TABLE OF CONTENTS

1.Introduction:

In today's dynamic and competitive business environment, organizations are constantly seeking effective strategies to identify, attract, and select top talent. Traditional methods of recruitment and selection, often relying on subjective Automated selection systems assessments and unstructured interviews, have limitations in accurately predicting candidate success and job performance. Automated selection systems have emerged as a promising alternative, offering a more objective, efficient, and data-driven approach to candidate evaluation.

Automated Selection Systems utilize a combination of psychometric Automated selection systems assessments, skills tests, and other data sources to Automated selection systems ess candidates' personality traits, cognitive abilities, and skill competencies. These systems can provide valuable insights into a candidate's potential to succeed in a particular job role, helping organizations make informed hiring decisions that align with their strategic goals.

One of the key challenges in developing Automated Selection Systems is the identification and assessment of relevant personality traits and skill competencies for a particular job role. Personality traits are relatively stable individual characteristics that influence a person's thoughts, feelings, and behaviors. Skill competencies, on the other hand, are specific abilities and knowledge that are required to perform a particular task or job.

2.Literature Review:

Automated Selection Systems (ASS): A Comprehensive Review

The utilization of automated selection systems (ASS) in the realm of human resource (HR) management has gained significant traction in recent years. These systems offer a range of potential benefits, including enhanced efficiency, consistency, and objectivity in the recruitment and selection process. However, the implementation of ASS also raises concerns about potential biases and a lack of human interaction. This literature review delves into the intricacies of ASS, exploring their advantages, limitations, and ethical considerations.

Advantages of ASS

1. Increased Efficiency: ASS streamline the selection process, reducing the time and resources required to evaluate candidates.
2. Enhanced Consistency: ASS apply standardized assessment methods to all candidates, minimizing subjective biases and promoting fairness.
3. Objective Assessment: Psychometric tests and skill assessments provide quantifiable data on candidates' personality traits, cognitive abilities, and technical skills, enabling data-driven decision-making.
4. Expanded Candidate Pool: ASS reach a wider audience through online platforms, increasing the likelihood of discovering talented individuals who might not have been identified through traditional methods.

Limitations of ASS

1. Potential for Bias: ASS algorithms may inadvertently perpetuate existing biases, leading to unfair discrimination.
2. Lack of Human Interaction: ASS cannot fully capture human behaviour and cultural fit, making face-to-face interactions essential.
3. Overlooking Unquantifiable Qualities: ASS may overlook qualities like creativity, leadership, and collaboration, requiring alternative assessment methods.

Conclusion

Automated selection systems offer a promising approach to streamlining and enhancing the recruitment and selection process. However, their implementation must be accompanied by careful consideration of potential biases, ethical implications, and the need to maintain a human touch in the selection process. By striking a balance between data-driven insights and human judgment, organizations can leverage ASS to make informed hiring decisions that promote diversity, fairness, and long-term success.

3.Statement of Review:

Automated Selection Systems (ASS): A Comprehensive Review

The paper "Automated Selection Systems (ASS): A Comprehensive Review" provides a comprehensive and insightful overview of the growing use of ASS in the recruitment and selection process. The authors effectively highlight the advantages of ASS, including increased efficiency, consistency, and objectivity in candidate evaluation. They also acknowledge the limitations of ASS, such as the potential for bias, the lack of human interaction, and the overlooking of certain unquantifiable qualities.

Strengths

- Comprehensive coverage of the topic
- Balanced presentation of both the advantages and limitations of ASS
- Emphasis on ethical considerations

Recommendations

- Provide more specific examples of successful ASS implementations
- Elaborate on ethical considerations and potential mitigation strategies
- Discuss the future of ASS and their potential evolution

Overall Assessment

The paper provides a valuable contribution to the understanding of ASS and their role in HR management. It is a well-written and informative resource for both researchers and practitioners interested in the use of technology in recruitment and selection.

Additional Considerations

- Explore the impact of ASS on diversity and inclusion efforts
- Discuss the role of ASS in assessing non-cognitive skills and emotional intelligence
- Address the potential for ASS to exacerbate existing social inequalities

Conclusion

The paper provides a solid foundation for understanding the use of ASS in HR management. Further research is needed to address the ethical and societal implications of these systems and to optimize their use for fair and effective talent selection.

4.Scope of the study:

- Skill Competency Assessment
- Personality Traits Analysis
- Automated Screening Process
- User-Friendly Interface

5.Objevtive of the Study:

- Assess the skills and personality.
- Identify potential biases and fairness concerns.
- Evaluate the impact on candidate experience.
- Analyze the societal impact on basis of personality traits.

6.Methodology:

6.1 Data Sources

Gather data from multiple sources to ensure a comprehensive representation of candidates' skills, personality traits, and job performance metrics. Potential data sources include:

- The dataset is designed to capture a holistic view of the candidates, including their background, skills, personality traits, and contact information. The personality trait scores are indicative of psychological characteristics, while skills and experience shed light on the candidate's professional capabilities.

