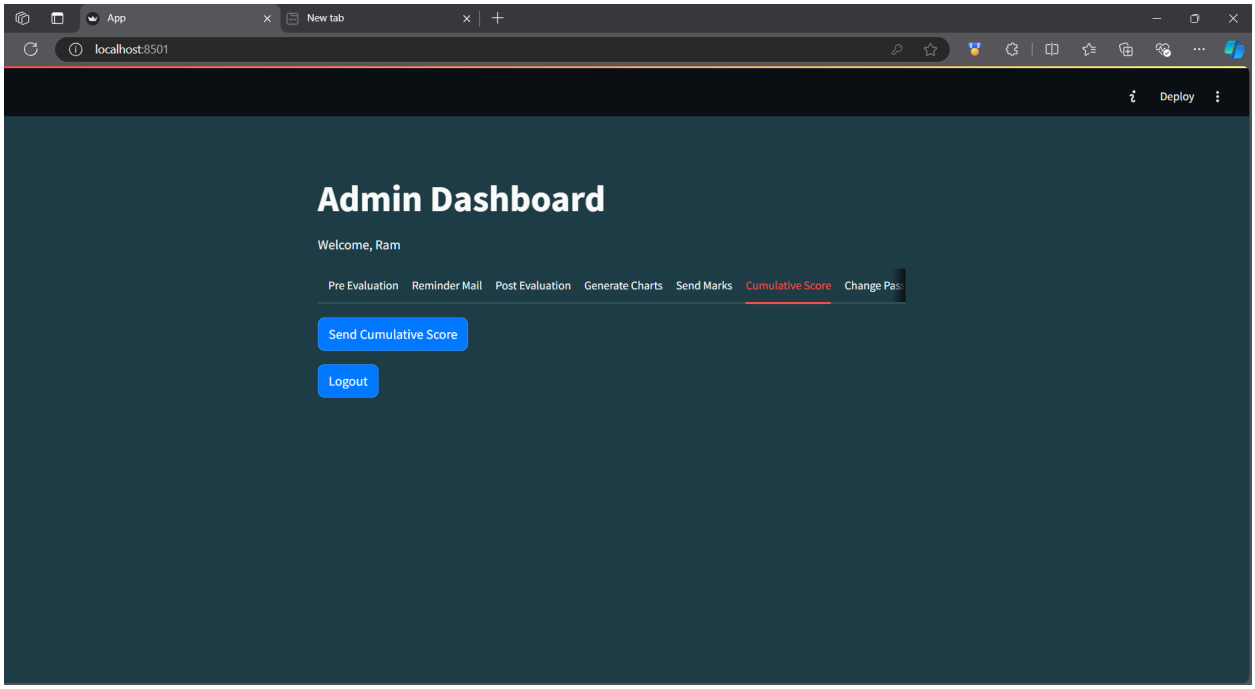
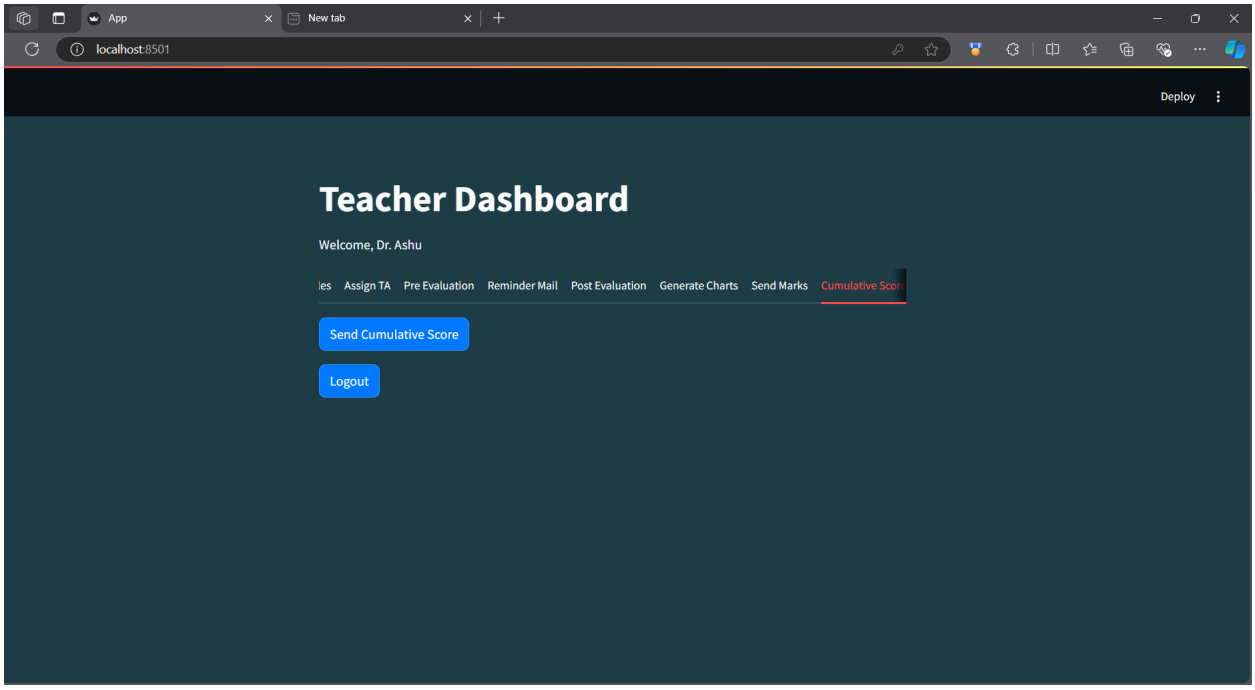
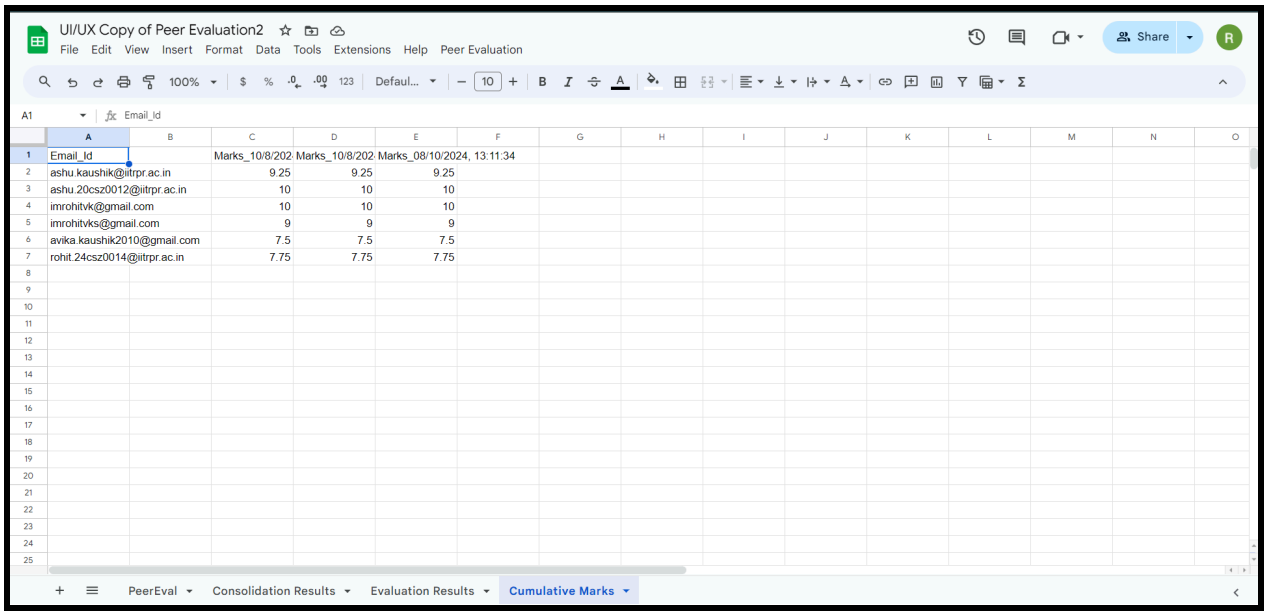
Peer Evaluation System UI/UX

Screenshots of the UI/UX design: -

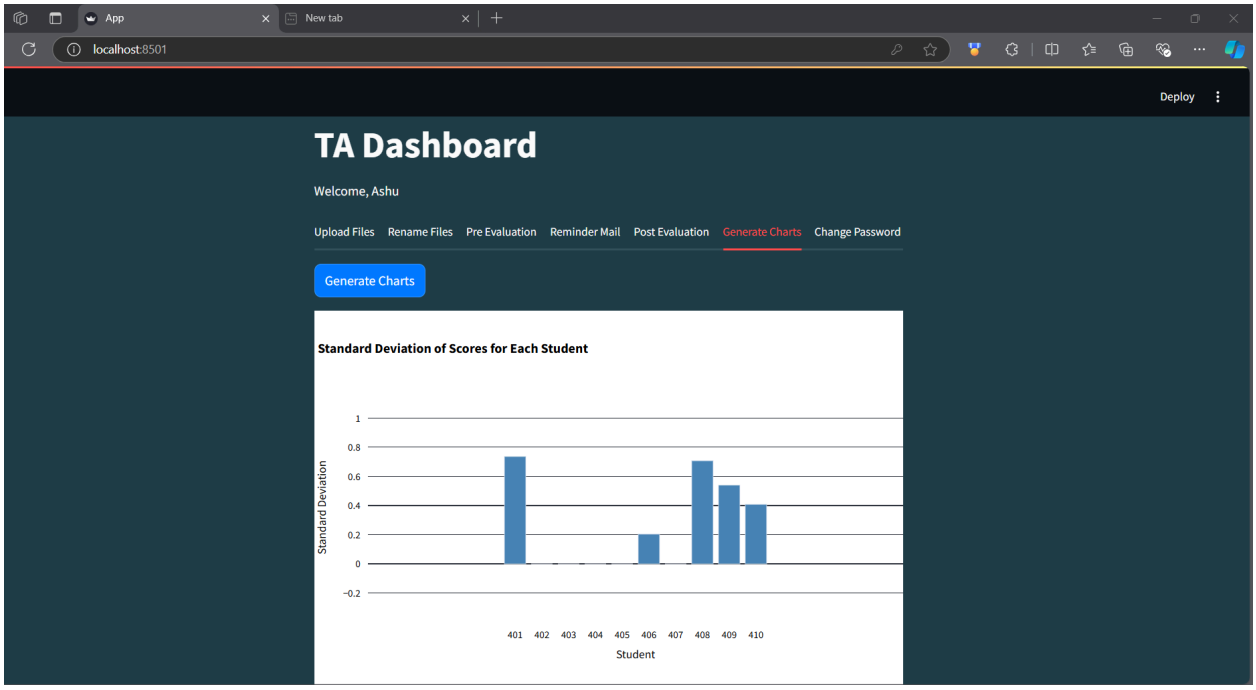
- The changes from the today's code are reflected below: -

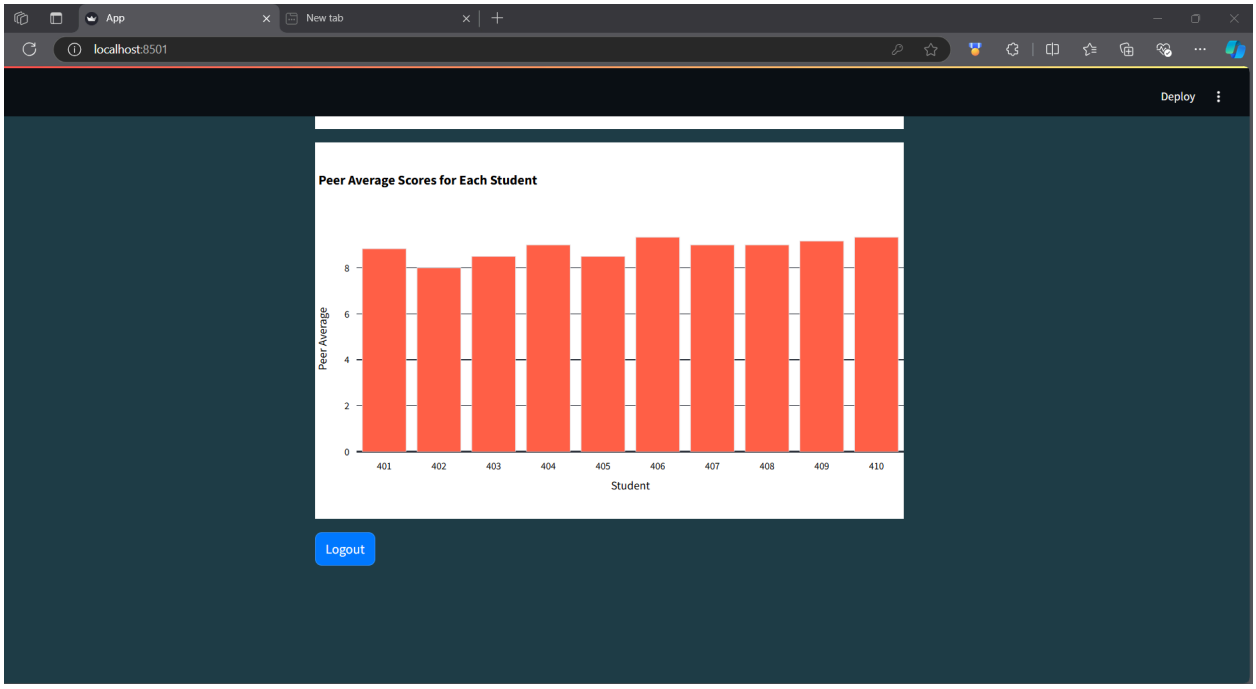


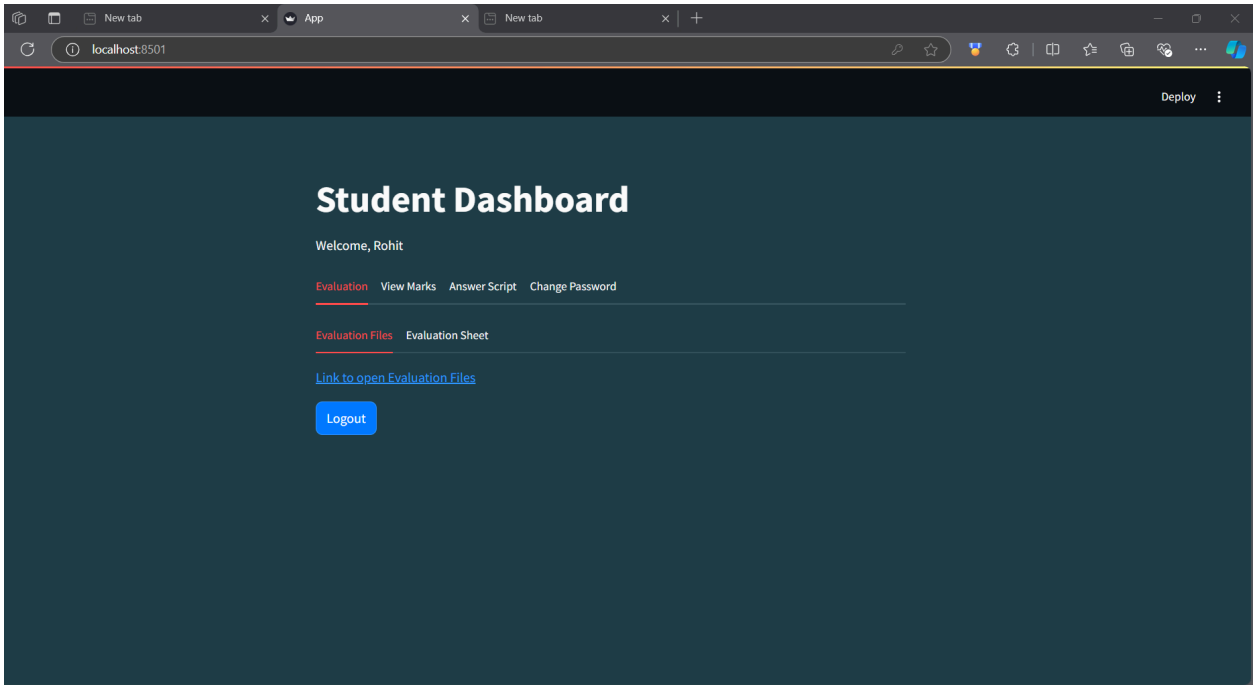




	A	B	C	D	E	F
1	Email_Id		Marks_10/8/202	Marks_10/8/202	Marks_08/10/2024, 13:11:34	
2	ashu.kaushik@iitrpr.ac.in		9.25	9.25	9.25	
3	ashu.20csz0012@iitrpr.ac.in		10	10	10	
4	imrohitvk@gmail.com		10	10	10	
5	imrohitvks@gmail.com		9	9	9	
6	avika.kaushik2010@gmail.com		7.5	7.5	7.5	
7	rohit.24csz0014@iitrpr.ac.in		7.75	7.75	7.75	




App
New tab
localhost:8501
Deploy
Peer Average Scores for Each Student
Peer Average
0
2
4
6
8
401
402
403
404
405
406
407
408
409
410
Student
Logout


New tab
App
New tab
localhost:8501
Deploy
Student Dashboard
Welcome, Rohit
Evaluation
View Marks
Answer Script
Change Password
Evaluation Files
Evaluation Sheet
Link to open Evaluation Files
