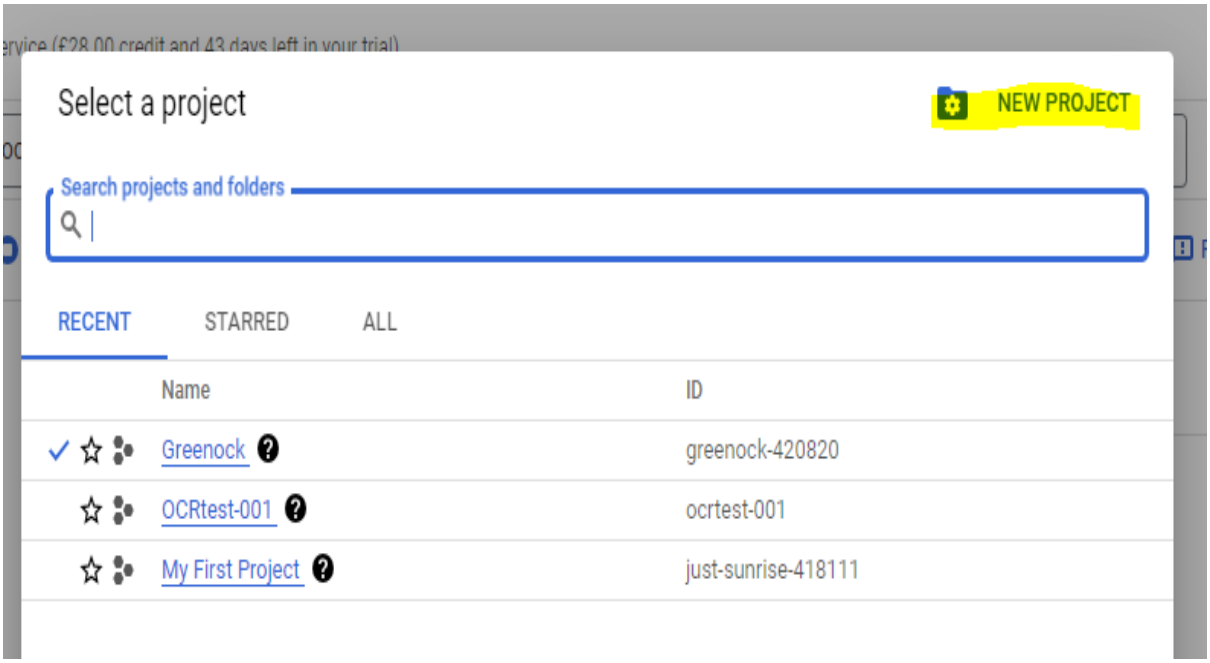
Google Cloud Document AI Integration:

Create Project

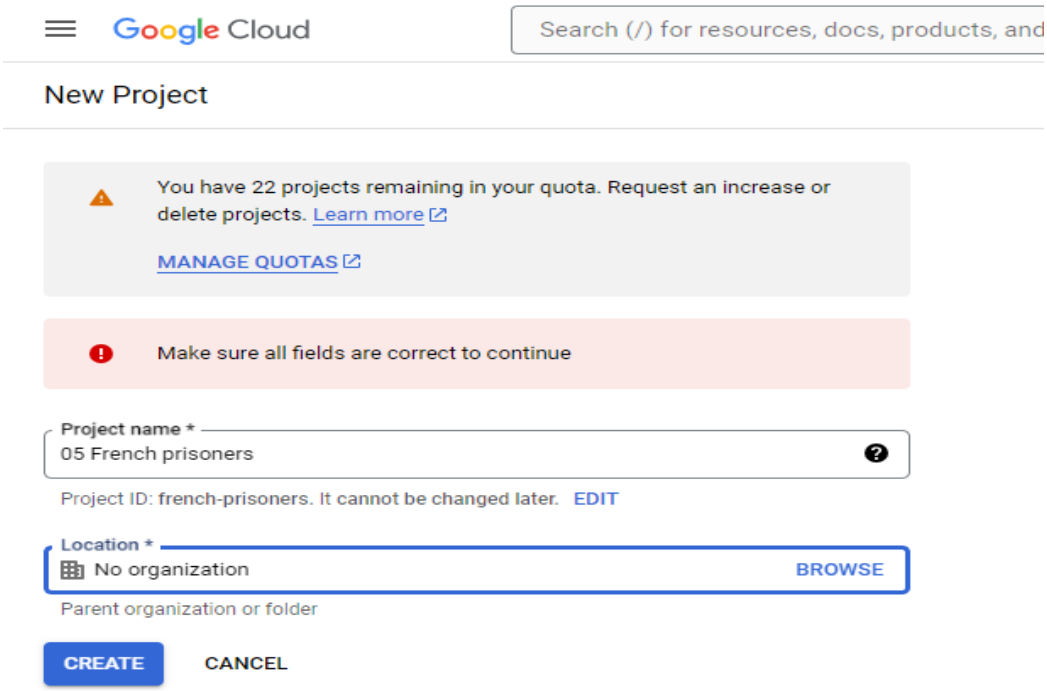
1. Go to Google Cloud Console
2. Click here