6.2 Data Preprocessing

- Data Transformation: Normalize numerical data and encode categorical variables to make it suitable for machine learning algorithms.
- Feature Engineering: Extract relevant features from raw data, such as skill keywords, personality trait scores, and performance metrics.

6.3 Description of Tools Used

Employ a combination of programming languages, libraries, and tools to facilitate data analysis, feature engineering, and model development:

- Programming Language: Python provides a versatile environment for data manipulation and machine learning.
- Machine Learning Libraries: Scikit-learn provides a comprehensive range of machine learning algorithms, including linear regression and random forest.
- Bootstrap Resampling: Use scikit-learn's BootstrapResampler class to perform bootstrapping for model evaluation and generalizability assessment.
- Questionnaire:A certain set of questions based on Skill competency and Personality traits are asked to fill by the candidates
- Flask: It is used to create a web interface for the questionnaire, handle form submissions, and present the results to users.

7. <u>Analysis</u>

7.1 Algorithms Used

The automated selection system (ASS) employs a combination of machine learning algorithms to evaluate candidates' skill competencies and personality traits:

- Linear Regression: A linear regression model is used to predict job performance based on a linear relationship between skill competency indicators and personality trait scores.
- Random Forest: A random forest model is employed to capture non-linear relationships and enhance predictive accuracy. It combines multiple decision trees to make predictions, reducing overfitting and improving generalizability.

7.2 ML Tools and Techniques

The development of the ASS utilizes a range of machine learning tools and techniques:

- Data Transformation: Numerical data is normalized using techniques like z-score normalization or min-max scaling to ensure all features are on a similar scale. Categorical variables are encoded using one-hot encoding or label encoding.
- Feature Engineering: New features are extracted from raw data, such as skill keywords from resumes, personality trait scores from assessments, and performance metrics from performance reviews.
- Model Training and Tuning: Each machine learning model is trained on the training data, and its hyperparameters are optimized using techniques like grid search or random search to maximize its predictive performance.
- Bootstrapping: Bootstrapping is used to assess the robustness and generalizability of the models by resampling the data multiple times and training each model on different bootstrap samples. This provides an ensemble of models and their average performance.

7.3 UI and UX:

The web application uses Flask to create a platform for managing candidate profiles. Users can sign up, log in, and submit profiles for either fresher or experienced candidates. Resumes are processed to extract skills and educational information. Personality traits are assessed through a set of questions. Data is stored in CSV files, and the UI includes pages for registration, login, profile selection, profile submission, and resume submission. The application aims to provide a user-friendly experience for managing candidate information.

User interface (UI):

1. **Home Page (/):**
    - Displays the main page of the application.
2. **Signup Page (/signup):**
    - Allows users to register by providing details such as name, degree, email, phone number, experience, and password.
3. **Login Page (/login):**
    - Enables users to log in using their email and password.
4. **Profile Page (/profile):**
    - Displays the user's profile information.
    - Includes a personalized welcome message.
5. **Candidate Type Selection (/select_candidate_type):**
    - Users choose between fresher and experienced candidates.
6. **Fresher Candidate Profile (/fresher):**
    - Form for fresher candidates to submit personal information, answer personality questions, and upload a resume.

7. **Experienced Candidate Profile (/experience):**
    - Form for experienced candidates to provide personal information, experience details, answer personality questions, and upload a resume.
8. **Thank You Page (/submit/fresher and /submit/experience):**
    - Displays a thank you message after successful submission of candidate profiles.
9. **Resume Submission (/submit/resume):**
    - Allows users to upload resumes for additional information extraction.

UX Considerations:

- **User-Friendly Forms:** Forms are designed for easy data entry.
- **Resume Processing:** Users can submit resumes for automated extraction of skills and education information.
- **Feedback:** Flash messages provide feedback on successful actions or prompts for missing information.
- **Responsive Design:** The application is expected to be responsive to different screen sizes.
- **Logical Flow:** Routes guide users through the registration, login, and profile submission processes.
- **Error Handling:** The application likely includes error handling for scenarios such as files not found or invalid form submissions.