Logout

Python Code: -

1. **App.py -**

```
import io
import re
import time
import bcrypt
import gspread
import requests
import streamlit as st
import matplotlib.pyplot as plt
import plotly.express as px
import pandas as pd
from Rename_File import process_signatures, load_stored_signatures
from googleapiclient.discovery import build
from googleapiclient.http import MediaIoBaseUpload
from googleapiclient.http import MediaIoBaseDownload
from oauth2client.service_account import ServiceAccountCredentials

# Google Sheets and Google Drive setup
SCOPE = [
  "https://spreadsheets.google.com/feeds",
  "https://www.googleapis.com/auth/drive"
]
CREDENTIALS_FILE = "peer-evaluation-sem1-e2fcf8b5fc27.json"
SHEET_NAME = "UserRoles"


# Initialize connection to Google Sheets
def connect_to_google_sheets():
  creds =
ServiceAccountCredentials.from_json_keyfile_name(CREDENTIALS_FILE,
SCOPE)
  client = gspread.authorize(creds)
  sheet = client.open(SHEET_NAME).sheet1
  return sheet
```

```
# Google Drive authentication
def authenticate_drive():
  creds =
ServiceAccountCredentials.from_json_keyfile_name(CREDENTIALS_FILE,
SCOPE)
  service = build('drive', 'v3', credentials=creds)
  return service


# Fetch users from Google Sheets
def get_users_from_sheets():
  sheet = connect_to_google_sheets()
  records = sheet.get_all_records()
  return records

def validate_password(password):
  pattern =
re.compile(r'^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?
&]{8,}$')
  return pattern.match(password)

# Add new user to Google Sheets with role auto-assignment
def register_user(username, password):
  sheet = connect_to_google_sheets()

  # Check if the email contains numeric values (assumed to be student)
  if re.search(r'\d', username):
    role = "Student"
  else:
    role = "Teacher"

  # Hash the password before saving
  hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

  new_user = [username, hashed_password.decode('utf-8'), role]

  #new_user = [username, password, role]
  sheet.append_row(new_user)
  return role
```

```
# Update role from Student to TA (only for Teachers)
def update_role_to_ta(username):
  sheet = connect_to_google_sheets()
  records = sheet.get_all_records()
  for i, user in enumerate(records, start=2):  # start=2 to account for 1-based index
in Google Sheets
      if user['username'] == username and user['role'] == 'Student':
        sheet.update_cell(i, 3, 'TA')  # Assuming role is in column 3
        return True
  return False



# Verify user credentials
def login(username, password, users):
  for user in users:
      if user['username'] == username:
        # Check if the password matches the stored hash
        if bcrypt.checkpw(password.encode('utf-8'),
user['password'].encode('utf-8')):
            st.session_state["login_status"] = True
            st.session_state["role"] = user["role"]
            st.session_state["username"] = username
            st.session_state["page"] = "dashboard"
            st.session_state["message"] = None
            return
        else:
            st.error("Incorrect Password!")
