RESUME EXTRACTOR WEB APP

TEAM 2 :-

1. **Rineesh M.S**
2. **Shadin P.K**
3. **Muhammed Zameel C**
4. **Muhammed Nihan P**
5. **Diljith**

AIM

To develop a web-based application that extracts and displays key information from resumes (PDF or DOCX) such as name, email, phone number, skills, education, and certifications using NLP techniques.

OBJECTIVES

- **Automatically extract candidate details from resumes.**
- **Use NLP to identify and categorize key fields (skills, education, etc.).**
- **Build a user-friendly Streamlit web interface.**
- **Support PDF and DOCX resume formats.**
- **Display structured output from unstructured text files.**

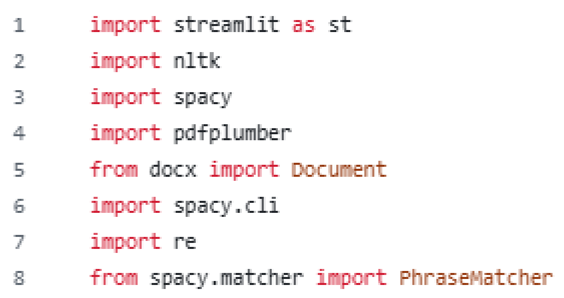
PROBLEM STATEMENT

Recruiters often have to manually sift through hundreds of resumes to extract key candidate information. This process is time-consuming and error-prone. A resume extraction system can streamline this process by automating data extraction, reducing workload, and improving accuracy and speed in shortlisting candidates.

STEP-BY-STEP IMPLEMENTATION

1. Import Libraries

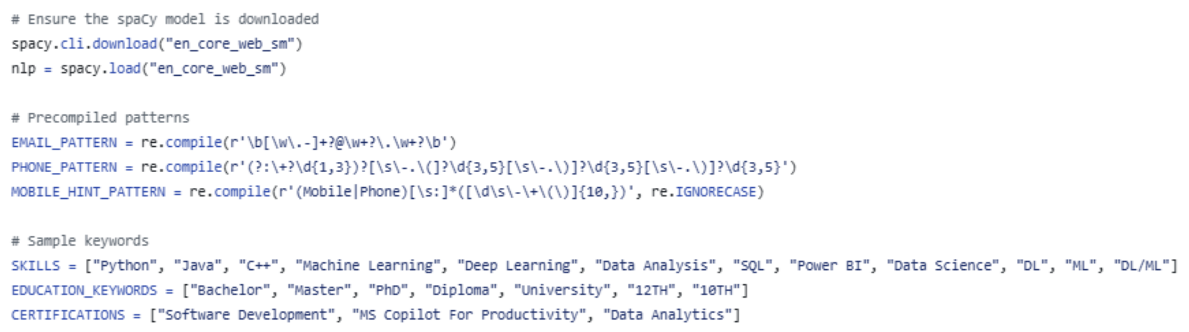
- **streamlit, pdfplumber, docx, re, spacy, nltk**



```python
import streamlit as st
import nltk
import spacy
import pdfplumber
from docx import Document
import spacy.cli
import re
from spacy.matcher import PhraseMatcher
```

