Task Report : Capital Placement

Objective: Build a PDF extractor to pull relevant details from CVs in PDF format, and match them against the job descriptions from the Hugging Face dataset.

I have taken near 13-15 job description from hugging face dataset and converted into pdf format

Approach 1- As mentioned in task description

Importing necessary libraries and modules:

1. Library Imports: The code imports essential libraries such as os, pandas, pdfplumber, transformers (for DistilBERT), CountVectorizer, cosine_similarity, and numpy.

2. Defining Paths and Functions: It defines paths to the job descriptions PDF file and the folder containing candidate resumes in PDF format. A function, extract_text_from_pdf, is available to extract text from PDF files.

3. Initializing DistilBERT: The code initializes DistilBERT tokenizer and model using the transformers library.

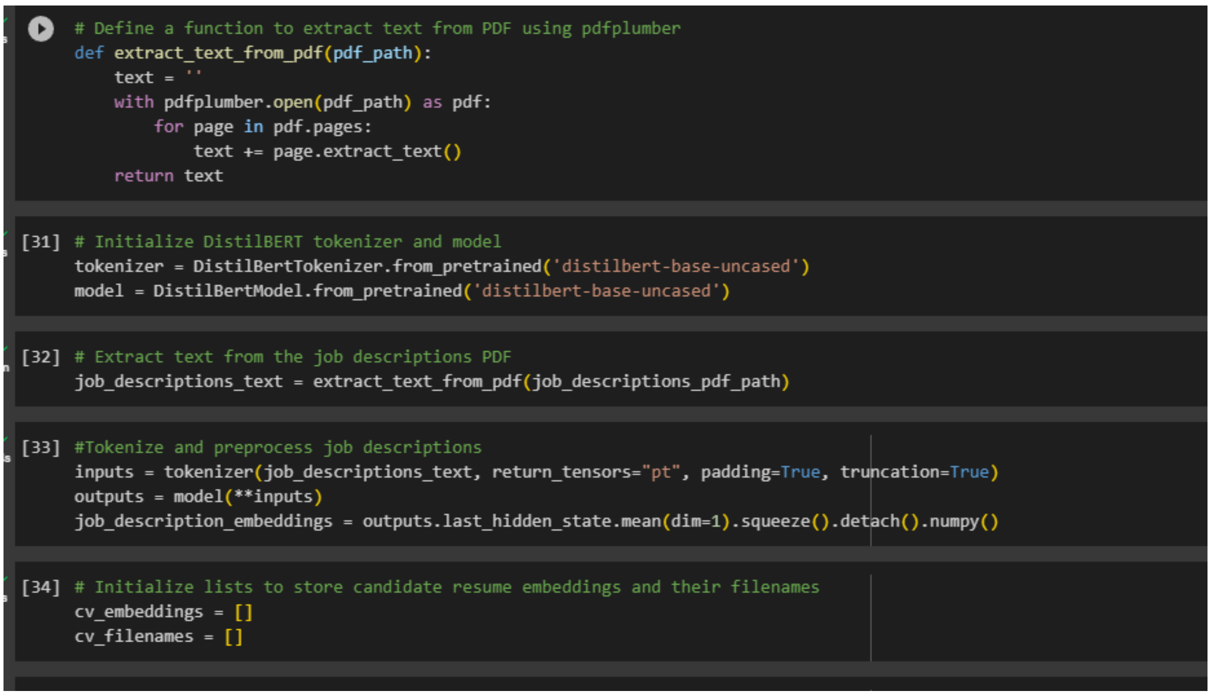
4. Extracting and Processing Job Descriptions: It extracts text from the job descriptions PDF file and processes it to obtain embeddings using DistilBERT.

5. Initializing Candidate Resume Embeddings List: An empty list (cv_embeddings) is initialized to store candidate resume embeddings.

6. Iterating Through Candidate Resumes: The code iterates through PDF files in the specified folder, extracts text from each candidate's resume, tokenizes and processes the text to obtain embeddings, and appends these embeddings to cv_embeddings.