3. Click on New Project

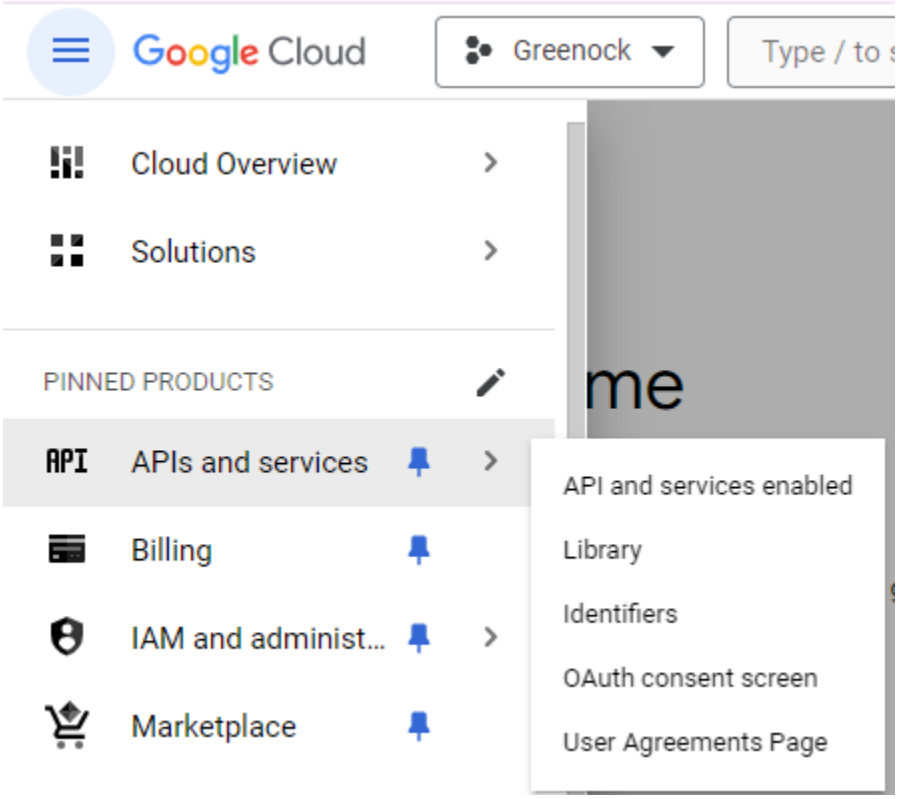


4. Give the project a name



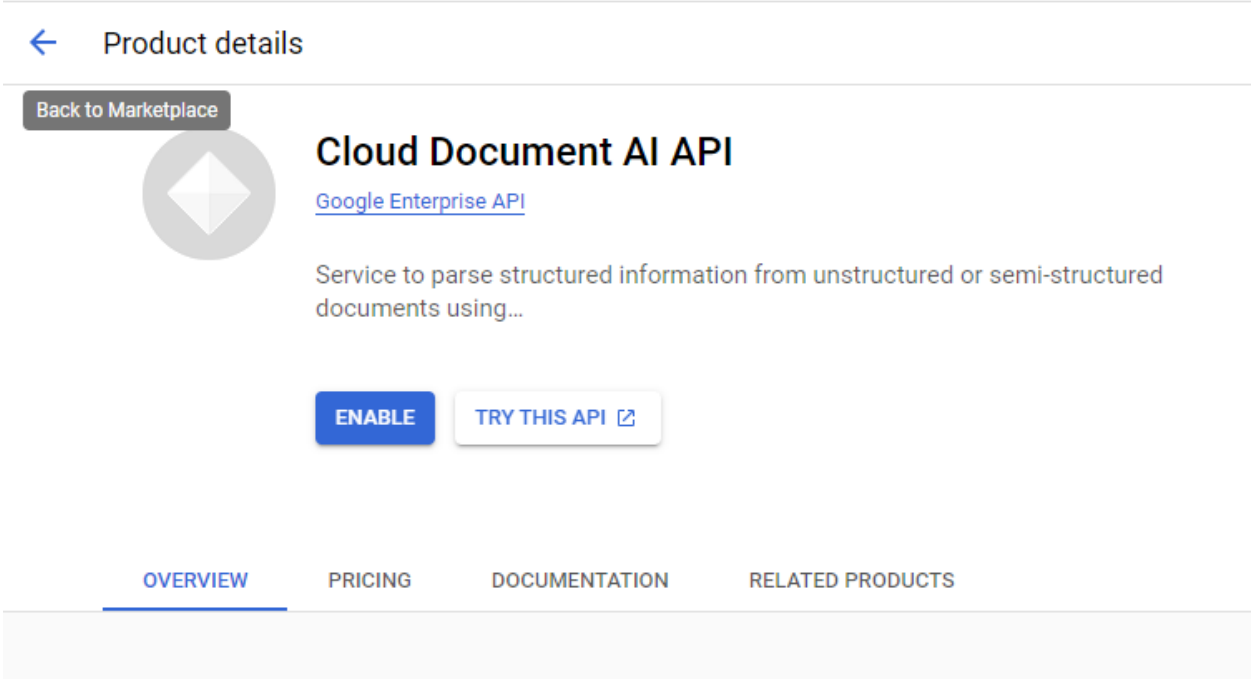
Cloud Document AI API

1. Go to the "API & Services" section in the Google Cloud Console.



2. Click on "Enable APIs and Services" and search for "Cloud Document AI API".

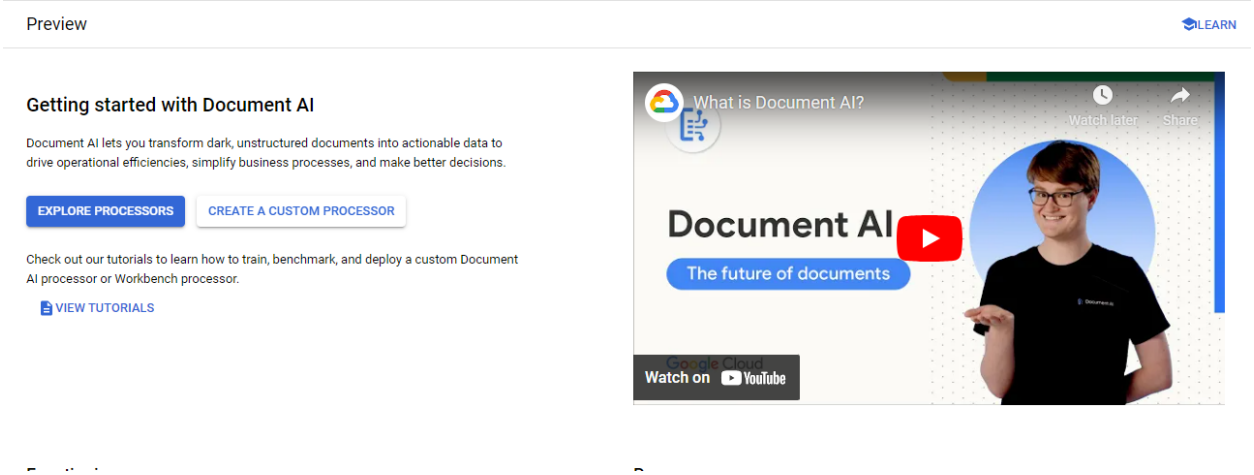
3. Select the Document AI API from the search results and click "Enable".



Document AI

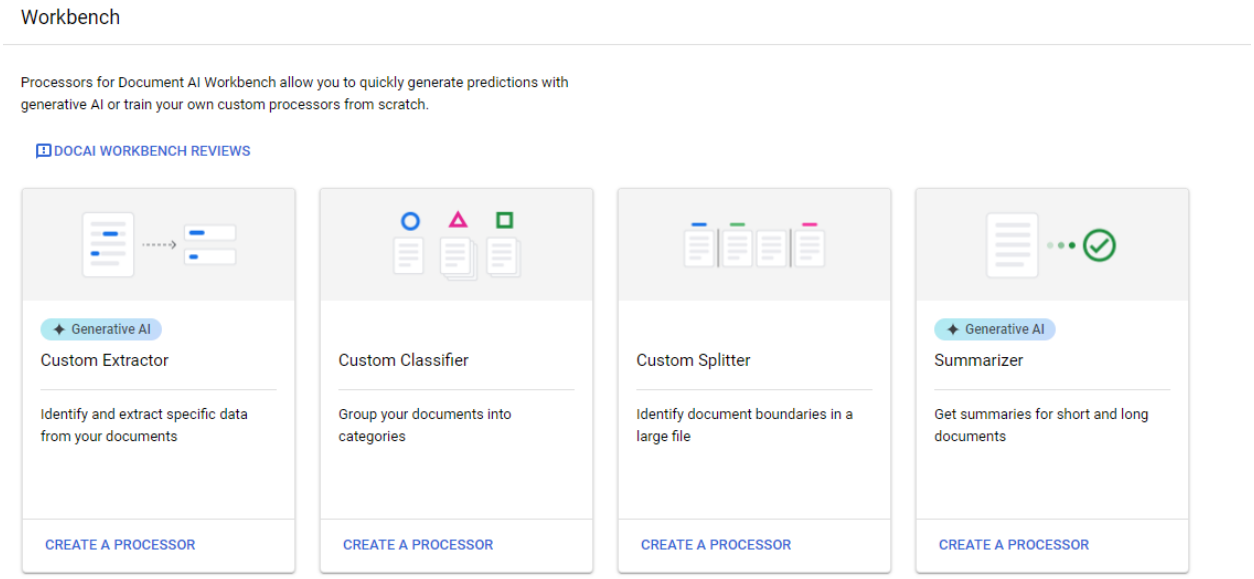
1. After enabling the API, search "Document AI" and then you'll be redirected to the Document AI dashboard.

2. This screen will appear



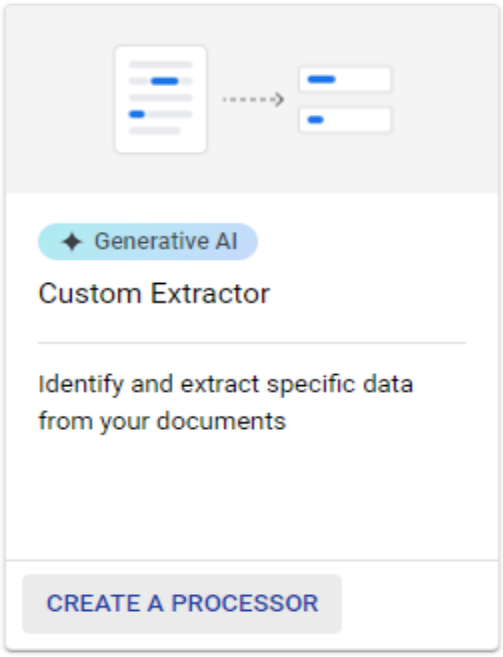
3. Click on “Create a Custom Processor”

4. This screen will appear

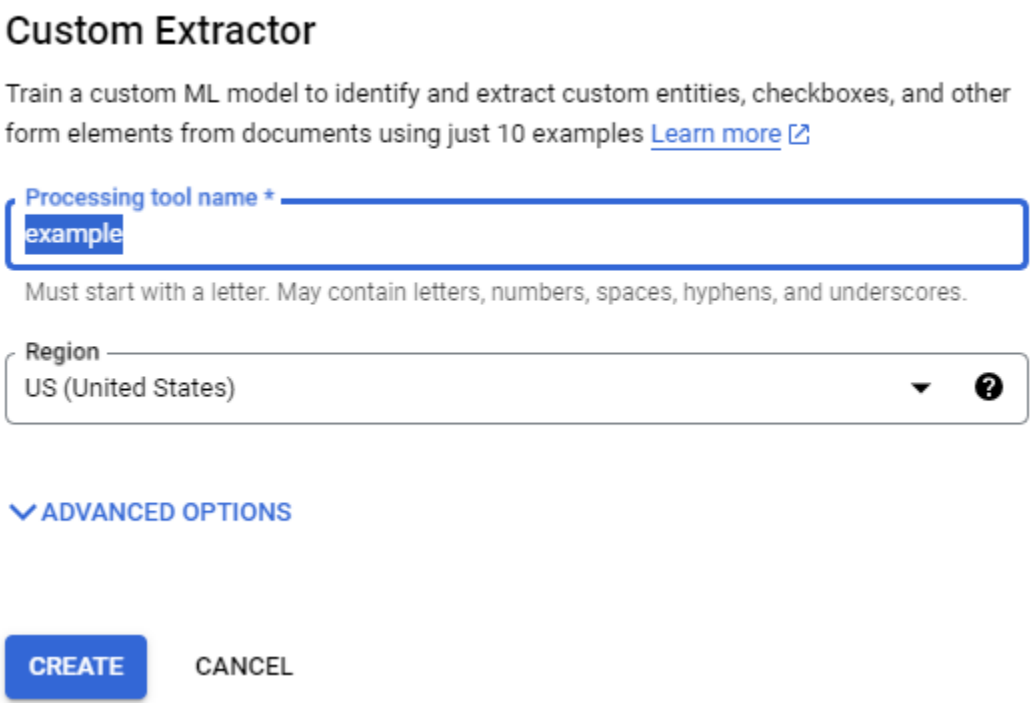


Creating Processor

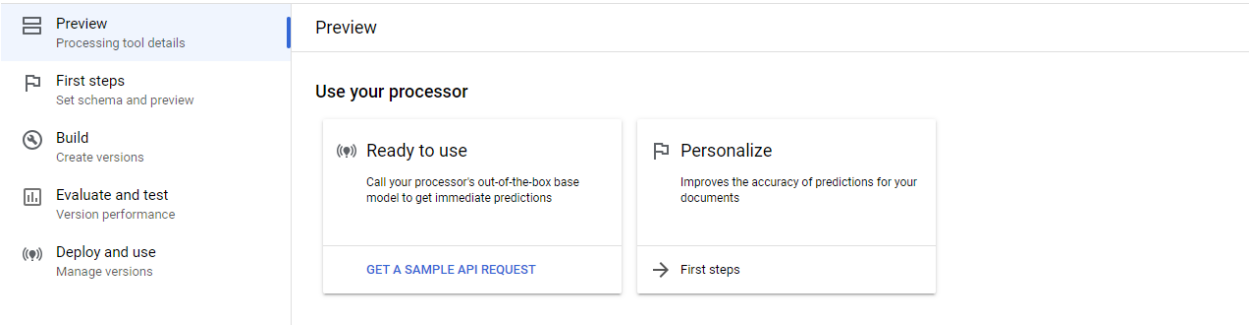
1. Under Custom Extractor click on “Create a Processor”



