Smart CV Filter - Technical Report
Code Implementation & Libraries Reference

Mohammed Abdullah
3rd Stage | Network Department (Evening Studies)
Python & AI Project

1 Core Libraries & Dependencies

Library	Purpose
`streamlit`	Creates the interactive web-based dashboard and UI components.
`PyPDF2`	Extracts raw text content from uploaded PDF resume files.
`pandas`	Manages structured data (DataFrames) for sorting, filtering, and Excel export.
`scikit-learn`	Provides the AI engine: `TfidfVectorizer` for text-to-number conversion and `cosine_similarity` for matching.
`matplotlib`	Generates the visual bar charts for candidate performance comparison.
`io`	Handles in-memory file buffers for generating downloadable Excel reports.

Installation Command:

```
pip install streamlit PyPDF2 pandas scikit-learn matplotlib xlsxwriter
```

2 Project Structure

File Name	Description
`app.py`	The main application logic containing the UI, AI model, and filtering rules.
`make_cvs.py`	A utility script to generate synthetic dummy data (100 unique PDF resumes) for testing.
`fack_cvs/`	Directory containing the generated PDF resumes.
`Technical_Report.pdf`	This documentation file.

3 Application Configuration & UI Setup

The application uses Streamlit for a rapid, responsive frontend.

3.1 Setup Code

```python
import streamlit as st

# Sets the browser tab title and layout
st.set_page_config(page_title="Smart CV Filter", page_icon=" ", layout="wide
    ")

# Session State for Reset Logic
if 'reset_key' not in st.session_state:
    st.session_state.reset_key = 0

def reset_app():
    st.session_state.reset_key += 1
    st.rerun()
```

The `reset_app()` function is critical for clearing the session state, allowing users to start a fresh scan without refreshing the browser manually.

4 The AI Engine: TF-IDF & Cosine Similarity

The core intelligence of the system relies on Unsupervised Learning using Vector Space Models. This allows semantic comparison between the Job Description and Resume Text.

4.1 Implementation

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# 1. Combine Job Specs (Index 0) + All CVs (Index 1..N)
all_texts = [job_description] + df['Text'].tolist()

# 2. Vectorization: Convert text to numerical vectors
# stop_words='english' removes common words like 'the', 'and'
vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer.fit_transform(all_texts)

# 3. Calculate Similarity
# Compares the Job Vector (0) against all CV Vectors (1:)
matches = cosine_similarity(tfidf_matrix[0], tfidf_matrix[1:])

# 4. Save Base Score (0-100)
df['Base Score'] = [round(score * 100, 2) for score in matches[0]]
```

This logic ensures that candidates are ranked based on the *statistical significance* of the keywords they share with the job description, rather than just simple string matching.

5 Custom Weighted Scoring Logic

To enhance the AI's output with practical business logic, a rule-based layer modifies the base score.

5.1 Bonus & Penalty System

```python
skill_bonus = 0
matched_skills = []

# Iterates through required skills defined in Sidebar
for skill in required_skills:
    if skill.lower() in text.lower():
        skill_bonus += 10  # Reward: Boost score
        matched_skills.append(skill)
    else:
        skill_bonus -= 10  # Penalty: Reduce score

# Final Score Calculation
df['Final Score'] = df['Base Score'] + df['Bonus']
# Clip ensures score remains valid (0-100)
df['Final Score'] = df['Final Score'].clip(0, 100)
```

This algorithm (+10/-10) ensures that candidates who possess critical skills rise to the top, while those missing key requirements are penalized, creating a fair and balanced ranking.

6 Advanced Filtering: Max Priority & Block List

The system includes strict filters to handle absolute requirements or disqualifications.

6.1 Max Priority (God Mode)

This acts as a strict gatekeeper. If enabled, it bypasses the scoring system and deletes unqualified candidates.

```python
if max_priority and custom_skills_list:
    found_priority = False
    for cs in custom_skills_list:
        if cs.lower() in text.lower():
            found_priority = True
            break

    # If the critical skill is missing, SKIP this candidate completely
    if not found_priority:
        continue
```

6.2 Block List Logic

Allows instant removal of candidates matching negative keywords (e.g., ”Intern”).

```python
if all_blocked:
    for word in all_blocked:
        # If a bad word is found, filter out the row
        df = df[~df['Text'].str.contains(word, case=False)]
```

7 Data Visualization & Export

7.1 Bar Chart (Matplotlib)

Visualizes the relative scores of the top candidates.

```
fig, ax = plt.subplots(figsize=(6, 2))
# Black bars for high contrast theme
ax.barh(df['Name'], df['Final Score'], color='#000000')
ax.set_yticks([]) # Hides y-axis labels for cleaner look
st.pyplot(fig)
```

7.2 Excel Export

Generates a downloadable report for HR use.

```
buffer = io.BytesIO()
# Export only relevant columns
export_df = df[['Name', 'Final Score', 'Skills Found']]
with pd.ExcelWriter(buffer, engine='xlsxwriter') as writer:
    export_df.to_excel(writer, index=False, sheet_name='Ranked Candidates')
```

8 Algorithm Summary

Feature	**Algorithm/Tool**	**Functionality**
Text Analysis	TF-IDF Vectorizer	Converts resume text into numerical vectors based on word frequency.
Relevance Scoring	Cosine Similarity	Measures the angle between job description and resume vectors (0-100%).
Custom Logic	Weighted Rules	Adds +10 points for skills, -10 points for missing skills.
Strict Filtering	Boolean Logic	”Max Priority” and ”Block List” instantly include or exclude candidates.
Visualization	Matplotlib	Displays score distribution via horizontal bar charts.