2. File Upload

- **Allow users to upload resumes in .pdf or .docx format.**

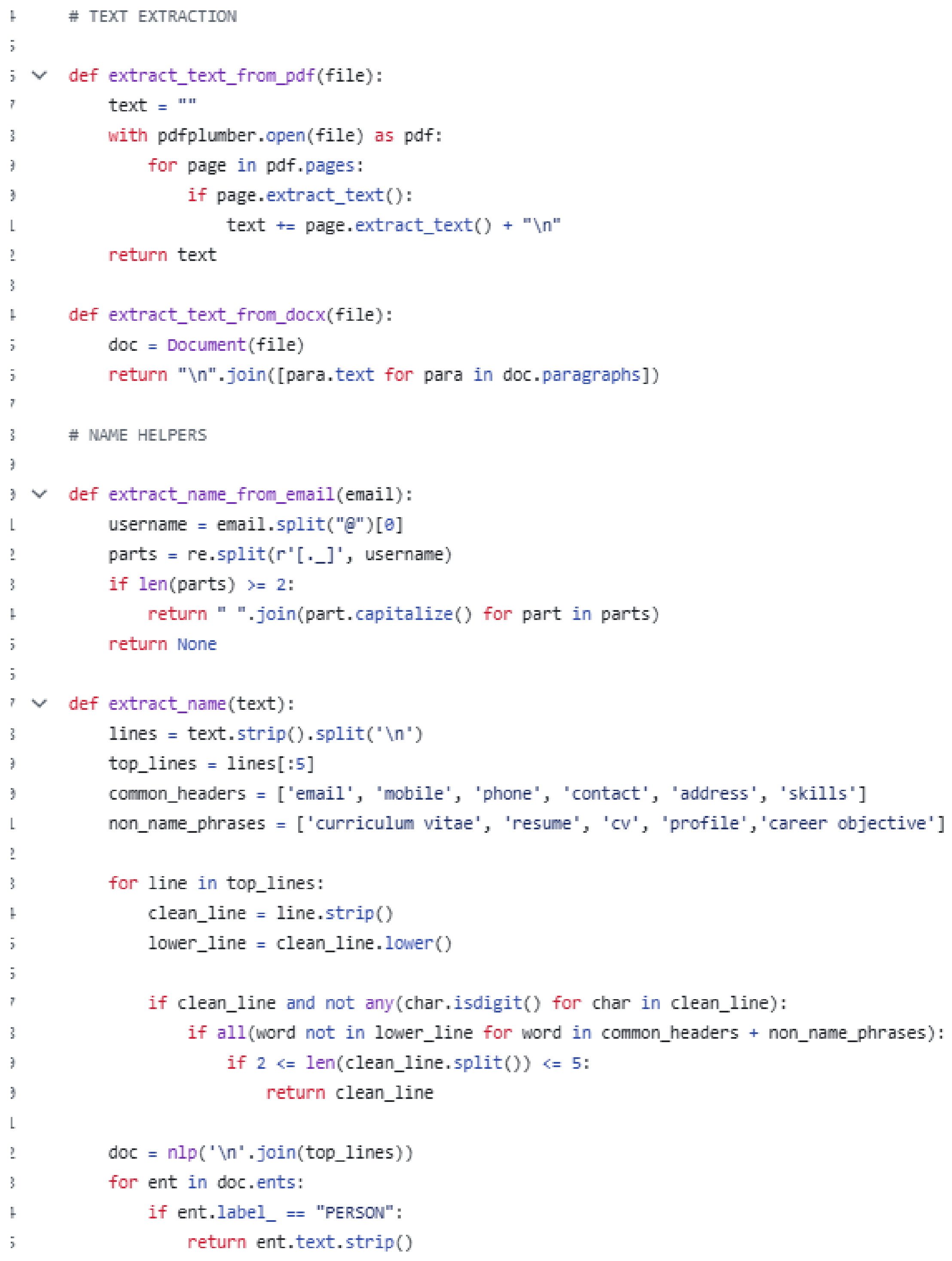


```python
# Ensure the spaCy model is downloaded
spacy.cli.download("en_core_web_sm")
nlp = spacy.load("en_core_web_sm")

# Precompiled patterns
EMAIL_PATTERN = re.compile(r'\b[\w\.-]+?@\w+?\.\w+?\b')
PHONE_PATTERN = re.compile(r'(?:\+?\d{1,3})?[\s\-.\(]?\d{3,5}[\s\-.\)]?\d{3,5}[\s\-.\)]?\d{3,5}')
MOBILE_HINT_PATTERN = re.compile(r'(Mobile|Phone)[\s:]*([\d\s\-\+\(\)]{10,})', re.IGNORECASE)

# Sample keywords
SKILLS = ["Python", "Java", "C++", "Machine Learning", "Deep Learning", "Data Analysis", "SQL", "Power BI", "Data Science", "DL", "ML", "DL/ML"]
EDUCATION_KEYWORDS = ["Bachelor", "Master", "PhD", "Diploma", "University", "12TH", "10TH"]
CERTIFICATIONS = ["Software Development", "MS Copilot For Productivity", "Data Analytics"]
```