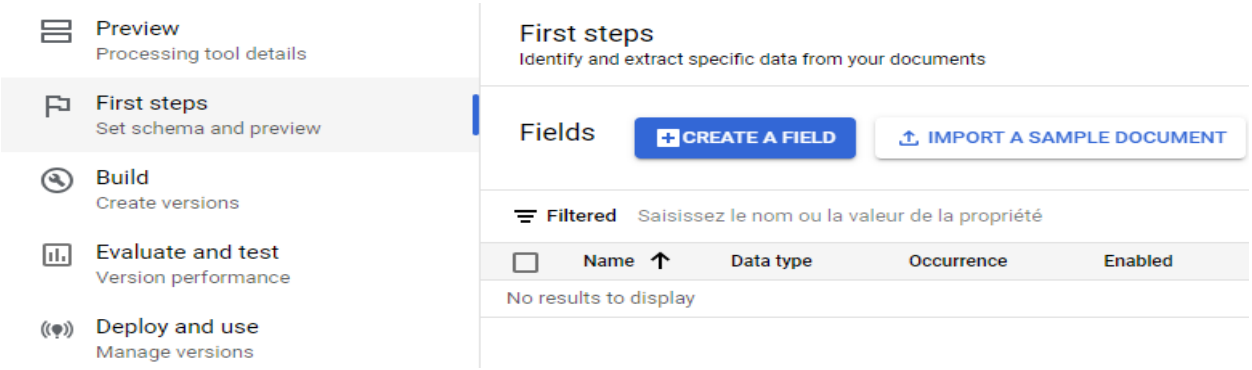
2. Give your processor name and then click on “Create”



3. After creating this screen will appear.



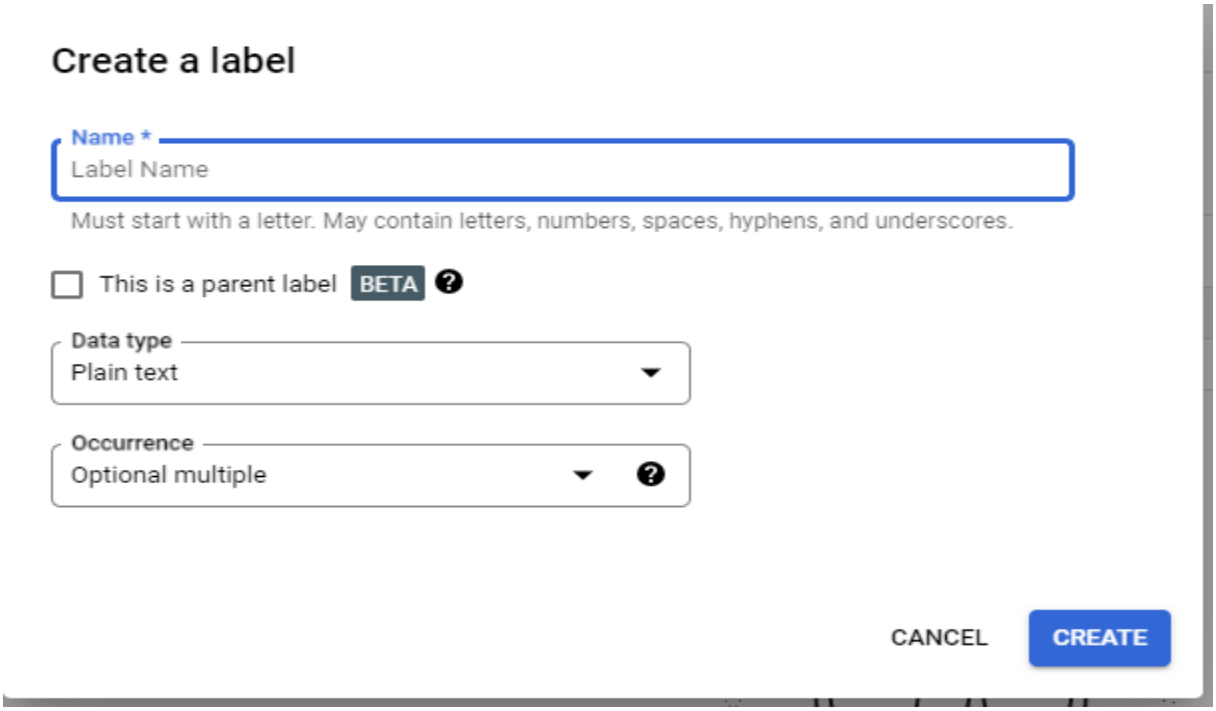
4. Then click on “First Steps”



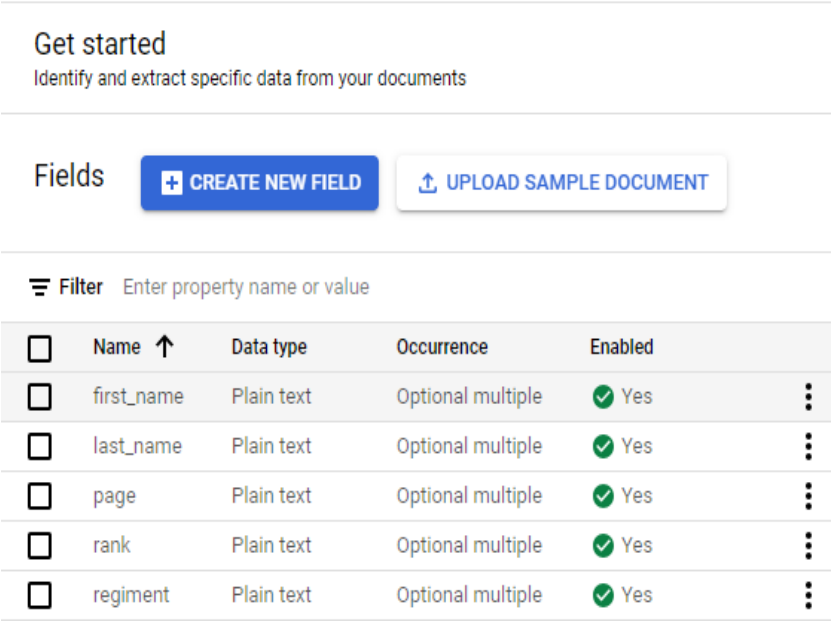
Schema Definition

Began by defining the schema of the documents in Document AI. This involved specifying the key information we wanted to extract, such as names, ranks, and regiments.

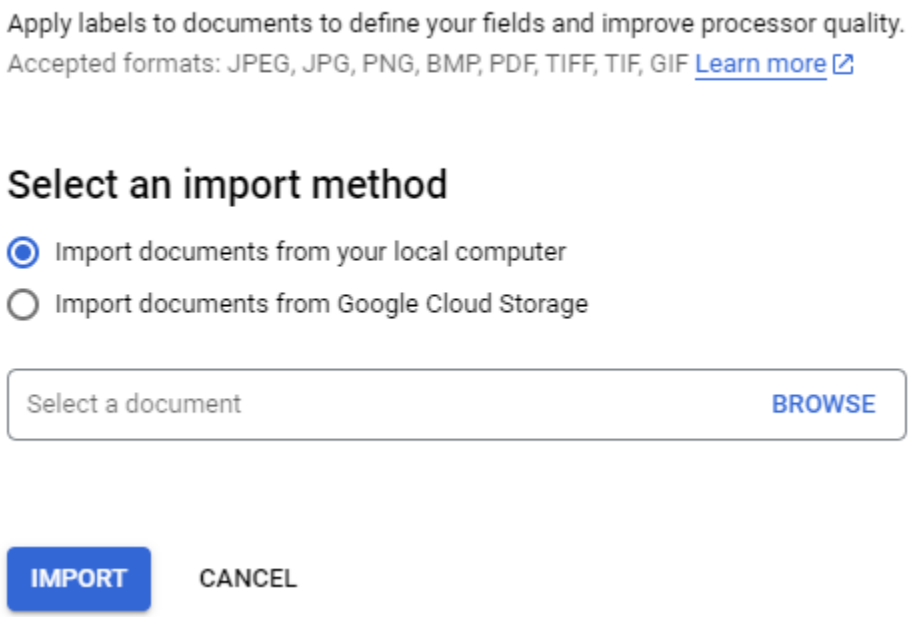
1. Click on “Create A Field” and define Label Name, Data type, and Occurrence.



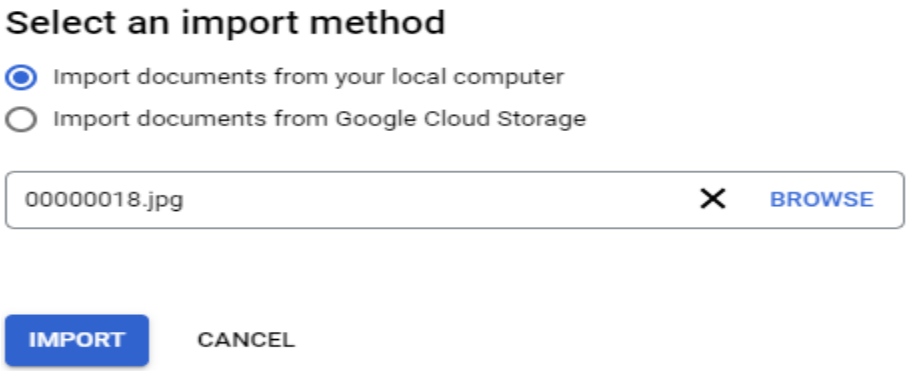
2. Define all the labels you want to extract from your data.