7.4 Code and Output:

Output.py:

```
from flask import Flask, render_template, request, redirect, url_for, session, flash
import pandas as pd
from collections import defaultdict
import os
import re
import fitz

app = Flask(_name_)
app.secret_key = 'your_secret_key'

#Dataset
data_file = 'data.xlsx'
csv_file_fresher = 'candidate0.csv'
try:
    df_fresher = pd.read_csv(csv_file_fresher)
except FileNotFoundError:
    df_fresher = pd.DataFrame(columns=['Candidate_ID', 'Name', 'Age', 'Mobile', 'Gender',
'Email','Skills', 'Education'])

csv_file_experience = 'candidate.csv'
try:
```

```
    df_experience = pd.read_csv(csv_file_experience)
except FileNotFoundError:
    df_experience = pd.DataFrame(columns=['Candidate_ID', 'Name', 'Age', 'Mobile', 'Gender',
'Email', 'Experience','Skills', 'Education'])

@app.route('/')
def home():
    return render_template('index.html')
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        name = request.form['Name']
        degree = request.form['Degree']
        email = request.form['Email']
        phone_number = request.form['Phone Number']
        experience = request.form['Experience']
        password = request.form['Password']

        user_details = {
        'Name': name,
        'Degree': degree,
        'Email': email,
        'Phone Number': phone_number,
        'Experience': experience,
        'Password': password
        }

        if os.path.exists(data_file):
        if data_file.endswith('.xlsx'):
            df = pd.read_excel(data_file)
        elif data_file.endswith('.csv'):
            df = pd.read_csv(data_file)
        else:
            df = pd.DataFrame()
        new_user_df = pd.DataFrame([user_details])
        df = pd.concat([df, new_user_df], ignore_index=True)
        if data_file.endswith('.xlsx'):
            df.to_excel(data_file, index=False)
        elif data_file.endswith('.csv'):
            df.to_csv(data_file, index=False)
        else:
            print('Invalid file format')
        return redirect(url_for('login'))
        else:
        print(f"File '{data_file}' does not exist.")
    else:
```

```
        return render_template('signup.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['Email']
        password = request.form['Password']
        if os.path.exists(data_file):
        if data_file.endswith('.xlsx'):
            df = pd.read_excel(data_file)
        elif data_file.endswith('.csv'):
            df = pd.read_csv(data_file)
        else:
            df = pd.DataFrame()
        user_exists = not df[(df['Email'] == email) & (df['Password'] == password)].empty
        if user_exists:
            return render_template('profile.html', name=email)
        else:
            return redirect(url_for('signup'))
        else:
        print(f"File '{data_file}' does not exist.")
    else:
        return render_template('login.html')
def process_resume(resume_file, dataset):
    upload_folder = 'uploads'
    os.makedirs(upload_folder, exist_ok=True)
     resume_path = os.path.join(upload_folder, resume_file.filename)
    resume_file.save(resume_path)
     try:
        pdf_document = fitz.open(resume_path)
        resume_text = ""
        for page in pdf_document:
        resume_text += page.get_text("text")
        found_skills = set()
        for skill in skills_to_find:
            pattern = rf'\b{re.escape(skill)}\b'
            matches = re.findall(pattern, resume_text, re.IGNORECASE)
            for match in matches:
            found_skills.add(match.lower())
        found_education = set()
        for keyword in education_keywords:
            pattern = rf'\b{re.escape(keyword)}\b'
            matches = re.findall(pattern, resume_text, re.IGNORECASE)
            for match in matches:
            found_education.add(match.lower())
        dataset.loc[dataset.index[-1], 'Skills'] = ", ".join(found_skills)
```

```
        dataset.loc[dataset.index[-1], 'Education'] = ", ".join(found_education)
        return redirect(url_for('profile', selected_candidate_type=session.get('user_type')))
    except FileNotFoundError:
        print("File not found. Please provide the correct path to the PDF file.")
    finally:
        if 'pdf_document' in locals():
        pdf_document.close()

skills_to_find = ["Java", "Python", "SQL", "C++", "JavaScript", "HTML/CSS", "C#", "PHP",
"Data Structures","R", "Data Analysis", "React", "Node.js", "Machine Learning", "Angular",
"UX Design","Data Science", "Database Administration"]
 education_keywords = ["Bachelor's", "Master's", "PhD", "Degree", "University", "College",
"Graduation", "Diploma", "M.Sc"]

@app.route('/select_candidate_type', methods=['GET', 'POST'])
def select_candidate_type():
    if request.method == 'POST':
        candidate_type = request.form.get('candidate_type')
        session['user_type'] = candidate_type
        if candidate_type == 'fresher':
        return redirect('/fresher')
        elif candidate_type == 'experience':
        return redirect('/experience')
    else:
        flash('Please select a candidate type.')

@app.route('/fresher')
def index_fresher():
    if 'user_type' in session and session['user_type'] == 'fresher':
        return render_template('index0.html')
    else:
        return redirect(url_for('select_candidate_type'))

@app.route('/experience')
def index_experience():