3. Extract Text

- **For PDF: Use pdfplumber to extract page-wise text.**

- **For DOCX: Use python-docx to read paragraphs.**



```python
# TEXT EXTRACTION

def extract_text_from_pdf(file):
    text = ""
    with pdfplumber.open(file) as pdf:
        for page in pdf.pages:
            if page.extract_text():
                text += page.extract_text() + "\n"
    return text

def extract_text_from_docx(file):
    doc = Document(file)
    return "\n".join([para.text for para in doc.paragraphs])

# NAME HELPERS

def extract_name_from_email(email):
    username = email.split("@")[0]
    parts = re.split(r'[._]', username)
    if len(parts) >= 2:
        return " ".join(part.capitalize() for part in parts)
    return None

def extract_name(text):
    lines = text.strip().split('\n')
    top_lines = lines[:5]
    common_headers = ['email', 'mobile', 'phone', 'contact', 'address', 'skills']
    non_name_phrases = ['curriculum vitae', 'resume', 'cv', 'profile','career objective']

    for line in top_lines:
        clean_line = line.strip()
        lower_line = clean_line.lower()

        if clean_line and not any(char.isdigit() for char in clean_line):
            if all(word not in lower_line for word in common_headers + non_name_phrases):
                if 2 <= len(clean_line.split()) <= 5:
                    return clean_line

    doc = nlp('\n'.join(top_lines))
    for ent in doc.ents:
        if ent.label_ == "PERSON":
            return ent.text.strip()

    emails = EMAIL_PATTERN.findall(text)
    if emails:
        name_from_email = extract_name_from_email(emails[0])
        if name_from_email:
            return name_from_email

    return "Not found"
```

```
# EXTRACTION

def extract_entities(text):
    doc = nlp(text)

    entities = {
        "Name": extract_name(text),
        "Email": None,
        "Mobile Number": None,
        "Skills": set(),
        "Education": set(),
        "Certifications": set()
    }

    emails = EMAIL_PATTERN.findall(text)
    if emails:
        entities["Email"] = emails[0]

    mobile_match = MOBILE_HINT_PATTERN.search(text)
    if mobile_match:
        entities["Mobile Number"] = mobile_match.group(2).strip()
    else:
        phones = PHONE_PATTERN.findall(text)
        clean_phones = [p.strip() for p in phones if len(p.replace(" ", "").replace("-", "")) >= 10]
        if clean_phones:
            entities["Mobile Number"] = clean_phones[0]

    matcher = PhraseMatcher(nlp.vocab, attr="LOWER")
    matcher.add("SKILL", [nlp.make_doc(skill) for skill in SKILLS])
    matcher.add("EDUCATION", [nlp.make_doc(ed) for ed in EDUCATION_KEYWORDS])
    matcher.add("CERTIFICATION", [nlp.make_doc(cert) for cert in CERTIFICATIONS])

    matches = matcher(doc)
    for match_id, start, end in matches:
        label = nlp.vocab.strings[match_id]
        span_text = doc[start:end].text
        if label == "SKILL":
            entities["Skills"].add(span_text)
        elif label == "EDUCATION":
            entities["Education"].add(span_text)
        elif label == "CERTIFICATION":
            entities["Certifications"].add(span_text)

    for key in ["Skills", "Education", "Certifications"]:
        entities[key] = sorted(entities[key])

    return entities
```

4. Natural Language Processing

- **Load the spaCy language model (en_core_web_sm).**
- **Match phrases for skills, education, and certifications.**
- **Use regex to extract:**
  - **Email addresses**
  - **Phone numbers**
  - **Names from email IDs (if not directly found)**