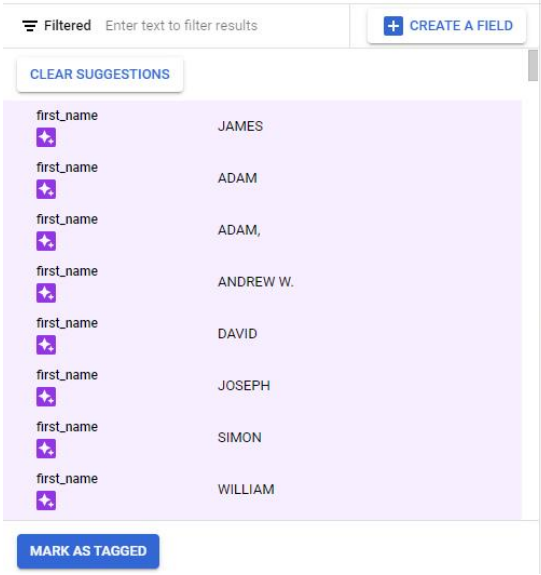
3. After creating fields click on “Upload Sample Document”



4. Now import a sample document from where ever you want, and then click “Import”



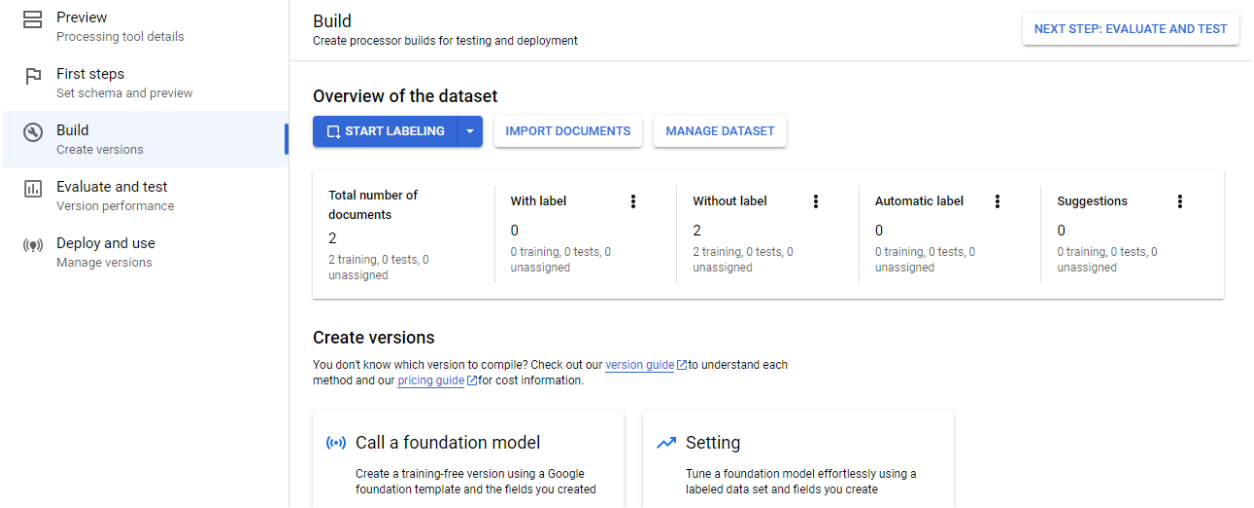
5. Label your document according to fields.



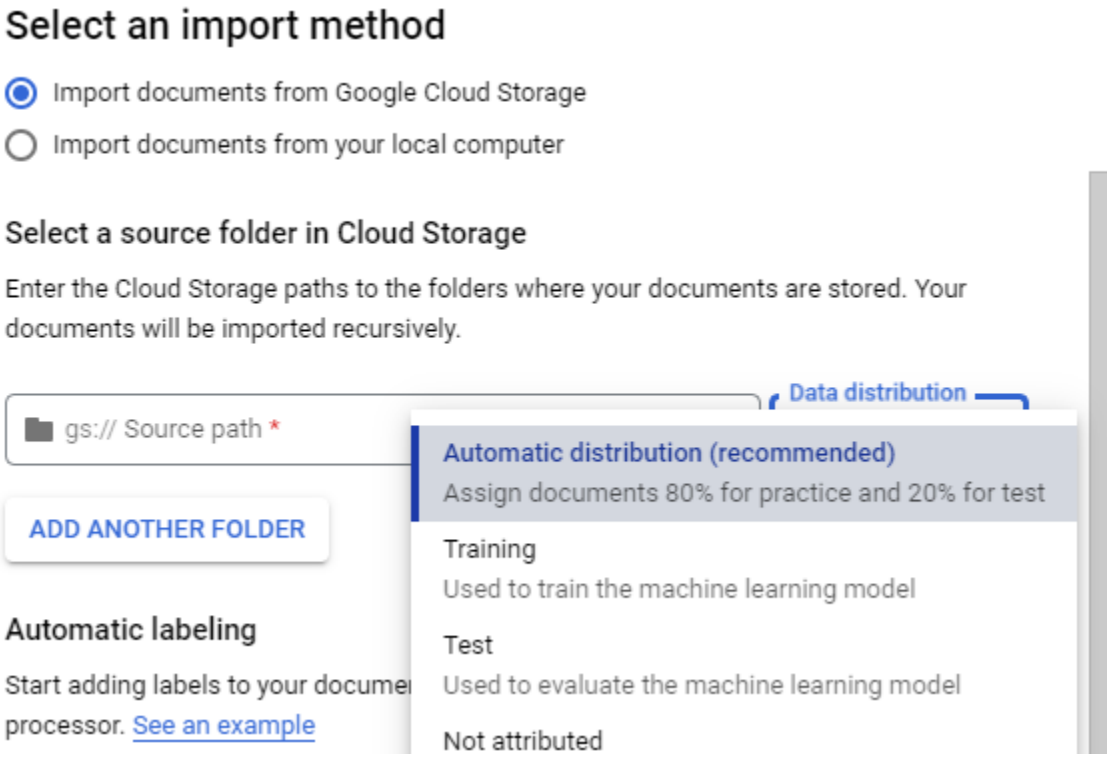
Data Preparation

Next, gathered a dataset of sample documents that were representative of the documents intended to process. These documents were manually labeled to identify the target entities and their locations within the documents.

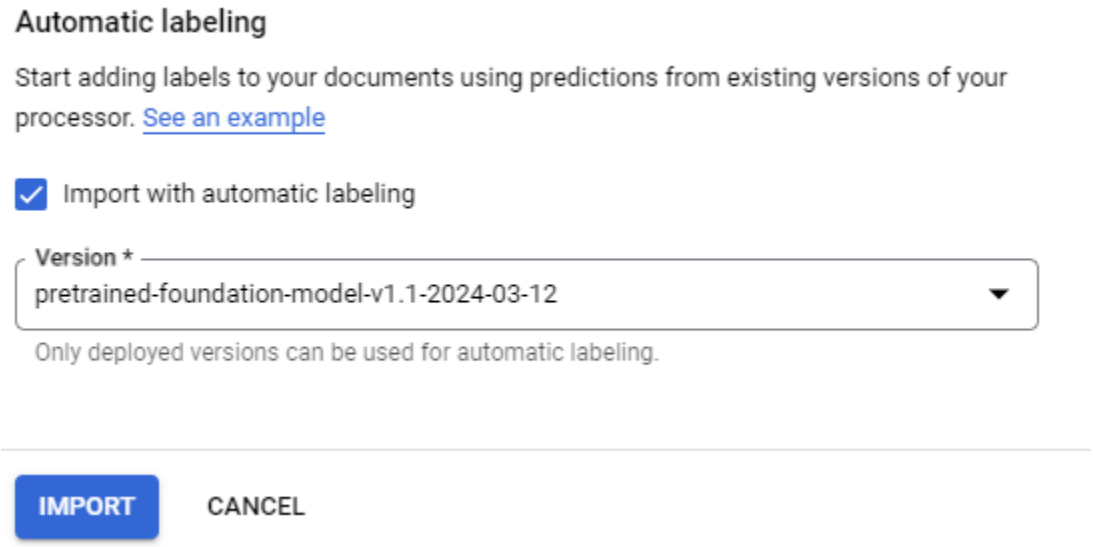
1. Click on “Build”. This screen will appear



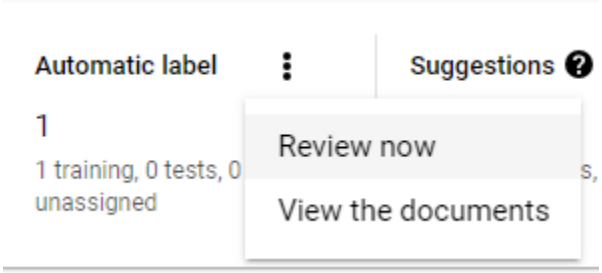
2. Click on “Import Documents” and give path of your documents, then select Data Select Data distribution.



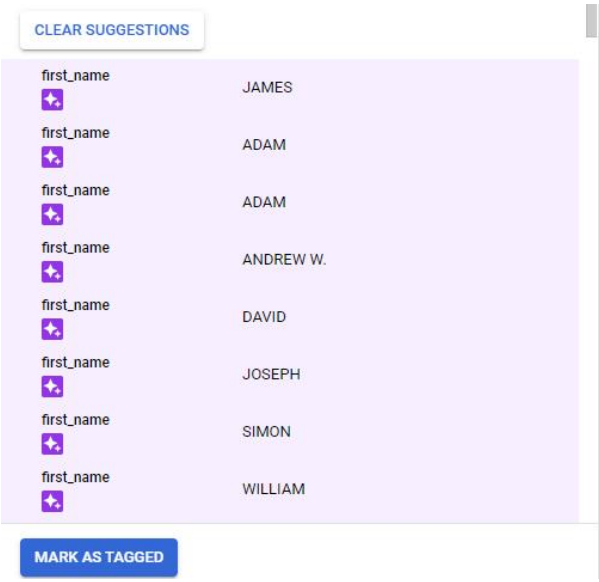
3. Then check “Import with automatic labeling”, and if you are importing document first time you can select pretrained version for automatic labeling, and then click on import.