            time.sleep(2)
            st.rerun()
            return
  st.error("Incorrect Username or Password!")
  time.sleep(2)
  st.rerun()


# Logout function
def logout():
  st.session_state["login_status"] = False
```

```
  st.session_state["role"] = None
  st.session_state["username"] = None
  st.session_state["page"] = "login"
  st.success("Logging out!")
  time.sleep(0.5)
  #st.session_state["message"] = "Logged out successfully"

# Function to change password
def change_password(username, current_password, new_password):
  sheet = connect_to_google_sheets()
  records = sheet.get_all_records()

  # Find the user in the records
  for i, user in enumerate(records, start=2):  # start=2 for 1-based indexing
(Google Sheets)
      if user['username'] == username:
        # Check if the current password matches the stored hash
        if bcrypt.checkpw(current_password.encode('utf-8'),
user['password'].encode('utf-8')):
            # Hash the new password
            hashed_new_password = bcrypt.hashpw(new_password.encode('utf-8'),
bcrypt.gensalt()).decode('utf-8')

            # Update the password in the sheet
            sheet.update_cell(i, 2, hashed_new_password)
            return True  # Password changed successfully
        else:
            return False  # Current password is incorrect
  return False  # User not found


def change_password_dashboard():
  st.header("Change Password")

  current_password = st.text_input("Current Password", type="password")
  new_password = st.text_input("New Password", type="password")
  confirm_password = st.text_input("Confirm New Password", type="password")

  if st.button("Change Password"):
      if new_password != confirm_password:
```

```
            st.error("New password and confirm password do not match!")
        elif not validate_password(new_password):
            st.error(
                "Password must include at least: - \n1. One uppercase letter. \n2. One
lowercase letter. \n3. One special character. \n4. One numerical digit. \n5. Must be
at least 8 characters long.")
        else:
            success = change_password(st.session_state['username'],
current_password, new_password)
            if success:
                st.success("Password changed successfully!")
                time.sleep(2)
                logout()
                st.rerun()
            else:
                st.error("Failed to change password. Incorrect current password.")


def trigger_google_apps_script(function_name):
  #web_app_url =
"https://script.google.com/macros/s/AKfycbwlBil062YhNYcbIqmP9obfLBKgoeI
dTdRDQ_BOB4rF1S6JhTxvVFH8MhW2x84bgyAVag/exec"  # Replace with
your web app URL
  #web_app_url =
"https://script.google.com/macros/s/AKfycbwyD-ImNeqaa7NzPCU5AZ6978PSR
vQuuyEreskmCnrzoM0P30EGHuR-sIoqklHBHlNlxQ/exec"
  web_app_url =
"https://script.google.com/macros/s/AKfycbyyeKLZfm5VihTXPqu5qloIl2CEs8m
RU3iJuepZJ4AvWm3R0w3yWBG8INj9v2-TUeGk3w/exec"

  url = f"{web_app_url}?action={function_name}"  # Append the function name
as the 'action' parameter
  try:
    response = requests.get(url)
    if response.status_code == 200:
      st.success(f"{function_name} executed successfully!")
    else:
      st.error(f"Failed to execute {function_name}. Status code:
{response.status_code}")
  except Exception as e:
```

```
        st.error(f"An error occurred: {str(e)}")


# Function to check if a file already exists in Google Drive folder
def file_exists(drive_service, folder_id, file_name):
    query = f"'{folder_id}' in parents and name='{file_name}'"
    results = drive_service.files().list(q=query, spaces='drive', fields='files(id,
name)').execute()
    files = results.get('files', [])
    return any(file['name'] == file_name for file in files)


# Function to upload PDF files to Google Drive
def upload_pdfs(uploaded_files, folder_id):
    drive_service = authenticate_drive()
    count = 0

    for uploaded_file in uploaded_files:
        if file_exists(drive_service, folder_id, uploaded_file.name):
            #st.warning(f"PDF file '{uploaded_file.name}' already exists in the
folder.")
            continue

        file_metadata = {
            'name': uploaded_file.name,
            'parents': [folder_id]
        }
        media = MediaIoBaseUpload(uploaded_file, mimetype='application/pdf')
        drive_service.files().create(body=file_metadata, media_body=media,
fields='id').execute()
        count = count + 1
        #st.session_state["success_message"] = f"Uploaded PDF file
'{uploaded_file.name}' to Google Drive"

    st.success(f" The {count} files are uploaded to the Google Drive.")


