Code:

```
import streamlit as st
import pandas as pd
import re
import PyPDF2
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.multiclass import OneVsRestClassifier
from collections import Counter

# Function to extract text from PDF
def extract_text_from_pdf(pdf_file):
    pdf_reader = PyPDF2.PdfReader(pdf_file)
    text = "".join([page.extract_text() for page in pdf_reader.pages if page.extract_text()])
    return text

# Function to extract email from text
def extract_email(text):
    emails = re.findall(r'[\w\.-]+@[\w\.-]+', text)
    return emails[0] if emails else "N/A"

# Function to extract name (Assumption: First two words are name)
def extract_name(text):
    words = text.split()
    return " ".join(words[:2]) if words else "N/A"

# Function to clean resume text
def clean_resume(text):
    text = re.sub(r'http\S+', ' ', text)
    text = re.sub(r'[^a-zA-Z ]', ' ', text)
    text = re.sub(r'\s+', ' ', text).strip()
```

```
    return text

# Load pre-trained classification dataset
def load_training_data():
    df = pd.read_csv("UpdatedResumeDataSet.csv", encoding='utf-8')
    df['cleaned_resume'] = df['Resume'].apply(clean_resume)
    le = LabelEncoder()
    df['Category'] = le.fit_transform(df['Category'])
    vectorizer = TfidfVectorizer(sublinear_tf=True, stop_words='english')
    X = vectorizer.fit_transform(df['cleaned_resume'])
    y = df['Category']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
    model = OneVsRestClassifier(KNeighborsClassifier())
    model.fit(X_train, y_train)
    return model, vectorizer, le

# Streamlit UI
st.title("AI-powered Resume Screening & Ranking System")

uploaded_files = st.file_uploader("Upload Resume PDFs", accept_multiple_files=True, type=["pdf"])

if uploaded_files:
    job_description = st.text_area("Enter Job Description:")
    if st.button("Process Resumes"):
        model, vectorizer, label_encoder = load_training_data()
        tfidf_vectorizer = TfidfVectorizer()
        job_desc_vector = tfidf_vectorizer.fit_transform([job_description])

        results = []
        for pdf in uploaded_files:
            resume_text = extract_text_from_pdf(pdf)
            cleaned_text = clean_resume(resume_text)
            name = extract_name(cleaned_text)
            email = extract_email(cleaned_text)
```

```
    # Predict Job Category
    vectorized_input = vectorizer.transform([cleaned_text])
    predicted_category = label_encoder.inverse_transform(model.predict(vectorized_input))[0]

    # Calculate Similarity
    resume_vector = tfidf_vectorizer.transform([cleaned_text])
    similarity = cosine_similarity(job_desc_vector, resume_vector)[0][0]

    results.append((name, email, similarity, predicted_category))

# Rank resumes
results.sort(key=lambda x: x[2], reverse=True)

# Display results
df_results = pd.DataFrame(results, columns=["Name", "Email", "Similarity", "Predicted Category"])
df_results.insert(0, "Rank", range(1, len(df_results) + 1))
st.write(df_results)

# Save results to CSV
df_results.to_csv("ranked_resumes.csv", index=False)
st.success("Results saved as ranked_resumes.csv")
```