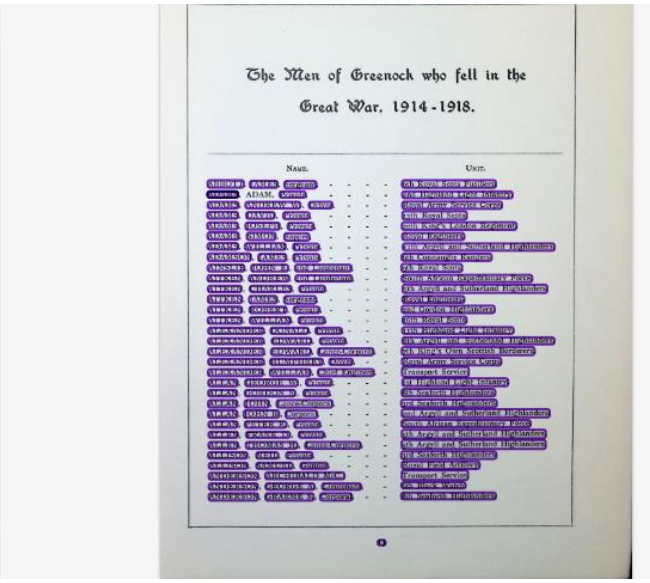


4. Then you have to review auto labeled data.

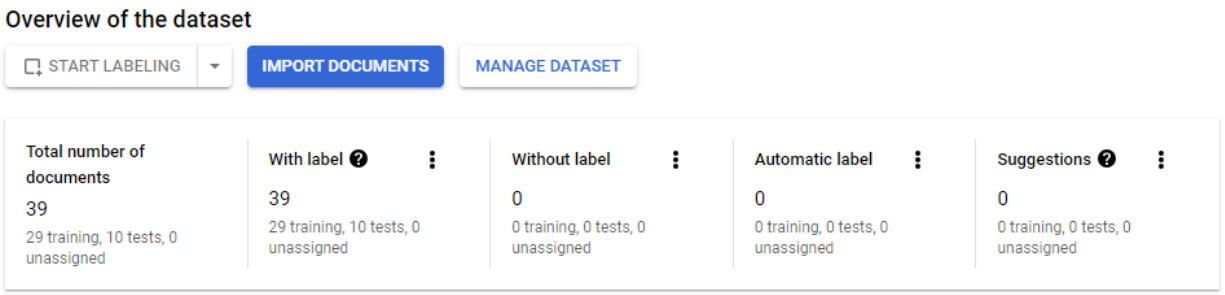


5. Check if all the fields are correctly labeled, if not then label incorrect or missing fields manually, and after doing labeling click on “Mark as Tagged”. Label all the documents.





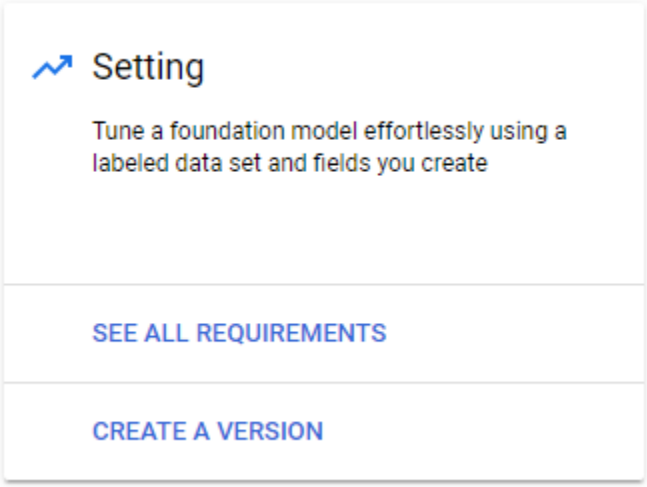
6. All the documents should be labeled.



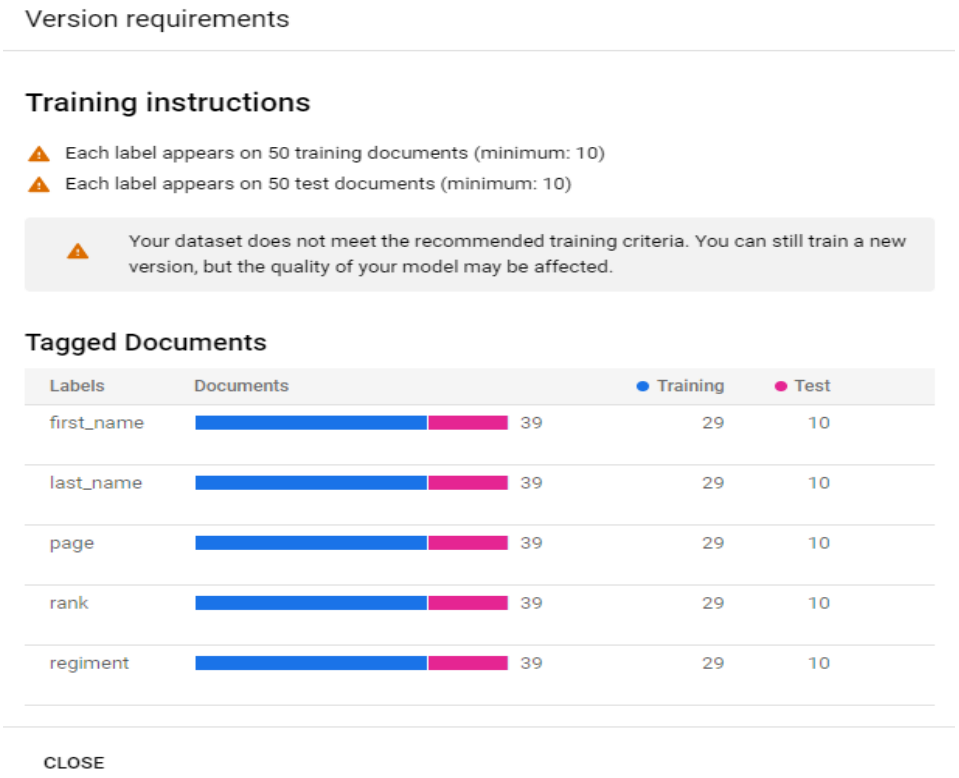
Training Version

With the labeled dataset prepared, initiated the fine-tuning process by creating version.

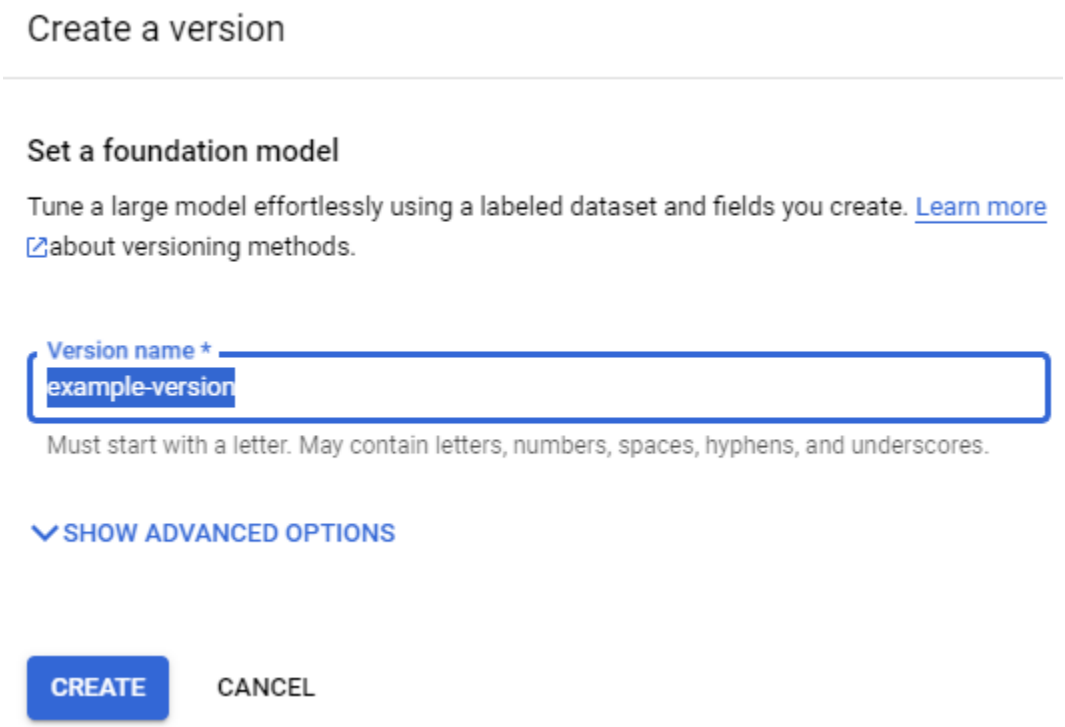
1. After labeling all the documents, we'll create a version.



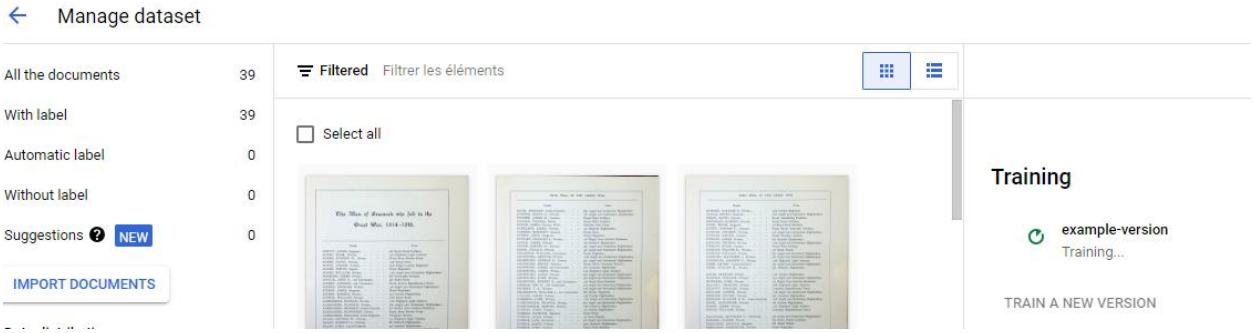
2. Before creating version see all requirements. You can only perform fine-tunning if your labels are meeting minimum requirements.