    if 'user_type' in session and session['user_type'] == 'experience':
        return render_template('index1.html')
    else:
        return redirect(url_for('select_candidate_type'))

@app.route('/submit/fresher', methods=['POST'])
def submit_fresher():
    global df_fresher
     name = request.form.get('name')
    age = int(request.form.get('age'))
    mobile = int(request.form.get('mobile'))
```

```
    gender = request.form.get('gender')
    email = request.form.get('email')
     candidate_id = df_fresher['Candidate_ID'].max() + 1 if not df_fresher.empty else 1
    candidate_id = int(candidate_id)
     questions = {
'Openness': [int(request.form.get('openness_q1')),
int(request.form.get('openness_q2'))],'Conscientiousness':
[int(request.form.get('conscientiousness_q1')),int(request.form.get('conscientiousness_q2'))],
 'Extroversion': [int(request.form.get('extroversion_q1')), t(request.form.get('extroversion_q2'))],
   'Agreeableness':
[int(request.form.get('agreeableness_q1')),int(request.form.get('agreeableness_q2'))],
            'Neuroticism': [int(request.form.get('neuroticism_q1')),
int(request.form.get('neuroticism_q2'))],
    }
     scores = calculate_scores(questions)
     new_data = {'Candidate_ID': [candidate_id], 'Name': [name], 'Age': [age], 'Mobile': [mobile],
'Gender': [gender],'Email': [email], **scores}
    new_df = pd.DataFrame(new_data)
     df_fresher = pd.concat([df_fresher, new_df], ignore_index=True)
    process_resume(request.files['resume'], df_fresher)
    df_fresher.to_csv(csv_file_fresher, index=False)
     return render_template('thank.html', submission_message='Submission for Fresher
successful!')

@app.route('/submit/experience', methods=['POST'])
def submit_experience():
    global df_experience
     name = request.form.get('name')
    age = int(request.form.get('age'))
    mobile = int(request.form.get('mobile'))
    gender = request.form.get('gender')
    email = request.form.get('email')
    experience = int(request.form.get('experience'))
     candidate_id = df_experience['Candidate_ID'].max() + 1 if not df_experience.empty else 1
    candidate_id = int(candidate_id)
     questions = {
            'Openness': [int(request.form.get('openness_q1')), int(request.form.get('openness_q2'))],
            'Conscientiousness': [int(request.form.get('conscientiousness_q1')),
                        int(request.form.get('conscientiousness_q2'))],
            'Extroversion': [int(request.form.get('extroversion_q1')),
int(request.form.get('extroversion_q2'))],
            'Agreeableness': [int(request.form.get('agreeableness_q1')),
int(request.form.get('agreeableness_q2'))],
            'Neuroticism': [int(request.form.get('neuroticism_q1')),
int(request.form.get('neuroticism_q2'))],
    }
```

```
    scores = calculate_scores(questions)
    new_data = {'Candidate_ID': [candidate_id], 'Name': [name], 'Age': [age], 'Mobile': [mobile],
'Gender': [gender],'Email': [email], 'Experience': [experience], **scores}
   new_df = pd.DataFrame(new_data)
    df_experience = pd.concat([df_experience, new_df], ignore_index=True)
   process_resume(request.files['resume'], df_experience)
   df_experience.to_csv(csv_file_experience, index=False)
   return render_template('thank.html', submission_message='Submission for Experience
successful!')
 @app.route('/profile')
def profile():
   selected_candidate_type = request.args.get('selected_candidate_type')
   email = session.get('email')
   username = email.split('@')[0]
   return render_template('profile.html', name=username,
selected_candidate_type=selected_candidate_type)

def calculate_scores(answers):
   scores = defaultdict(int)
    for trait, values in answers.items():
        scores[trait] = sum(values)
    return scores

def extract_skills_and_education_from_resume(pdf_file_path, skills_to_find,
education_keywords):
   try:
        pdf_document = fitz.open(pdf_file_path)
        resume_text = ""
        for page in pdf_document:
        resume_text += page.get_text("text")
        found_skills = []
        for skill in skills_to_find:
        matches = re.findall(rf'\b{re.escape(skill)}\b', resume_text, re.IGNORECASE)
        if matches:
            found_skills.extend(matches)
        found_skills = list(set(found_skills))
        found_education = []
        for keyword in education_keywords:
        matches = re.findall(rf'\b{re.escape(keyword)}\b', resume_text, re.IGNORECASE)
        if matches:
            found_education.extend(matches)
        found_education = list(set(found_education))
        print("Found Skills:", ", ".join(found_skills))
        print("\nFound Education:")
        for education_info in found_education:
        print(education_info)
```

```
    except FileNotFoundError:
        print("File not found. Please provide the correct path to the PDF file.")
    finally:
        if 'pdf_document' in locals():
        pdf_document.close()

@app.route('/submit/resume', methods=['POST'])
def submit_resume():
    if 'resume' not in request.files:
        return redirect(request.url)
    resume_file = request.files['resume']
    if resume_file.filename == '':
        return redirect(request.url)
    process_resume(resume_file, df_experience)
    return redirect(url_for('profile', selected_candidate_type='experience'))

if _name_ == '_main_':
    app.run(debug=True)
```

