TASK 1: PDF Data Extraction

Objective: The objective of the Task 1 is to extract specific details from multiple CVs (resumes) in PDF format. The details to be extracted include "Category" (job role), "Skills," and "Education" from each CV. The code aims to automate this data extraction process, making it efficient and consistent.

Dataset: The dataset for this task has been collected from Kaggle, Resume dataset: https://www.kaggle.com/datasets/snehaanbhawal/resume-dataset

Libraries used:

1. os: This module provides a way to interact with the operating system, specifically for file and directory handling. It's used to navigate the directory containing the CV PDFs.
2. PyPDF2: PyPDF2 is a library for extracting text and performing various operations on PDF files. In this code, it's used to read PDF files and extract text from them.
3. re: The 're' module is for regular expressions, which are used to search for specific patterns (such as "Category," "Skills," and "Education") within the extracted text.

Approach:

Function Definitions:

- extract_cv_details(pdf_path): This function takes the path of a PDF file (CV) as input and extracts the relevant details (Category, Skills, and Education) using regular expressions. The extracted details are stored in a dictionary.
- process_cvs_in_directory(directory_path): This function processes all CVs within a specified directory. It iterates through the files in the directory, identifies PDF files, and calls extract_cv_details for each CV, collecting the results in a list.

PDF Text Extraction:

- The code uses PyPDF2 to open each PDF file, read its pages, and extract text from each page.

Regular Expressions:

- It employs case-insensitive regular expressions to search for specific sections in the extracted text, such as "Category," "Skills," and "Education."

Data Storage:

- Extracted details from each CV are stored in a dictionary (cv_details), and a list (cv_details_list) accumulates these dictionaries for all CVs.

Directory Processing:

- The code navigates through the specified directory (cv_directory), identifying PDF files, and extracting CV details using the defined functions.

Challenges Faced and Solutions:

- Some PDFs had complex layouts, leading to extraction issues. We addressed this by fine-tuning our extraction code to handle various layouts.
- Inconsistent formatting of CVs required custom regular expressions to accurately extract information.

Outcome:

The outcome of Task 1 is the extraction of specific details from multiple CVs in PDF format. These details include the job category (role), skills, and education qualifications of each candidate. The code processes all CVs within the specified directory and stores the extracted information in a structured format.

This code facilitates the automation of data extraction from CVs, making it a valuable tool for tasks such as candidate profiling, job matching, and analysis. It enables the efficient processing of CVs at scale, saving time and effort for HR professionals and recruiters. The extracted data can be further analyzed or used for matching candidates with job descriptions based on their qualifications and skills.

TASK 2: JOB DESCRIPTION DATA UNDERSTANDING

Objective: Fetch and comprehend job descriptions from the Hugging Face dataset. The objective is to load a dataset of job descriptions and extract a subset of these job descriptions for further analysis. This task is part of the initial exploration and understanding of the job descriptions dataset.

Dataset: Hugging Face Job Descriptions Dataset, https://huggingface.co/datasets/jacob-hugging-face/job-descriptions/viewer/default/train?row=0

Libraries:

- datasets: The datasets library, provided by Hugging Face, is used to load and manage datasets. It simplifies the process of downloading, accessing, and preprocessing various datasets for natural language processing tasks.

Approach:

Library Import:

- Import the datasets library, which is used to load and manage datasets conveniently.

Dataset Loading:

- We load the job descriptions dataset using the datasets.load_dataset function. The dataset name ("jacob-hugging-face/job-descriptions") should be replaced with the actual dataset name. This function fetches the dataset from the Hugging Face repository.

Data Extraction:

- We extract a subset of job descriptions from the loaded dataset. Specifically, we extract the "job_description" field from the training split of the dataset.
- The code snippet dataset['train']['job_description'][:15] extracts the first 15 job descriptions from the training split.

Printing Job Descriptions:

- Iterate through the extracted job descriptions and print them one by one, including an identifier for each description.

Outcome:

The outcome of this code is the retrieval and printing of a subset of job descriptions from the dataset. This allows us to gain an initial understanding of the nature and content of the job descriptions. The printed job descriptions can be further analyzed or used as input for various natural language processing tasks, such as text classification, summarization, or matching candidates to job postings.

TASK 3: CANDIDATE-JOB MATCHING

Objective: The objective of Task 3 is to perform candidate-job matching by comparing job descriptions with the details extracted from CVs. The code uses the DistilBERT model to create text embeddings for both job descriptions and CVs, calculates cosine similarity between these embeddings, and returns the top 5 matches for each job description.

Tools Used:

1. transformers: This library is used to access pre-trained natural language processing models, including DistilBERT. It provides an easy interface for tokenization and model loading.
2. torch: PyTorch is used for handling tensor operations and working with deep learning models.
3. sklearn.metrics.pairwise.cosine_similarity: This module from scikit-learn is used to calculate the cosine similarity between two embeddings.