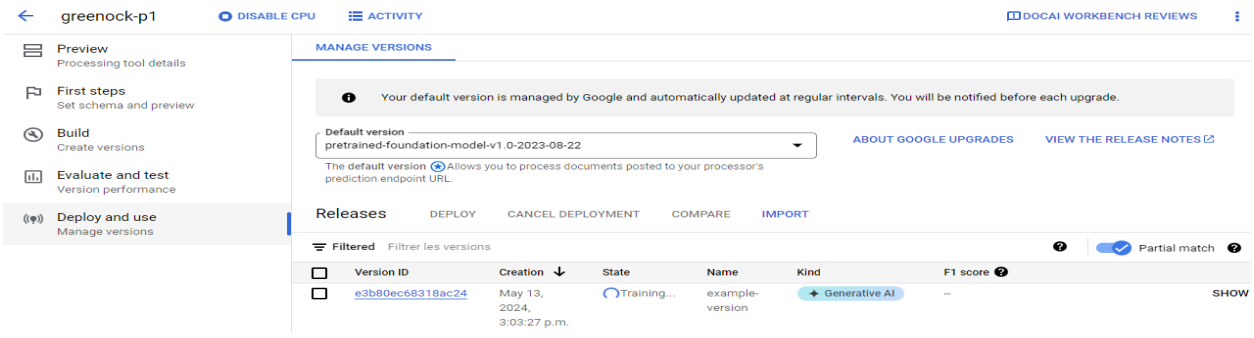
3. If you are meeting requirements, then create version by defining version name, and click “Create”.



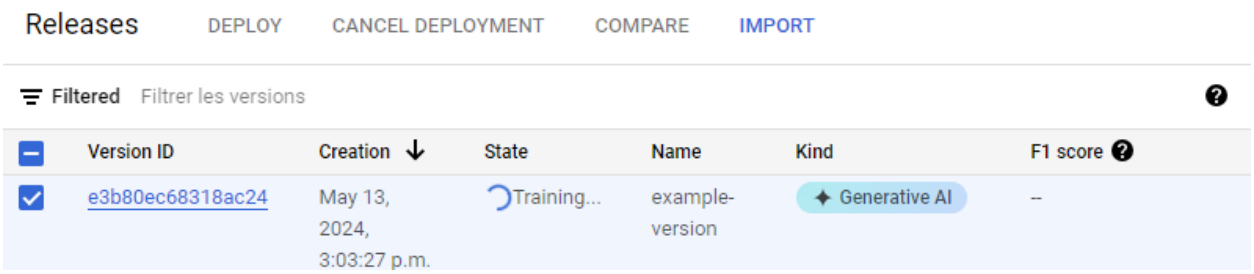
4. After creating version training will automatically gets start and you can view it in “Manage Dataset”



Or you can also view it in “Deploy and use” page



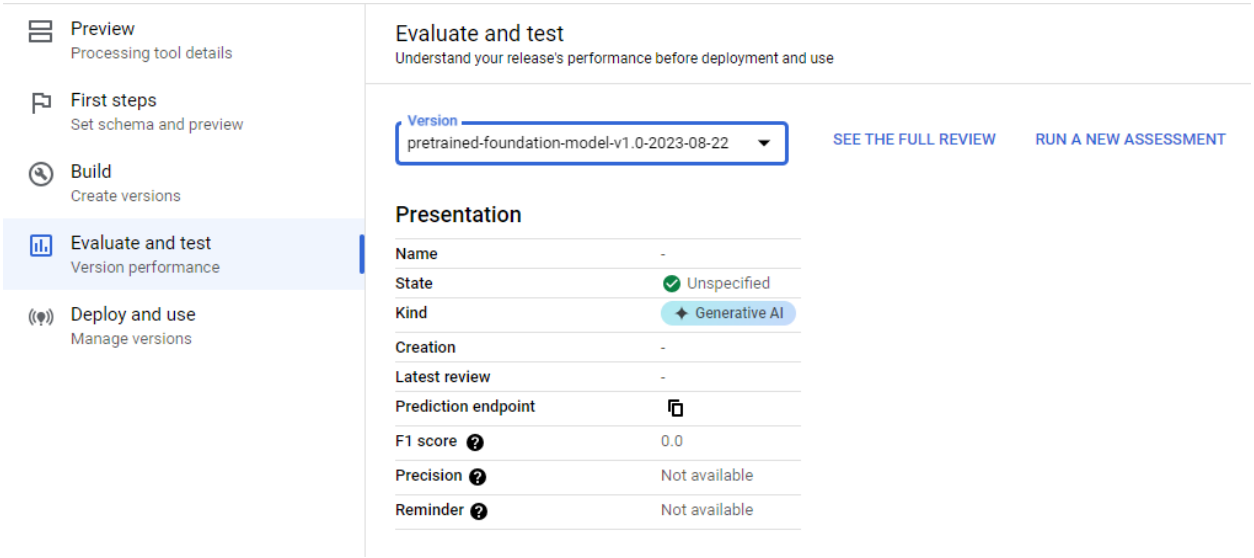
5. After completion of training, select the version and then “Deploy” to deploy the version.



Evaluation and Validation

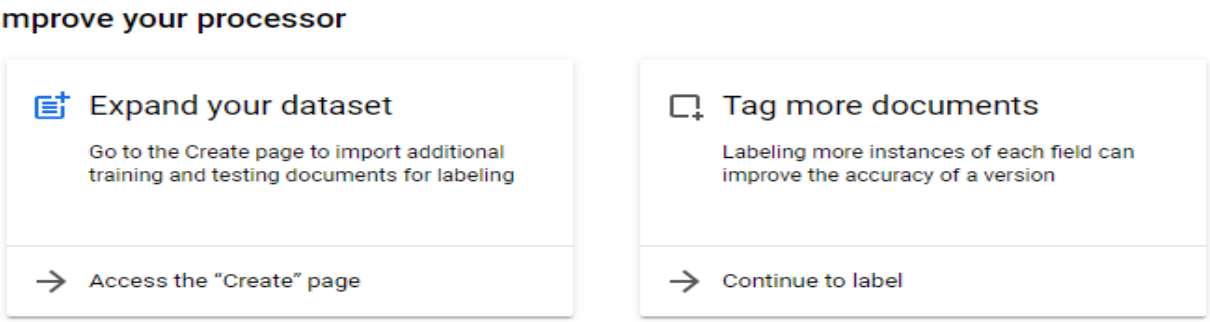
After training, done evaluation and validation tests to ensure the accuracy and effectiveness of the trained version. Tested the version and measured performance metrics such as F1 score.

1. After deploying go to “Evaluate and test” page to evaluate version by selecting it from dropdown.



2. Test your version on document by clicking on “Import a Test Document”.

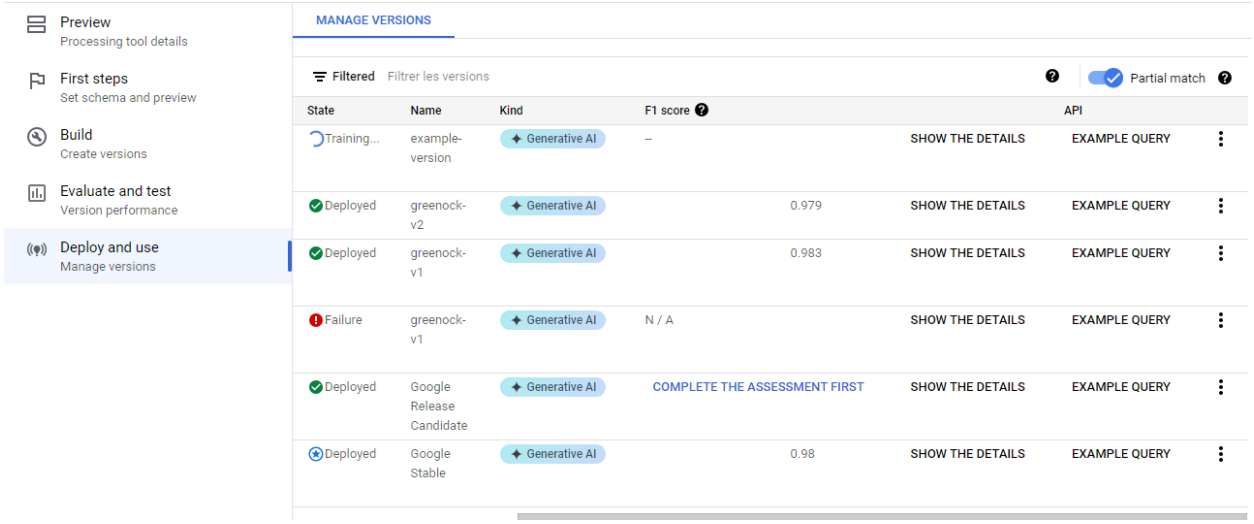
3. After evaluating and testing, if your version requires improvements then you have to expand your dataset and label more documents.



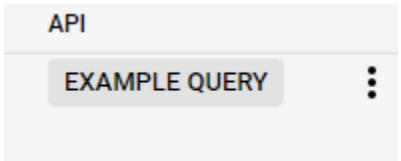
Processor Deployment

Once the version was trained and validated, deployed the Document AI processor to the Google Cloud platform. This allowed to utilize the processor programmatically to extract data from documents.