7. Calculating Cosine Similarity: It calculates the cosine similarity between each candidate's resume and the job descriptions to determine how closely each resume matches the job descriptions.

8. Displaying Matching Scores: Finally, the code displays the matching scores for each candidate's resume, indicating the similarity between each resume and the job descriptions.



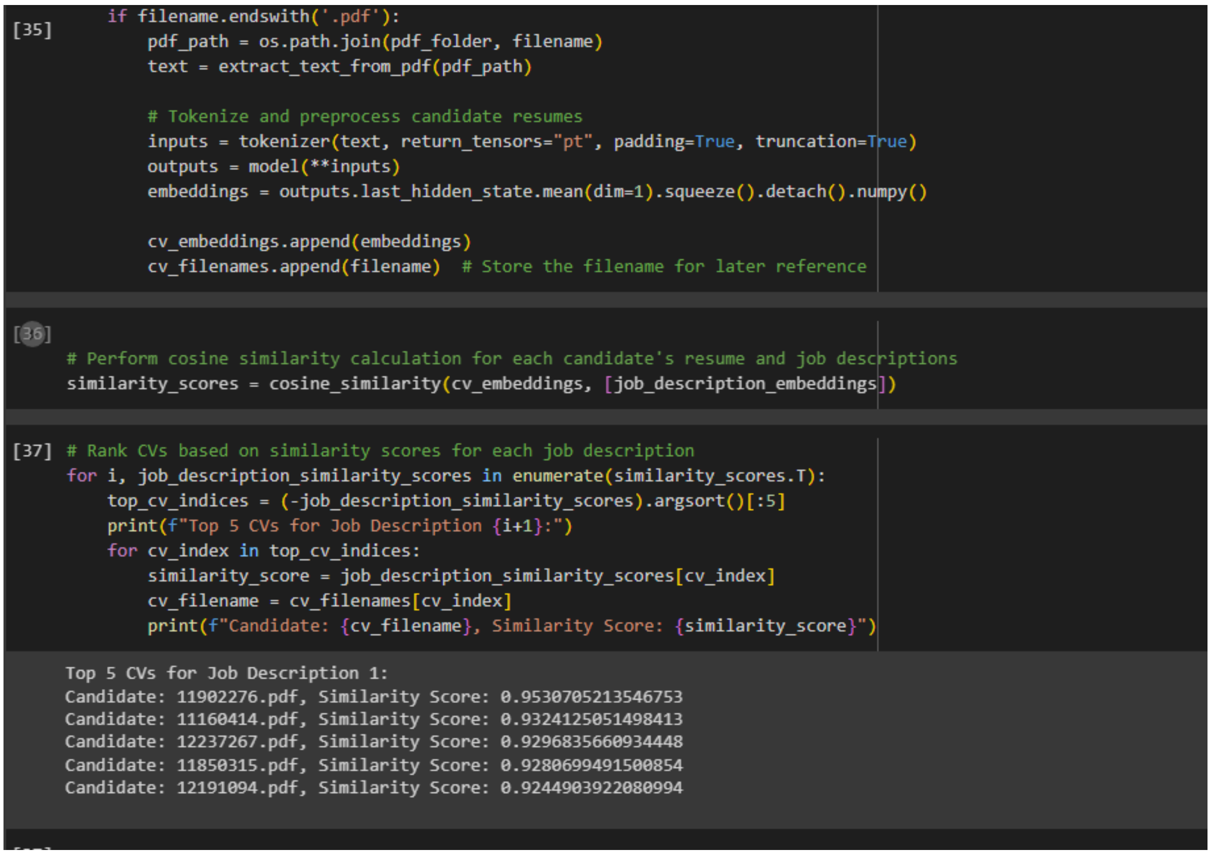
```
# Define a function to extract text from PDF using pdfplumber
def extract_text_from_pdf(pdf_path):
    text = ''
    with pdfplumber.open(pdf_path) as pdf:
        for page in pdf.pages:
            text += page.extract_text()
    return text
```

```
[31] # Initialize DistilBERT tokenizer and model
     tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
     model = DistilBertModel.from_pretrained('distilbert-base-uncased')
```

```
[32] # Extract text from the job descriptions PDF
     job_descriptions_text = extract_text_from_pdf(job_descriptions_pdf_path)
```

```
[33] #Tokenize and preprocess job descriptions
     inputs = tokenizer(job_descriptions_text, return_tensors="pt", padding=True, truncation=True)
     outputs = model(**inputs)
     job_description_embeddings = outputs.last_hidden_state.mean(dim=1).squeeze().detach().numpy()
```

```
[34] # Initialize lists to store candidate resume embeddings and their filenames
     cv_embeddings = []
     cv_filenames = []
```


```
[35]        if filename.endswith('.pdf'):
                pdf_path = os.path.join(pdf_folder, filename)