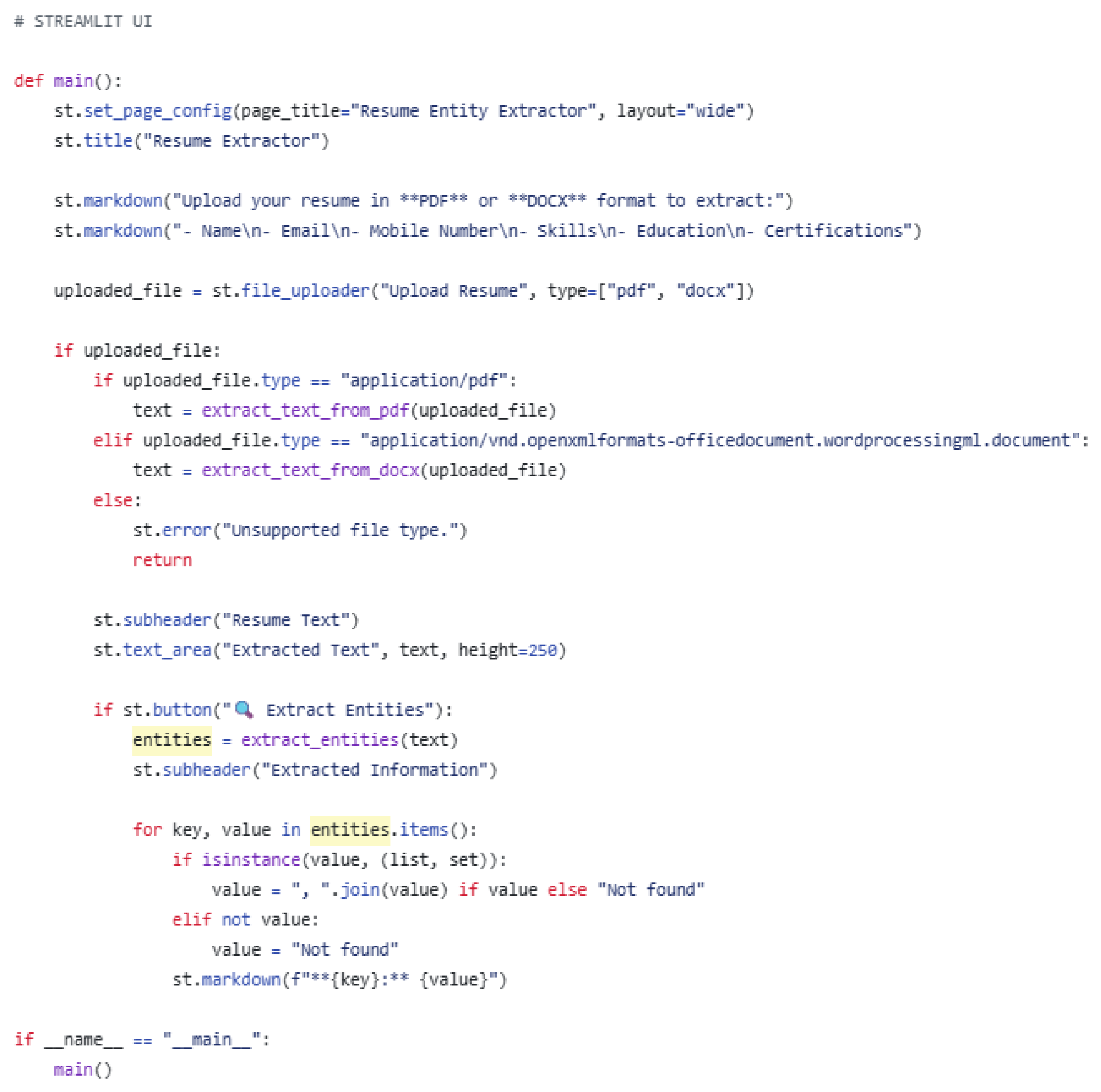
5. Display Extracted Results

- **Output key extracted fields such as:**
  - **Name**
  - **Email**
  - **Phone**
  - **Skills**
  - **Education**
  - **Certifications**

6. Streamlit App

- **Interactive UI with file uploader**
- **Buttons to trigger parsing**

- **Display results cleanly in the browser**



```python
# STREAMLIT UI

def main():
    st.set_page_config(page_title="Resume Entity Extractor", layout="wide")
    st.title("Resume Extractor")

    st.markdown("Upload your resume in **PDF** or **DOCX** format to extract:")
    st.markdown("- Name\n- Email\n- Mobile Number\n- Skills\n- Education\n- Certifications")

    uploaded_file = st.file_uploader("Upload Resume", type=["pdf", "docx"])

    if uploaded_file:
        if uploaded_file.type == "application/pdf":
            text = extract_text_from_pdf(uploaded_file)
        elif uploaded_file.type == "application/vnd.openxmlformats-officedocument.wordprocessingml.document":
            text = extract_text_from_docx(uploaded_file)
        else:
            st.error("Unsupported file type.")
            return

        st.subheader("Resume Text")
        st.text_area("Extracted Text", text, height=250)

        if st.button("🔍 Extract Entities"):
            entities = extract_entities(text)
            st.subheader("Extracted Information")

            for key, value in entities.items():
                if isinstance(value, (list, set)):
                    value = ", ".join(value) if value else "Not found"
                elif not value:
                    value = "Not found"
                st.markdown(f"**{key}:** {value}")

if __name__ == "__main__":
    main()
```

7. Dependencies (requirements.txt)

txt

CopyEdit

streamlit

nltk

spacy

pdfplumber

python-docx

CONCLUSION

This Resume Extractor tool simplifies the recruitment process by automating data extraction from resumes. Built with NLP techniques and deployed via Streamlit, it offers an efficient and interactive way to parse resumes in real time. Future improvements could include multilingual support, ranking candidates, and integrating with HR systems.