# Function to upload Google Sheets files to Google Drive
def upload_sheets(uploaded_file, folder_id):
    drive_service = authenticate_drive()
```

```
  if file_exists(drive_service, folder_id, uploaded_file.name):
      st.warning(f"Google Sheet file '{uploaded_file.name}' already exists in the
folder.")
      return

  file_metadata = {
      'name': uploaded_file.name,
      'parents': [folder_id]
  }

  # Upload the file to Google Drive
  media = MediaIoBaseUpload(uploaded_file, mimetype=uploaded_file.type,
resumable=True)
  uploaded = drive_service.files().create(body=file_metadata,
media_body=media, fields='id').execute()

  st.success("The Excel sheet has been uploaded to the Google Drive.")


# Helper function to connect to a specific Google Sheet
def connect_to_google_sheets_with_name(sheet_name):
  creds =
ServiceAccountCredentials.from_json_keyfile_name(CREDENTIALS_FILE,
SCOPE)
  client = gspread.authorize(creds)
  sheet = client.open(sheet_name)
  return sheet


def get_student_details(username):
  # Connect to the specific Google Sheet containing marks
  sheet_name = "UI/UX Copy of Peer Evaluation2"
  sheet = connect_to_google_sheets_with_name(sheet_name)  # Modify to accept
a sheet name
  peer_eval_sheet = sheet.worksheet('PeerEval')  # Open the "PeerEval" sheet

  # Fetch all the data from the "PeerEval" sheet
  records = peer_eval_sheet.get_all_records()
```

```
    # Find marks for the current user
    for record in records:
        if record['EMail ID'] == username:   # Ensure this matches your column name
            return record['Average Marks'], record['Unique ID'], record['Assigned
Folder Link'], record['Spreadsheet Link']  # Returning the Average Mark's and
Unique id

    return None, None, None, None  # If no details found for the user

# Fetch the student's PDF from Google Drive using unique ID
def get_student_pdf(unique_id):
    drive_service = authenticate_drive()
    folder_id = "1fT-inciLQut85BGEQrjMSWbVRcTsdWfQ"
    query = f"'{folder_id}' in parents and name contains '{unique_id}'"
    results = drive_service.files().list(q=query, fields="files(id, name)").execute()
    files = results.get('files', [])

    if files:
        file_id = files[0]['id']
        file_name = files[0]['name']

        # Download the PDF
        request = drive_service.files().get_media(fileId=file_id)
        fh = io.BytesIO()
        downloader = MediaIoBaseDownload(fh, request)
        done = False
        while not done:
            status, done = downloader.next_chunk()

        fh.seek(0)
        return fh, file_name

    return None, None


def renaming_files():
    # Authenticate Google Drive
    service = authenticate_drive()

    # Google Drive folder IDs
```

```
  stored_signatures_folder_id = '14QLNPdIRUZ3ici-GePoEewUCmxemjhUD'
#The folder where we want to keep the Stored signature
  uploaded_signatures_folder_id = '1ORVrU-UoXyDS-1ovyuk7FAjb_p94gnsx'
#This will be the folder where our pdf files are kept
  destination_folder_id = '1bPhLMZONpsPDxM9z_vQD2J9jAjtwG3FG'  #
Folder where renamed files will be moved

  # Load stored signatures directly from Google Drive
  stored_signatures, stored_filenames = load_stored_signatures(service,
stored_signatures_folder_id)

  # Process uploaded signatures and copy renamed files to the destination folder
  matched_files = process_signatures(service, uploaded_signatures_folder_id,
stored_signatures, stored_filenames, destination_folder_id)

  print(f"Matching process completed. Total matched files:
{len(matched_files)}")

# Connect to Google Sheets
def fetch_sheet_data(sheet_name, worksheet_name):
  # gc = gspread.service_account(filename=CREDENTIALS_FILE)
  # spreadsheet = gc.open(sheet_name)
  # worksheet = spreadsheet.worksheet(worksheet_name)
  # return worksheet.get_all_values()
  creds =
ServiceAccountCredentials.from_json_keyfile_name(CREDENTIALS_FILE,
SCOPE)  # Replace with your credentials file
  client = gspread.authorize(creds)
  sheet = client.open(sheet_name).worksheet(worksheet_name)  # Open the
specific worksheet
  return sheet


# Fetch Peer Average and Standard Deviation data
def get_peer_average_data(sheet):
  data = sheet.get_all_values()

  for index, row in enumerate(data):
    if row[0] == "Peer":  # Check the first column of each row
      st_idx = index
```

```
        if row[0] == "SD":  # Check the first column of each row
            std_idx = index
        if row[0] == "Peer Average":  # Check the first column of each row
            pavg_idx = index

    # Extract student IDs, Standard Deviations, and Peer Averages
    student_ids = data[st_idx][1:]  # First row has student IDs starting from B1
    std_devs = list(map(float, data[std_idx][1:]))  # Row 6 has Standard Deviations
starting from B6
    peer_averages = list(map(float, data[pavg_idx][1:]))  # Row 7 has Peer
Averages starting from B7

    # Create a DataFrame with relevant data
    df = pd.DataFrame({
        'Student': student_ids,
        'Standard Deviation': std_devs,
        'Average Score': peer_averages
    })

    return df