                text = extract_text_from_pdf(pdf_path)

                # Tokenize and preprocess candidate resumes
                inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True)
                outputs = model(**inputs)
                embeddings = outputs.last_hidden_state.mean(dim=1).squeeze().detach().numpy()

                cv_embeddings.append(embeddings)
                cv_filenames.append(filename)  # Store the filename for later reference
```

```
[36]
    # Perform cosine similarity calculation for each candidate's resume and job descriptions
    similarity_scores = cosine_similarity(cv_embeddings, [job_description_embeddings])
```

```
[37] # Rank CVs based on similarity scores for each job description
    for i, job_description_similarity_scores in enumerate(similarity_scores.T):
        top_cv_indices = (-job_description_similarity_scores).argsort()[:5]
        print(f"Top 5 CVs for Job Description {i+1}:")
        for cv_index in top_cv_indices:
            similarity_score = job_description_similarity_scores[cv_index]
            cv_filename = cv_filenames[cv_index]
            print(f"Candidate: {cv_filename}, Similarity Score: {similarity_score}")
```

```
Top 5 CVs for Job Description 1:
Candidate: 11902276.pdf, Similarity Score: 0.9530705213546753
Candidate: 11160414.pdf, Similarity Score: 0.9324125051498413
Candidate: 12237267.pdf, Similarity Score: 0.9296835660934448
Candidate: 11850315.pdf, Similarity Score: 0.9280699491500854
Candidate: 12191094.pdf, Similarity Score: 0.9244903922080994
```