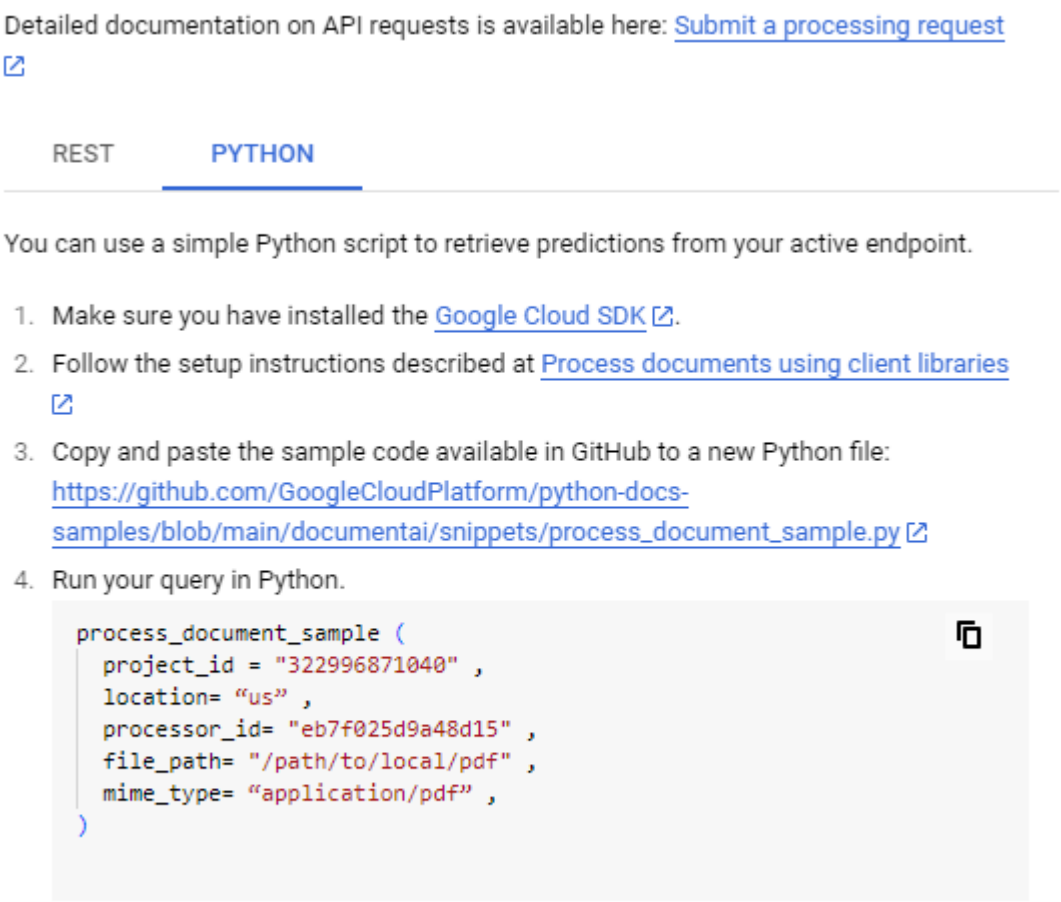
1. If your version has good F1-score and performing well on test documents. Then proceed to "Deploy and use" page again.



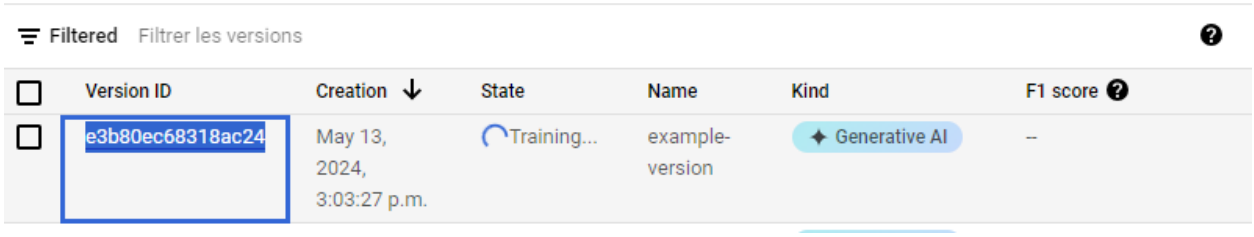
2. Click on "Example Query" of your selected version.



3. Click on “Python”.



4. You can copy processor version from here.



Integration with Content Extraction Script

In the content_extract.ipynb script, the Document AI processor is utilized to perform data extraction tasks. Here's how it integrates into the workflow:

1. **Initialization:** The script initializes the Document AI client and specifies the processor to use for document processing.
2. **Document Processing:** For each page of the PDF document, the script sends the image content to the Document AI processor for analysis. The processor applies OCR and extracts structured data based on the defined entity types.
3. **Data Extraction:** The extracted data, including entities like names, ranks, and regiments, is then processed further to format it appropriately and save it into a CSV file for easy analysis.

Usage

1. Open the content_extract.ipynb notebook in your Jupyter environment.
2. Provide the necessary input parameters such as project ID, location, processor ID, path (directory or file path), and processor version ID.
3. Execute the script to extract data from the PDF document.
4. Review the extracted data saved in the output CSV file.

```
determine_process(
    project_id="322996871040",
    location="us",
    processor_id="eb7f025d9a48d15",
    path="test",
    #processor_version_id="69f0b6febc1d5673"
)
```

The screenshot shows a code snippet where the default processor version is used in the determine_process function. This decision is based on testing, where it was found that the default version provided better results in terms of accuracy and efficiency compared to other versions.

Word Search

2. coordinates.ipynb

Purpose

This script is designed to locate and visualize the coordinates of specific words within a PDF document. It utilizes the Google Vision API for optical character recognition (OCR) and then identifies the coordinates of the specified words.

Dependencies

- **PyMuPDF:** For working with PDF documents.
- **Pillow:** For image processing tasks.
- **requests:** For making HTTP requests.

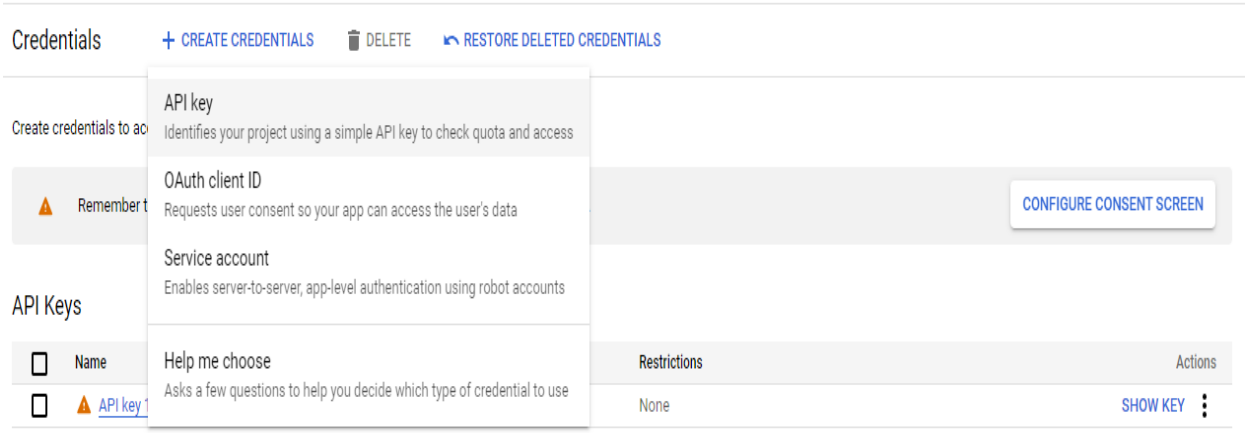
Functionality

1. **OCR and Data Extraction:** Extracts text and coordinates from the PDF document using the Google Vision API.
2. **Word Coordinate Search:** Searches for the specified word within the extracted data and retrieves its coordinates.
3. **Visualization:** Draws bounding boxes around the specified word on each page of the PDF document and saves the modified pages as images.

Coordinate Extraction and Bounding Box Drawing:

Text and Coordinate Extraction:

API Generation: The script utilizes the Google Cloud Vision API to process the PDF document. To use the API, we need to generate an API key through the Google Cloud Platform (GCP) Console. This API key is then used to authenticate requests sent to the Vision API.



API Usage: With the API key in hand, the script makes HTTP requests to the Vision API, passing the PDF document's image content. The API returns a JSON response containing the detected text and the bounding box coordinates of each word on every page of the document.


```
[{'text': 'The', 'vertices': [{'x': 106, 'y': 163}, {'x': 145, 'y': 163}, {'x': 145, 'y': 181}, {'x': 106, 'y': 181}], 'page':
1}, {'text': 'Men', 'vertices': [{'x': 154, 'y': 163}, {'x': 201, 'y': 163}, {'x': 201, 'y': 181}, {'x': 154, 'y': 181}], 'pag
e': 1}, {'text': 'of', 'vertices': [{'x': 212, 'y': 163}, {'x': 230, 'y': 163}, {'x': 230, 'y': 181}, {'x': 212, 'y': 181}], 'p
age': 1}, {'text': 'Greenock', 'vertices': [{'x': 239, 'y': 163}, {'x': 320, 'y': 163}, {'x': 320, 'y': 181}, {'x': 239, 'y': 1
81}], 'page': 1}, {'text': 'who', 'vertices': [{'x': 329, 'y': 163}, {'x': 362, 'y': 163}, {'x': 362, 'y': 181}, {'x': 329,
```


Storage of Coordinates:

CSV Storage: After extracting the text and coordinates from the Vision API response, the script save this data to a CSV file. The CSV file includes columns for the text, vertices, and of each word.

	text	vertices	page
1	The	[{'x': 106, 'y': 163}, {'x':...	1
2	Men	[{'x': 154, 'y': 163}, {'x':...	1
3	of	[{'x': 212, 'y': 163}, {'x':...	1
4	Greenock	[{'x': 239, 'y': 163}, {'x':...	1
5	who	[{'x': 329, 'y': 163}, {'x':...	1
6	fell	[{'x': 370, 'y': 164}, {'x':...	1
7	in	[{'x': 410, 'y': 164}, {'x':...	1
8	the	[{'x': 437, 'y': 164}, {'x':...	1
9	Great	[{'x': 174, 'y': 204}, {'x':...	1
10	War	[{'x': 234, 'y': 204}, {'x':...	1

Dictionary Storage: Alternatively, the extracted data stored in a dictionary format within the script's memory. Each dictionary entry contains the text, vertices, and page of a word, making it easy to access and manipulate the data programmatically.