# Generate Standard Deviation Chart
def create_standard_deviation_chart(df):
    fig = px.bar(df, x='Student', y='Standard Deviation', title='Standard Deviation of
Scores for Each Student',
                 color_discrete_sequence=['#4682B4'])
    fig.update_layout(yaxis_title='Standard Deviation', xaxis_title='Student',
                      plot_bgcolor='white', paper_bgcolor='white',  # Set background
colors to white
                      font_color='black',  title_font_color='black',
                      xaxis=dict(title_font=dict(color='black'),
tickfont=dict(color='black')),
                      yaxis=dict(title_font=dict(color='black'),tickfont=dict(color='black'))
                      )
    return fig


# Generate Peer Average Scores Chart
def create_average_scores_chart(df):
```

```
  fig = px.bar(df, x='Student', y='Average Score', title='Peer Average Scores for
Each Student',
               color_discrete_sequence=['#FF6347'])
  fig.update_layout(yaxis_title='Peer Average', xaxis_title='Student',
                    plot_bgcolor='white', paper_bgcolor='white',  # Set background
colors to white
                    font_color='black',  title_font_color='black',
                    xaxis=dict(title_font=dict(color='black'),
tickfont=dict(color='black')),
                    yaxis=dict(title_font=dict(color='black'),
tickfont=dict(color='black'))
                    )
  return fig

# Trigger when "Generate Charts" button is clicked
def generate_charts():
  sheet = fetch_sheet_data('UI/UX Copy of Peer Evaluation2', 'Evaluation
Results')
  df = get_peer_average_data(sheet)  # Get the data
  # Display two charts
  st.plotly_chart(create_standard_deviation_chart(df))
  st.plotly_chart(create_average_scores_chart(df))



# Function to list all files in the given folder (Google Drive)
def list_spreadsheets_in_folder(folder_id):
  service = authenticate_drive()  # Authenticate Drive using the new function
  query = f"'{folder_id}' in parents and
(mimeType='application/vnd.google-apps.spreadsheet' OR
mimeType='application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
')"
  results = service.files().list(q=query).execute()
  files = results.get('files', [])

  if not files:
      print('No spreadsheets found in the folder.')
      return []

  return files
```

```
# Function to copy data from each spreadsheet to the target sheet
def copy_data_to_peer_eval(folder_id, target_spreadsheet_name,
target_worksheet_name):
  # Get the target spreadsheet using the new function
  target_spreadsheet =
connect_to_google_sheets_with_name(target_spreadsheet_name)

  try:
    target_worksheet = target_spreadsheet.worksheet(target_worksheet_name)
  except gspread.WorksheetNotFound:
    print(f"Worksheet '{target_worksheet_name}' not found in the target
spreadsheet.")
    return

  # List all spreadsheets in the folder
  files = list_spreadsheets_in_folder(folder_id)

  for file in files:
    print(f"Copying data from: {file['name']}")
    # Open each spreadsheet and copy its data
    source_spreadsheet = connect_to_google_sheets_with_name(file['name'])
    source_worksheet = source_spreadsheet.get_worksheet(0)  # Assuming data
is in the first worksheet
    source_data = source_worksheet.get_all_values()

    # Append data to the target worksheet
    if source_data:
      target_worksheet.append_rows(source_data, value_input_option="RAW")

  print("Data copied to PeerEval worksheet successfully.")

def admin_dashboard():
 st.title("Admin Dashboard")
 st.write(f"Welcome, {st.session_state['username'].split('.')[0].capitalize()}")


 # Create tabs for each action
 tab, tab0, tab1, tab2, tab3, tab4, tab5, tab6, tab7, tab8 = st.tabs(
```

```
    ["Upload Data", "Rename Files", "Assign TA", "Pre Evaluation", "Reminder
Mail", "Post Evaluation", "Generate Charts",
     "Send Marks", "Cumulative Score", "Change Password"])


 # Tab for File upload option
 with tab:
    # Folder ID for the Google Drive folder where the files will be saved
    folder_id = "1fT-inciLQut85BGEQrjMSWbVRcTsdWfQ"  # Replace this
with your folder ID

    # Allow file upload for multiple Google Sheets
    st.subheader("Upload Student Information")
    sheet_files = st.file_uploader("Upload Google Sheet", type=["xlsx"],
accept_multiple_files=False,
                                   key="sheet_uploader")

    if sheet_files:
        upload_sheets(sheet_files, folder_id)

    # Allow file upload for multiple PDFs
    st.subheader("Upload PDF Files")
    pdf_files = st.file_uploader("Upload PDF files", type=["pdf"],
accept_multiple_files=True, key="pdf_uploader")

    if pdf_files:
        upload_pdfs(pdf_files, folder_id)

 with tab0:
    if st.button("Rename Files"):
        renaming_files()

 # Tab for TA update
 with tab1:
    student_username = st.text_input("Enter Student's Username")
    if st.button("Update Role to TA"):
        if update_role_to_ta(student_username):
            st.success(f"{student_username.split('.')[0].capitalize()}'s role updated to
TA.")
        else:
```

```
            st.error("Failed to update the role. Check if the username exists and belongs to a student.")