Approach 2 – This approach will divide the job description and canditate's resume into two list and then it will match them but the thing is here I have to check for each and every pdf it will not iterate over all the pdf in the file.

1. Library Imports and Setup: The code begins by importing essential libraries, including NumPy for numerical operations and Pandas for data manipulation.It sets up the directory structure by iterating through files in a specified directory.
2. PDF Generation from CSV Data: It uses the FPDF library to create a PDF document (job_descriptions.pdf) where job descriptions will be stored.The font and size for the PDF are configured to ensure proper formatting.
3. Loading CSV Data: The code loads job descriptions from a CSV file (training_data.csv) into a Pandas DataFrame (df).

4. Populating the PDF with Job Descriptions: It iterates through the rows of the CSV file, extracting job titles and descriptions. Job titles and

descriptions are added to the PDF using pdf.cell() and pdf.multi_cell() functions. Encoding is applied to handle special characters.

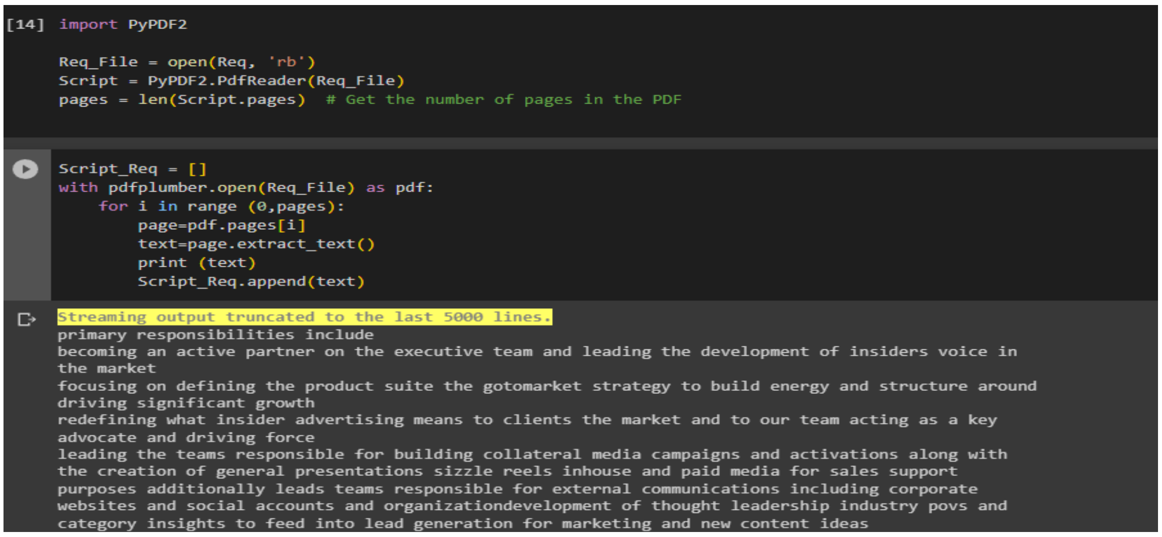
5. Saving the PDF: After all job descriptions are added to the PDF, it is saved as job_descriptions.pdf in the specified directory. Text Extraction from Candidate's Resume (CV) and Job Descriptions PDF: It extracts text from a specific candidate's CV (55104715.pdf) using the pdfplumber library, cleaning and formatting the text.

6. Text Cleaning for CV and Job Descriptions PDF: Text extracted from both the CV and job descriptions PDF is cleaned by removing line breaks and replacing them with empty strings.

7. Calculating Text Similarity: The code calculates the similarity between the cleaned CV text and the cleaned job descriptions text using cosine similarity. It uses the CountVectorizer from scikit-learn to convert the text into numerical vectors. The cosine_similarity function calculates the cosine similarity score between the CV and job descriptions text vectors.

8. Storing Extracted Text in Script Variables:Text extracted from both the CV and job descriptions PDF is stored in Script and Script_Req variables, respectively, for further processing.

9. Displaying Similarity Score: Finally, the code prints the calculated similarity score, indicating how closely the CV matches the job descriptions in terms of textual content.

