# AI Resume Screening & Candidate Ranking System

This document contains the Python script for an AI-based Resume Screening and Candidate Ranking System using Streamlit.

## Python Code

import streamlit as st  
from PyPDF2 import PdfReader  
import pandas as pd  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.metrics.pairwise import cosine\_similarity  
import base64  
  
# Function to set background image  
def set\_background(image\_file):  
 with open(image\_file, "rb") as f:  
 encoded\_string = base64.b64encode(f.read()).decode()  
   
 css = f"""  
 <style>  
 .stApp {{  
 background-image: url("data:image/jpg;base64,{encoded\_string}");  
 background-size: cover;  
 background-position: center;  
 }}  
 </style>  
 """  
 st.markdown(css, unsafe\_allow\_html=True)  
  
# Set background image (Change path as needed)  
set\_background("pexels-goumbik-590041.jpg")  
  
# Custom CSS for styling  
st.markdown("""  
 <style>  
 div[data-testid="stTextArea"] textarea {  
 background-color: #8cb3c7 !important;  
 color: #000 !important;  
 border-radius: 10px;  
 border: 2px solid #333;  
 padding: 10px;  
 }  
 div[data-testid="stFileUploader"] {  
 background-color: #8cb3c7 !important;  
 border-radius: 10px;  
 padding: 10px;  
 }  
 </style>  
""", unsafe\_allow\_html=True)  
  
# Streamlit App Title  
st.title("AI Resume Screening & Candidate Ranking System")  
  
# Job Description Input  
st.header("Job Description")  
job\_description = st.text\_area("Enter the job description")  
  
# Resume Upload  
st.header("Upload Resumes")  
uploaded\_files = st.file\_uploader("Upload PDF files", type=["pdf"], accept\_multiple\_files=True)  
  
# Function to extract text from PDF  
def extract\_text\_from\_pdf(file):  
 pdf = PdfReader(file)  
 text = ""  
 for page in pdf.pages:  
 extracted\_text = page.extract\_text()  
 if extracted\_text:  
 text += extracted\_text  
 return text  
  
# Function to rank resumes based on job description  
def rank\_resumes(job\_description, resumes):  
 documents = [job\_description] + resumes  
 vectorizer = TfidfVectorizer().fit\_transform(documents)  
 vectors = vectorizer.toarray()  
   
 job\_description\_vector = vectors[0]  
 resume\_vectors = vectors[1:]  
  
 cosine\_similarities = cosine\_similarity(job\_description\_vector.reshape(1, -1), resume\_vectors).flatten()  
 return cosine\_similarities  
  
# Function to normalize scores  
def normalize\_score(similarity\_score):  
 return round(similarity\_score \* 100, 2) # Convert to percentage & round off  
  
# Processing uploaded resumes  
if uploaded\_files and job\_description.strip():  
 st.header("Ranking Resumes")  
   
 resumes = [extract\_text\_from\_pdf(file) for file in uploaded\_files]  
 scores = rank\_resumes(job\_description, resumes)  
  
 # Convert scores to percentage  
 normalized\_scores = [normalize\_score(score) for score in scores]  
  
 # Display results  
 results = pd.DataFrame({"Resume": [file.name for file in uploaded\_files], "Score (%)": normalized\_scores})  
 results = results.sort\_values(by="Score (%)", ascending=False)  
  
 st.write(results)