🧠 Experiment 10: Mini Project Based on NLP Applications

✨ Project Title: Resume Matcher – AI-Powered Resume Analysis Tool

This mini project demonstrates the use of **Natural Language Processing (NLP)** to analyze how well a resume matches a job description.

🔍 Key Objectives:

- **Extract and clean text** from resume PDF
- **Preprocess** both resume and job description using NLP techniques
- **Compare using cosine similarity** to compute a match score
- **Identify and visualize** top keywords
- **Provide improvement suggestions** based on missing keywords

This application provides candidates with insights to tailor their resumes for specific job roles more effectively.

🔧 Install & Import Required Libraries

```
# !pip install streamlit PyPDF2 plotly nltk
import streamlit as st
import PyPDF2
import re
import io
import plotly.graph_objects as go
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from collections import Counter
import math
```

📚 Download NLTK Resources

We use NLTK's **tokenizers**, **stopwords**, and **lemmatizer** for preprocessing.

```
def download_nltk_data():
    try:
        nltk.data.find('tokenizers/punkt')
        nltk.data.find('corpora/stopwords')
        nltk.data.find('corpora/wordnet')
    except LookupError:
        nltk.download('punkt')
        nltk.download('stopwords')
        nltk.download('wordnet')

download_nltk_data()
```

🧹 Text Preprocessing

We tokenize the text, remove stopwords, and apply lemmatization to normalize the content.

```
def process_text(text):
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()
    tokens = word_tokenize(text.lower())
    tokens = [lemmatizer.lemmatize(token) for token in tokens if token.isalnum()]
    tokens = [token for token in tokens if token not in stop_words]
    return tokens
```

📄 Extract Text from Resume PDF

```
def extract_text_from_pdf(pdf_file):
    try:
        with io.BytesIO(pdf_file.read()) as pdf_stream:
            pdf_reader = PyPDF2.PdfReader(pdf_stream)
            text = ""
            for page in pdf_reader.pages:
                page_text = page.extract_text()
                if page_text:
```

```
                text += " " + page_text
            text = re.sub(r'\s+', ' ', text)
            text = re.sub(r'[^\w\s@.,-]', '', text)
            return text.lower().strip()
    except Exception as e:
        return None
```

🤝 Compare Resume with Job Description

We compute cosine similarity between the two term frequency vectors and return the match percentage and keywords.

```
def compare_resume_with_job(resume_text, job_description):
    resume_tokens = process_text(resume_text)
    job_tokens = process_text(job_description)
    resume_freq = Counter(resume_tokens)
    job_freq = Counter(job_tokens)
    all_terms = set(resume_tokens + job_tokens)
    dot_product = sum(resume_freq[term] * job_freq[term] for term in all_terms)
    resume_magnitude = math.sqrt(sum(f * f for f in resume_freq.values()))
    job_magnitude = math.sqrt(sum(f * f for f in job_freq.values()))
    similarity = dot_product / (resume_magnitude * job_magnitude) if resume_magnitude and job_magnitude else 0
    return {
        'match_percentage': similarity * 100,
        'resume_keywords': resume_freq.most_common(10),
        'job_keywords': job_freq.most_common(10)
    }
```

📊 Visualize Keyword Match

```
def create_keyword_visualization(results):
    all_words = list(set([w for w, _ in results['resume_keywords']] + [w for w, _ in results['job_keywords']]))
    resume_dict = dict(results['resume_keywords'])
    job_dict = dict(results['job_keywords'])
    max_score = max([score for _, score in results['resume_keywords'] + results['job_keywords']] + [1])
    resume_scores = [resume_dict.get(word, 0)/max_score * 100 for word in all_words]
    job_scores = [job_dict.get(word, 0)/max_score * 100 for word in all_words]
    fig = go.Figure()
    fig.add_trace(go.Bar(x=resume_scores, y=all_words, name='Resume', orientation='h'))
    fig.add_trace(go.Bar(x=job_scores, y=all_words, name='Job Description', orientation='h'))
    fig.update_layout(title='Keyword Relevance Comparison', xaxis_title='Relevance Score (%)', yaxis_title='Keywords', barmode='group')
    return fig
```

🖥️ Streamlit App Interface

```
def main():
    st.title("📄 Resume Matcher - NLP Based Analysis")
    job_description = st.text_area("Paste Job Description here 👇", height=200)
    uploaded_file = st.file_uploader("Upload Resume (PDF only)", type=["pdf"])
    if uploaded_file and job_description:
        resume_text = extract_text_from_pdf(uploaded_file)
        if resume_text:
            results = compare_resume_with_job(resume_text, job_description)
            st.metric("Match Score (%)", f"{results['match_percentage']:.2f}")
            st.subheader("🔑 Top Keywords Comparison")
            st.plotly_chart(create_keyword_visualization(results), use_container_width=True)
            missing = set(w for w, _ in results['job_keywords']) - set(w for w, _ in results['resume_keywords'])
            if missing:
                st.warning("Consider adding these keywords: " + ', '.join(missing))
            else:
                st.success("Great! Your resume aligns well with the job description!")
        else:
            st.error("Failed to extract text from PDF.")
```

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

✅ Conclusion

This project applies key NLP techniques in a meaningful way to solve a common real-world problem in the job application process.

Skills Demonstrated:

- Text preprocessing (tokenization, lemmatization, stopword removal)
- Text vectorization and cosine similarity

- Data visualization using Plotly
- Streamlit web app interface

Improvements:

- Highlighting missing keywords directly in the resume text
- Adding support for DOCX or image-based resumes
- Ranking suggestions for resume rewriting

This project can be a useful tool for job seekers and recruiters alike.