Search and Match:

Specifying Search Word: Users can input the search word they want to locate. This word is then used to search through the extracted data for a match.

```
pdf_path = "test.pdf"
search_word = "abbott, james"
draw_bounding_box(pdf_path, search_word)
```

Iterative Search: The script iterates through the stored data, comparing each word against the specified search word. Matching words are identified based on text similarity or exact matches, depending on the search criteria.

```
def find_word_coordinates(words, coordinates):
    word_coordinates = []
    # Split words on the basis of space, comma, or period if it's a string
    if isinstance(words, str):
        words = re.split(r'[ ,.]', words)
    for word in words:
        for coord in coordinates:
            if coord["text"].lower() == word.lower():
                word_coordinates.append({
                    "text": coord["text"],
                    "vertices": coord["vertices"],
                    "page": coord["page"]
                })
    return word_coordinates
```

Drawing Bounding Boxes:

PIL Library Usage: Upon finding the coordinates of the specified word, the script utilizes the Python Imaging Library (PIL) to draw bounding boxes around the word on each page of the PDF document.

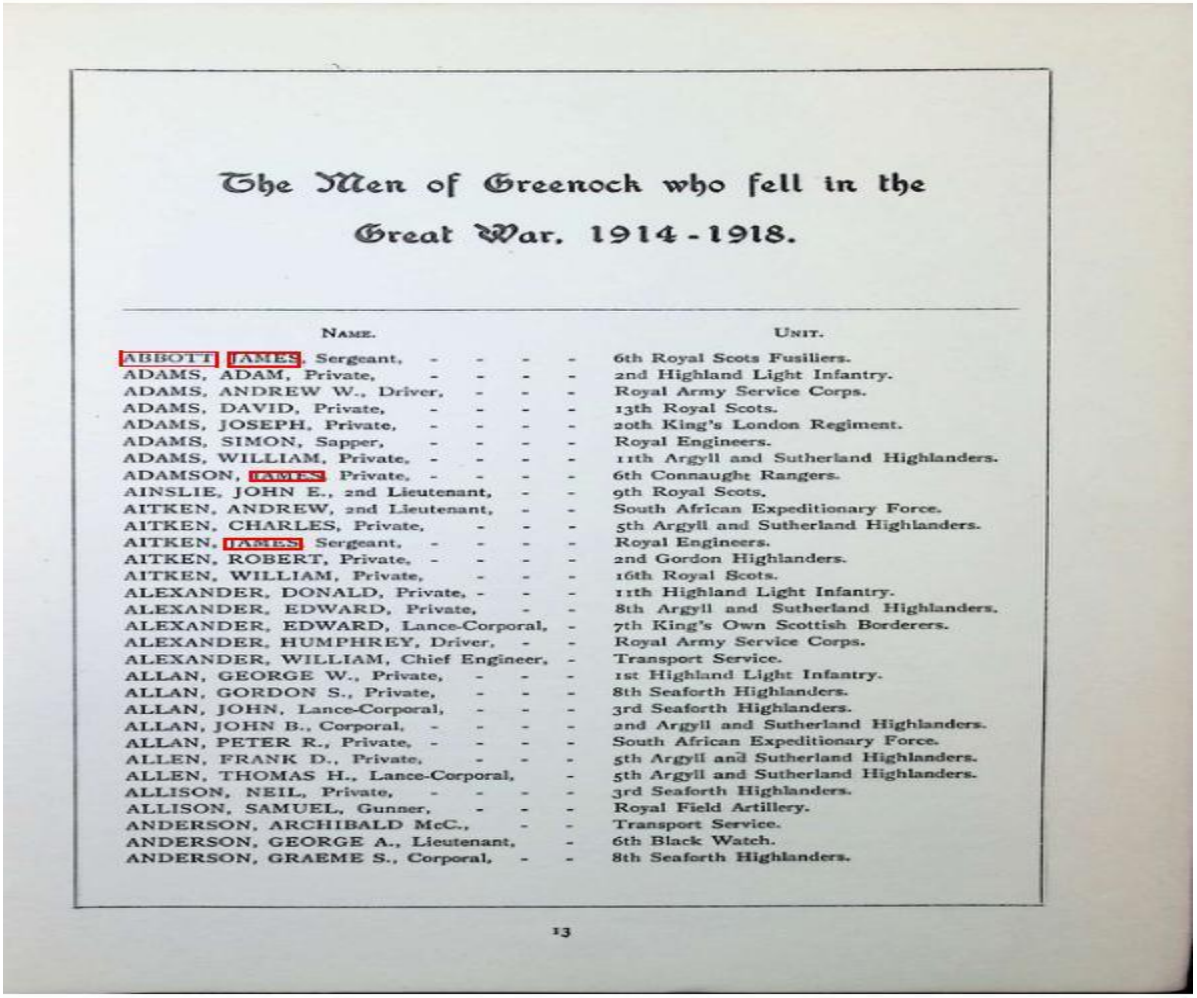
Bounding Box Drawing: The PIL library provides functions to draw rectangles (bounding boxes) around specified coordinates on images. The script creates a new image for each page of the PDF, overlays bounding boxes around the identified word, and saves the modified images.

```
pil_image = Image.open(io.BytesIO(image))
draw = ImageDraw.Draw(pil_image)
for coord in word_coordinates:
    if isinstance(coord, dict) and coord['page'] == page_number + 1:
        vertices = coord['vertices']
        for i in range(4):
            draw.line([(vertices[i]['x'], vertices[i]['y']), (vertices[(i + 1) % 4]['x'], vertices[(i + 1) % 4]['y'])],
output_image_path = f"output_page_{page_number + 1}.jpg"
if any(isinstance(coord, dict) and coord['page'] == page_number + 1 for coord in word_coordinates):
    print(f"Saving bounding box on page {page_number + 1}...")
    pil_image.save(output_image_path)
    print(f"Bounding box saved as '{output_image_path}'.")
```

Visualization and Verification:

Image Output: The modified pages with bounding boxes are saved as images, allowing users to visually inspect the locations of the specified word within the PDF document.

Verification Process: Users can review the generated images to verify the accuracy of the bounding box placement and ensure that the specified word has been correctly identified on each page.



The Men of Greenock who fell in the Great War, 1914-1918.

Name.	Unit.
ABBOTT, JAMES, Sergeant,	6th Royal Scots Fusiliers.
ADAMS, ADAM, Private,	2nd Highland Light Infantry.
ADAMS, ANDREW W., Driver,	Royal Army Service Corps.
ADAMS, DAVID, Private,	13th Royal Scots.
ADAMS, JOSEPH, Private,	20th King's London Regiment.
ADAMS, SIMON, Sapper,	Royal Engineers.
ADAMS, WILLIAM, Private,	11th Argyll and Sutherland Highlanders.
ADAMSON, JAMES, Private,	6th Connaught Rangers.
AINSLIE, JOHN E., 2nd Lieutenant,	9th Royal Scots.
AITKEN, ANDREW, 2nd Lieutenant,	South African Expeditionary Force.
AITKEN, CHARLES, Private,	5th Argyll and Sutherland Highlanders.
AITKEN, JAMES, Sergeant,	Royal Engineers.
AITKEN, ROBERT, Private,	2nd Gordon Highlanders.
AITKEN, WILLIAM, Private,	16th Royal Scots.
ALEXANDER, DONALD, Private,	11th Highland Light Infantry.
ALEXANDER, EDWARD, Private,	8th Argyll and Sutherland Highlanders.
ALEXANDER, EDWARD, Lance-Corporal,	7th King's Own Scottish Borderers.
ALEXANDER, HUMPHREY, Driver,	Royal Army Service Corps.
ALEXANDER, WILLIAM, Chief Engineer,	Transport Service.
ALLAN, GEORGE W., Private,	1st Highland Light Infantry.
ALLAN, GORDON S., Private,	8th Seaforth Highlanders.
ALLAN, JOHN, Lance-Corporal,	3rd Seaforth Highlanders.
ALLAN, JOHN B., Corporal,	2nd Argyll and Sutherland Highlanders.
ALLAN, PETER R., Private,	South African Expeditionary Force.
ALLEN, FRANK D., Private,	5th Argyll and Sutherland Highlanders.
ALLEN, THOMAS H., Lance-Corporal,	5th Argyll and Sutherland Highlanders.
ALLISON, NEIL, Private,	3rd Seaforth Highlanders.
ALLISON, SAMUEL, Gunner,	Royal Field Artillery.
ANDERSON, ARCHIBALD McC.,	Transport Service.
ANDERSON, GEORGE A., Lieutenant,	6th Black Watch.
ANDERSON, GRAEME S., Corporal,	8th Seaforth Highlanders.

13

Usage

1. Open the coordinates.ipynb notebook in your Jupyter environment.
2. Provide the PDF file path and the search word.
3. Execute the script to visualize the bounding boxes around the specified word in the PDF document.
4. Check the generated images with bounding boxes to verify the locations of the search word.