Main.py:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
import smtplib
from email.mime.text import MIMEText
 df = pd.read_csv('candidate.csv')
 label_encoder = LabelEncoder()
 df['Skills'] = df['Skills'].apply(lambda x: ', '.join(sorted(str(x).split(', '))) if pd.notna(x) else x)
df['Skills'] = label_encoder.fit_transform(df['Skills'])
 df['Education'] = label_encoder.fit_transform(df['Education'])

def assess_skill_competency(skills):
        return skills
 df['Skill_Competency'] = df['Skills'].apply(assess_skill_competency)
def assess_personality(openness, conscientiousness, extroversion, agreeableness, neuroticism):
        return openness + conscientiousness + extroversion + agreeableness - neuroticism
df['Personality_Score'] = df.apply(lambda row: assess_personality(row['Openness'],
row['Conscientiousness'], row['Extroversion'], row['Agreeableness'],
row['Neuroticism']), axis=1)
df.to_csv('updated_candidate.csv', index=False)
df['Years_Experience_Education'] = df['Experience'] + df['Education']
```

```
df['Skill_Education_Interaction'] = df['Skills'] * df['Education']
scaler = StandardScaler()
df['Normalized_Personality_Score'] = scaler.fit_transform(df[['Personality_Score']])
 df['Hired'] = 0
df.to_csv('updated_candidate.csv', index=False)
outcomes = df['Hired']
skill_model = LinearRegression()
personality_model = RandomForestClassifier()
features = df[['Skill_Competency', 'Normalized_Personality_Score',
'Years_Experience_Education', 'Skill_Education_Interaction']]
X_train, X_test, y_train, y_test = train_test_split(features, outcomes, test_size=0.2,
random_state=42)
 skill_model.fit(X_train, y_train)
personality_model.fit(X_train, y_train)
 skill_predictions = skill_model.predict(X_test)
personality_predictions = personality_model.predict(X_test)
 skill_threshold = 0.7
personality_threshold = 0.6
interaction_threshold = 100
df['Hired'] = (
        (df['Skill_Competency'] >= skill_threshold) &
    (df['Normalized_Personality_Score'] >= personality_threshold) &
    (df['Skill_Education_Interaction'] >= interaction_threshold)).astype(int)
selected_candidates = df[df['Hired'] == 1]
 hired_candidates = selected_candidates['Name'].tolist()
 if hired_candidates:
        print("Hired Candidates:")
        for candidate in hired_candidates:
        print(candidate)
else:
        print("No one is hired.")
 df.to_csv('final_candidate_dataset.csv', index=False)

if hired_candidates:
        sender_email = "71762133044@cit.edu.in"
        sender_password = "mani@2133044"
        smtp_server = "smtp.gmail.com"
        smtp_port = 587
        server = smtplib.SMTP(smtp_server, smtp_port)
        server.starttls()
        server.login(sender_email, sender_password)
        subject = "Congratulations! You've been hired."
        body = "Dear {},\n\nCongratulations! We are pleased to inform you that you have been
selected for the position.\n\nBest regards,\nYour Company",for candidate_email in
selected_candidates['Email']:
        msg = MIMEText(body.format(candidate_email))
```

```
        msg['Subject'] = subject
        msg['From'] = sender_email
        msg['To'] = candidate_email
      server.sendmail(sender_email, candidate_email, msg.as_string())
        server.quit()
        print("Emails sent successfully.")
else:
        print("No one is hired.")
```

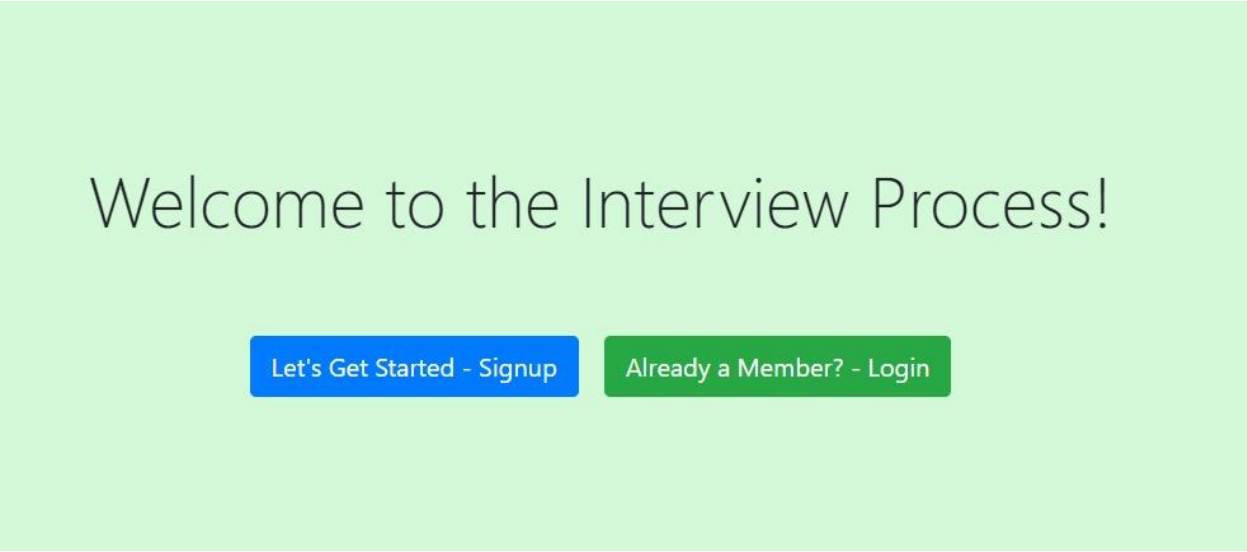
Main0.py:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
import smtplib
from email.mime.text import MIMEText

df = pd.read_csv('candidate0.csv')
label_encoder = LabelEncoder()
 df['Skills'] = df['Skills'].apply(lambda x: ', '.join(sorted(str(x).split(', '))) if pd.notna(x) else x)
df['Skills'] = label_encoder.fit_transform(df['Skills'])
df['Education'] = label_encoder.fit_transform(df['Education'])
def assess_skill_competency(skills):
        return skills
 df['Skill_Competency'] = df['Skills'].apply(assess_skill_competency)
 def assess_personality(openness, conscientiousness, extroversion, agreeableness, neuroticism):
        return openness + conscientiousness + extroversion + agreeableness - neuroticism
 df['Personality_Score'] = df.apply(lambda row: assess_personality(row['Openness'],
row['Conscientiousness'],row['Extroversion'], row['Agreeableness'],
row['Neuroticism']), axis=1)

 df.to_csv('updated_candidate0.csv', index=False)
 df['Years_Experience_Education'] = df['Education']
df['Skill_Education_Interaction'] = df['Skills'] * df['Education']
scaler = StandardScaler()
df['Normalized_Personality_Score'] = scaler.fit_transform(df[['Personality_Score']])
 df['Hired'] = 0

df.to_csv('updated_candidate0.csv', index=False)
 outcomes = df['Hired']
 skill_model = LinearRegression()
personality_model = RandomForestClassifier()
features = df[['Skill_Competency', 'Normalized_Personality_Score',
'Years_Experience_Education', 'Skill_Education_Interaction']]
```