Approach:

1. Initialization: The code initializes the DistilBERT tokenizer and model, loading the pre-trained 'distilbert-base-uncased' model.
2. Text Embedding: The text_to_embedding function tokenizes the input text using the tokenizer and converts it into embeddings using the DistilBERT model. It returns the mean of the last hidden state as the text embedding.
3. Cosine Similarity: The calculate_similarity function calculates the cosine similarity between two embeddings using scikit-learn's cosine_similarity method.
4. Matching Process: The code processes job descriptions and CVs to find top matches. For each job description, it follows these steps:
    - Tokenizes and embeds the job description.
    - Iterates through CV details, which include the Category, Skills, and Education.
    - Combines the CV details into a single text string.
    - Tokenizes and embeds the CV text.
    - Calculates the cosine similarity between the job description embedding and CV embedding.
    - Stores the CV details and their corresponding similarity scores in a list.
    - Sorts the CVs by similarity in descending order and selects the top 5 matches.

Outcome:

This task produces a list of top 5 matching CVs for each job description, along with their similarity scores. The output is structured with headings for each job description and includes similarity scores and CV details for the top matches. This information can be valuable for recruiters or hiring managers to identify candidates whose skills and qualifications align closely with specific job roles.

The code demonstrates how to leverage natural language processing and deep learning techniques to automate the candidate-job matching process, making it more efficient and objective. It provides a structured and quantifiable way to evaluate the suitability of candidates for specific job positions based on the similarity of their CVs to job descriptions.

SNAPSHOTS

```python
#Task 2: Job Description Data Understanding

import datasets

# Load the job descriptions dataset from Hugging Face (replace with the actual dataset name)
dataset = datasets.load_dataset("jacob-hugging-face/job-descriptions")

# Extract 10-15 job descriptions
job_descriptions = dataset['train']['job_description'][:15]

# Print the job descriptions
for idx, description in enumerate(job_descriptions):
    print(f"Job Description {idx + 1}:\n{description}\n")
```

```
Job Description 1:
minimum qualifications
bachelors degree or equivalent practical experience years of experience in saas or productivity tools businessexperience man
aging enterprise accounts with sales cycles
preferred qualifications
 years of experience building strategic business partnerships with enterprise customersability to work through and with a re
seller ecosystem to scale the businessability to plan pitch and execute a territory business strategyability to build relati
onships and to deliver results in a crossfunctionalmatrixed environmentability to identify crosspromoting and uppromoting op
portunities within the existing account baseexcellent account management writtenverbal communication strategic and analytica
lthinking skills
about the job
as a member of the google cloud team you inspire leading companies schools and government agencies to work smarter with goog
le tools like google workspace search and chrome you advocate the innovative power of our products to make organizations mor
e productive collaborative and mobile your guiding light is doing whats right for the customer you will meet customers exact
ly where they are at and provide them the best solutions for innovation using your passion for google products you help spre
ad the magic of google to organizations around the world
```

Fig 1: Snapshot showing code snippet for second task along with the corresponding output

```python
from tabulate import tabulate

# Print top matches for each job description using tabulate
for job_desc, matches in top_matches.items():
    print("=" * 50)
    print(f"Job Description: {job_desc}\n")
    table_data = []

    for cv_details, similarity in matches:
        table_data.append([f"Similarity Score: {similarity}", f"CV Details: {cv_details}"])

    print(tabulate(table_data, headers=["", ""], tablefmt="fancy_grid"))
```

```
| Similarity Score: 0.8938312530517578 | CV Details: {'Skills': 'and uncommon creative mindset to bring a positive initiativ
e to your business operations and an upward curve to your company', 'Education': 'welcoming every opportunity to expand inte
llectual, cultural, and professional horizons Highly competitive team'}                                                      |

| Similarity Score: 0.882458508014679  | CV Details: {'Category': '. Possessing a strong bias for action and thriving as par
t of a team atmosphere I have developed a reputation for being results oriented and', 'Skills': 'beyond the MillerCoors team
to across the entire agency sharing strategy best practices, award', 'Education': 'series)'} |

| Similarity Score: 0.8815091252326965 | CV Details: {'Skills': 'to a dynamic sales organization and combine entrepreneurial
drive with business-management skills to propel gains in revenue,', 'Education': 'Bachelor of Science'}
```

Fig 2: Final presentation of top 5 profile based on each job description

RECOMMENDATIONS AND INSIGHTS

The project demonstrated the effectiveness of the PDF extraction tool, allowing for automated processing of resumes.

Matching candidates to job descriptions based on similarity scores can be a valuable tool for recruiters, as it streamlines the selection process.

Further refinements could include considering more advanced models and fine-tuning parameters to enhance matching accuracy.

CONCLUSION

In conclusion, this project achieved the objectives of extracting details from resumes, understanding job descriptions, and matching candidates to jobs based on skills and education. Despite some challenges, I successfully completed the tasks and demonstrated the potential of automation in the recruitment process.