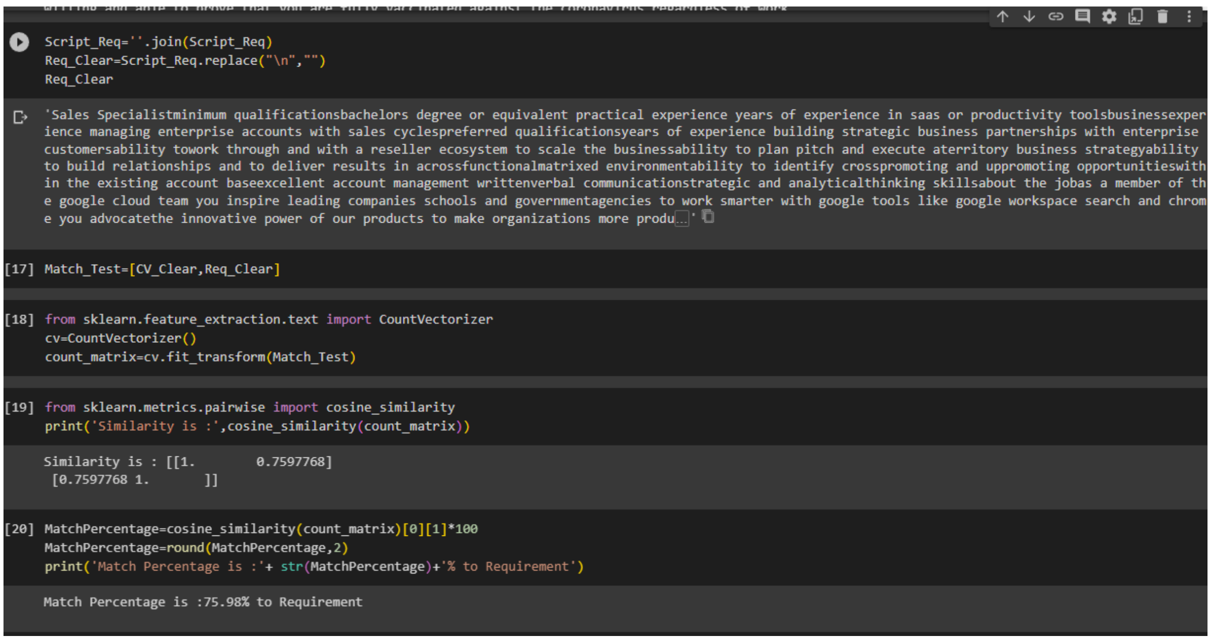


```
[14] import PyPDF2

     Req_File = open(Req, 'rb')
     Script = PyPDF2.PdfReader(Req_File)
     pages = len(Script.pages)  # Get the number of pages in the PDF
```

```
     Script_Req = []
     with pdfplumber.open(Req_File) as pdf:
         for i in range (0,pages):
             page=pdf.pages[i]
             text=page.extract_text()
             print (text)
             Script_Req.append(text)
```

```
Streaming output truncated to the last 5000 lines.
primary responsibilities include
becoming an active partner on the executive team and leading the development of insiders voice in
the market
focusing on defining the product suite the gotomarket strategy to build energy and structure around
driving significant growth
redefining what insider advertising means to clients the market and to our team acting as a key
advocate and driving force
leading the teams responsible for building collateral media campaigns and activations along with
the creation of general presentations sizzle reels inhouse and paid media for sales support
purposes additionally leads teams responsible for external communications including corporate
websites and social accounts and organizationdevelopment of thought leadership industry povs and
category insights to feed into lead generation for marketing and new content ideas
```



```
     Script_Req=''.join(Script_Req)
     Req_Clear=Script_Req.replace("\n","")
     Req_Clear
```

```
'Sales Specialistminimum qualificationsbachelors degree or equivalent practical experience years of experience in saas or productivity toolsbusinessexper
ience managing enterprise accounts with sales cyclespreferred qualificationsyears of experience building strategic business partnerships with enterprise
customersability towork through and with a reseller ecosystem to scale the businessability to plan pitch and execute aterritory business strategyability
to build relationships and to deliver results in acrossfunctionalmatrixed environmentability to identify crosspromoting and uppromoting opportunitieswith
in the existing account baseexcellent account management writtenverbal communicationstrategic and analyticalthinking skillsabout the jobas a member of th
e google cloud team you inspire leading companies schools and governmentagencies to work smarter with google tools like google workspace search and chrom
e you advocatethe innovative power of our products to make organizations more produ...'
```

```
[17] Match_Test=[CV_Clear,Req_Clear]
```

```
[18] from sklearn.feature_extraction.text import CountVectorizer
     cv=CountVectorizer()
     count_matrix=cv.fit_transform(Match_Test)
```

```
[19] from sklearn.metrics.pairwise import cosine_similarity
     print('Similarity is :',cosine_similarity(count_matrix))
```

```
Similarity is : [[1.        0.7597768]
 [0.7597768 1.        ]]
```

```
[20] MatchPercentage=cosine_similarity(count_matrix)[0][1]*100
     MatchPercentage=round(MatchPercentage,2)
     print('Match Percentage is :'+ str(MatchPercentage)+'% to Requirement')
```

```
Match Percentage is :75.98% to Requirement
```