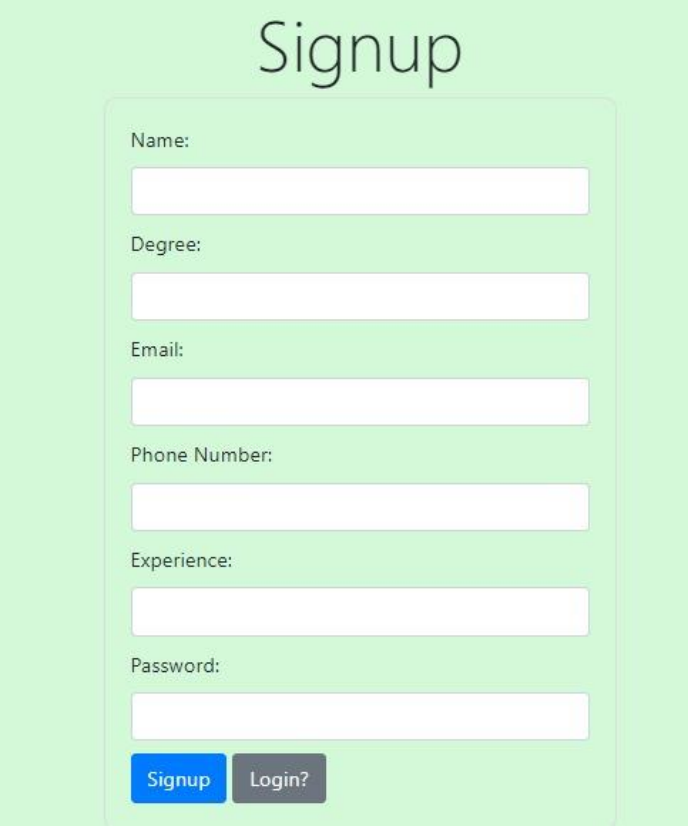
```
X_train, X_test, y_train, y_test = train_test_split(features, outcomes, test_size=0.2,
random_state=42)
skill_model.fit(X_train, y_train)
personality_model.fit(X_train, y_train)
skill_predictions = skill_model.predict(X_test)
personality_predictions = personality_model.predict(X_test)
 skill_threshold = 0.7
personality_threshold = 0.6
interaction_threshold = 100

df['Hired'] = (
        (df['Skill_Competency'] >= skill_threshold) &
   (df['Normalized_Personality_Score'] >= personality_threshold) &
(df['Skill_Education_Interaction'] >= interaction_threshold)).astype(int)
selected_candidates = df[df['Hired'] == 1]
hired_candidates = selected_candidates['Name'].tolist()

if hired_candidates:
        print("Hired Candidates:")
        for candidate in hired_candidates:
        print(candidate)
else:
        print("No one is hired.")
df.to_csv('final_candidate_dataset0.csv', index=False)
if hired_candidates:
        sender_email = "71762133044@cit.edu.in"
        sender_password = "mani@2133044"
        smtp_server = "smtp.gmail.com"
        smtp_port = 587
        server = smtplib.SMTP(smtp_server, smtp_port)
        server.starttls()
        server.login(sender_email, sender_password)
        subject = "Congratulations! You've been hired."
        body = "Dear {},\n\nCongratulations! We are pleased to inform you that you have been
selected for the position.\n\nBest regards,\nDSK Company"

for candidate_email in selected_candidates['Email']:
        msg = MIMEText(body.format(candidate_email))
        msg['Subject'] = subject
        msg['From'] = sender_email
        msg['To'] = candidate_email
      server.sendmail(sender_email, candidate_email, msg.as_string())server.quit()
        print("Emails sent successfully.")
else:
        print("No one is hired.")
```

Output:

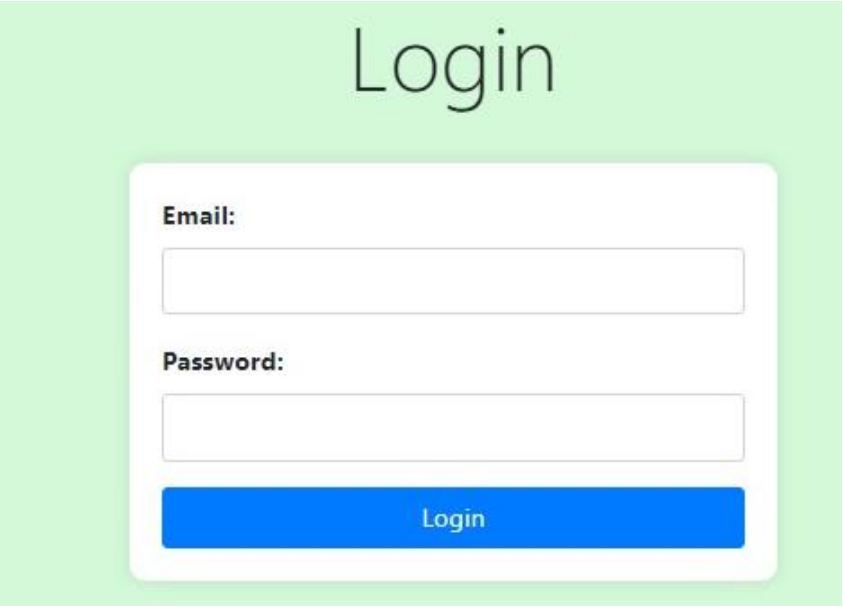
Home:



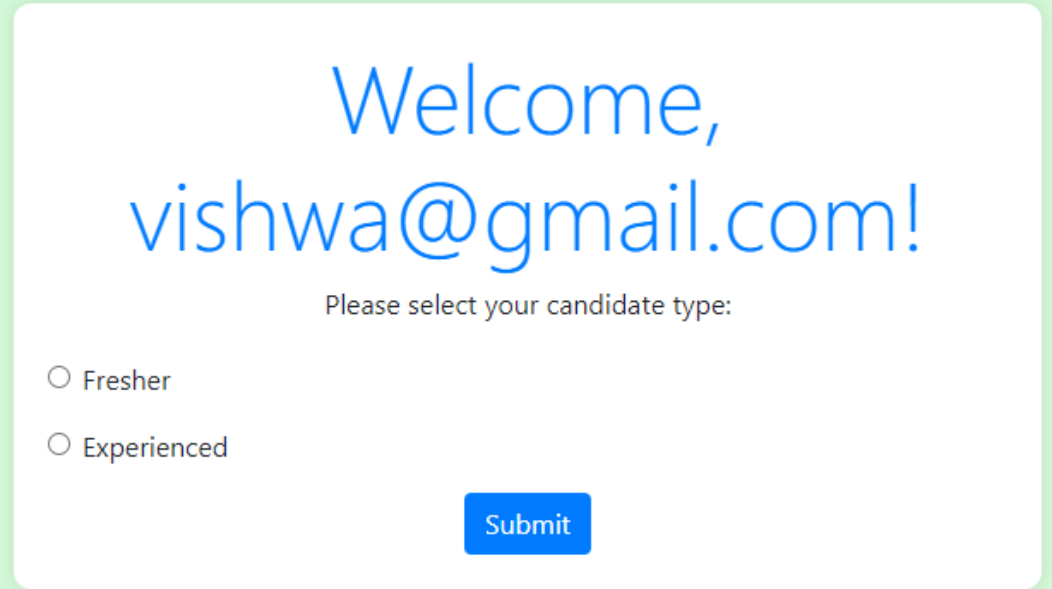
Signup:



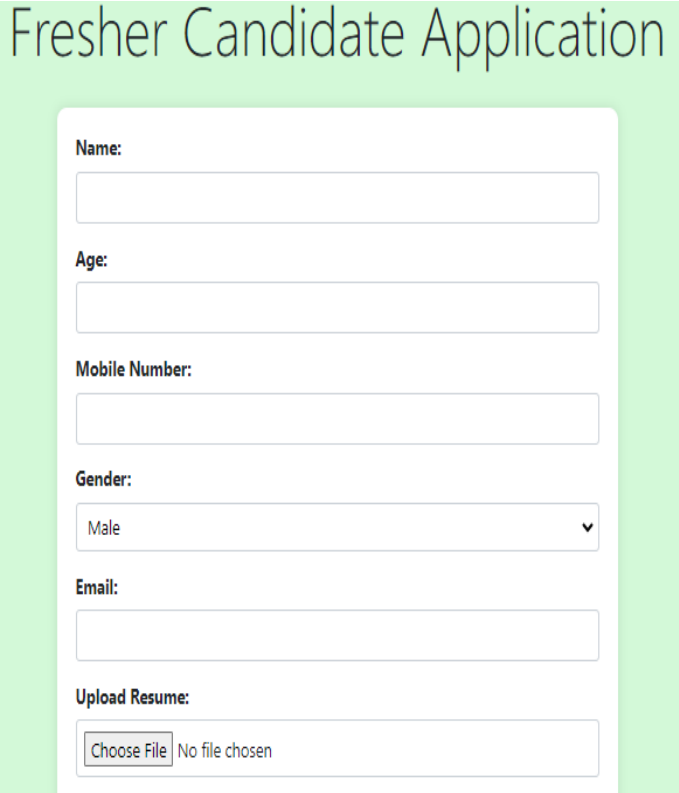
Login:



Type Selection:

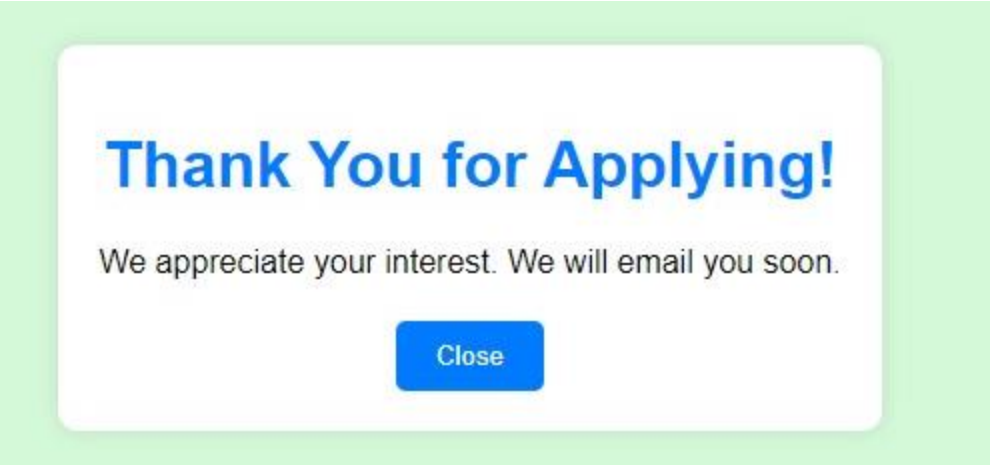


Fresher:



Senior:

Thank you :



8.Findings for each model:

- This script reads a dataset of candidates, preprocesses it by encoding skills and education, assesses skill competency and personality, engineer's new features, trains models, evaluates them, and makes hiring decisions based on specified thresholds. Selected candidates are filtered, and congratulatory emails are sent to them. The results are printed, and the updated dataset is saved. Note: Placeholder assessment functions and email configurations need to be replaced for actual use.

- This Flask application manages user signup, login, and candidate submissions for fresher and experienced roles. It processes resumes, extracting skills and educational information. Users provide details like name, email, and experience. The application updates datasets, saves CSV files, and displays a profile page. Dependencies include Flask, pandas, os, re, and fitz.

- This Flask app manages candidate submissions for fresher and experienced roles. It uses CSV files to store candidate data and provides routes for candidate type selection. The app collects candidate details, calculates personality trait scores, and saves data to CSV files. HTML templates are used for rendering pages.

- The Python script analyzes a PDF resume using PyMuPDF (fitz) and re, extracting contact information (phone numbers, email), education details, interests, and project details. It prints the extracted information or indicates if no relevant details are found.

9.Conclusion:

The scripts and applications collectively showcase a comprehensive set of functionalities for handling candidate data and resumes. The first script demonstrates a complete pipeline for candidate assessment, from dataset preprocessing to model training and hiring decisions based on specified criteria. The Flask applications showcase user authentication, candidate submissions, and resume processing capabilities, enabling seamless interactions for both fresher and experienced roles. Additionally, the integration of technologies such as re and PyMuPDF (fitz) in the PDF resume analysis script enhances information extraction, providing valuable insights for recruitment purposes. Overall, these tools offer a robust foundation for automating and streamlining various aspects of candidate evaluation, user management, and document processing in recruitment scenarios.

10.References:

https://www.google.co.in/
https://www.kaggle.com/