    # Tab for Pre Evaluation
    with tab2:
        #target_spreadsheet_name = "UI/UX Copy of Peer Evaluation2"
        target_spreadsheet_name = "Sample_Run"
        target_worksheet_name = "PeerEval"
        if st.button("Pre Evaluation"):
            copy_data_to_peer_eval(folder_id, target_spreadsheet_name, target_worksheet_name)
            #trigger_google_apps_script("PreEval")

    # Tab for Checking Pending Evaluations
    with tab3:
        if st.button("Reminder Mail"):
            trigger_google_apps_script("CheckEval")

    # Tab for Post Evaluation
    with tab4:
        if st.button("Post Evaluation"):
            trigger_google_apps_script("PostEval")

    # Tab for Generating Charts
    with tab5:
        if st.button("Generate Charts"):
            generate_charts()
            #trigger_google_apps_script("GenChart")

    # Tab for Sending Marks
    with tab6:
        if st.button("Send Marks"):
            trigger_google_apps_script("SendMail")

    with tab7:
        if st.button("Send Cumulative Score"):
            trigger_google_apps_script("SendFinalM")

    with tab8:
        change_password_dashboard()
```

```
def teacher_dashboard():
  st.title("Teacher Dashboard")
  #st.write(f"Welcome, {st.session_state['username']}")
  var_user = st.session_state['username'].split('@')[0]
  if '.' in var_user:
    st.write(f"Welcome, Dr. {var_user.split('.')[0].capitalize()}")
  else:
    st.write(f"Welcome, Dr. {var_user.capitalize()}")

  # Create tabs for each action
  tab, tab0, tab1, tab2, tab3, tab4, tab5, tab6, tab7 = st.tabs(["Rename Files",
"Assign TA", "Pre Evaluation", "Reminder Mail",
                                  "Post Evaluation", "Generate Charts", "Send
Marks", "Cumulative Score", "Change Password"])

  # Tab for TA update
  with tab:
    if st.button("Rename Files"):
      renaming_files()
  with tab0:
    student_username = st.text_input("Enter Student's Username")
    if st.button("Update Role to TA"):
      if update_role_to_ta(student_username):
        st.success(f"{student_username.split('.')[0].capitalize()}'s role updated to
TA.")
      else:
        st.error("Failed to update the role. Check if the username exists and
belongs to a student.")


  # Tab for Pre Evaluation
  with tab1:
    if st.button("Pre Evaluation"):
      trigger_google_apps_script("PreEval")

  # Tab for Checking Pending Evaluations
  with tab2:
    if st.button("Reminder Mail"):
```

```
            trigger_google_apps_script("CheckEval")

    # Tab for Post Evaluation
    with tab3:
        if st.button("Post Evaluation"):
            trigger_google_apps_script("PostEval")

    # Tab for Generating Charts
    with tab4:
        if st.button("Generate Charts"):
            generate_charts()
            #trigger_google_apps_script("GenChart")
    with tab5:
        if st.button("Send Marks"):
            trigger_google_apps_script("SendMail")

    with tab6:
        if st.button("Send Cumulative Score"):
            trigger_google_apps_script("SendFinalM")

    with tab7:
        change_password_dashboard()



# Role-based content: Teacher Dashboard with multiple file uploads
def ta_dashboard():
    st.title("TA Dashboard")
    st.write(f"Welcome, {st.session_state['username'].split('.')[0].capitalize()}")
    #st.write(f"Welcome, {st.session_state['username']}")

    # Create tabs for each action
    tab, tab5, tab0, tab1, tab2, tab3, tab4 = st.tabs(
        ["Upload Data", "Rename Files", "Pre Evaluation", "Reminder Mail", "Post
Evaluation", "Generate Charts", "Change Password"])

    # Tab for File upload option
    with tab:
        # Folder ID for the Google Drive folder where the files will be saved
```

```
    folder_id = "1fT-inciLQut85BGEQrjMSWbVRcTsdWfQ"  # Replace this with your folder ID

    # Allow file upload for multiple Google Sheets
    st.subheader("Upload Student Information")
    sheet_files = st.file_uploader("Upload Google Sheet", type=["xlsx"], accept_multiple_files=False,
                                   key="sheet_uploader")

    if sheet_files:
        upload_sheets(sheet_files, folder_id)

    # Allow file upload for multiple PDFs
    st.subheader("Upload PDF Files")
    pdf_files = st.file_uploader("Upload PDF files", type=["pdf"], accept_multiple_files=True, key="pdf_uploader")

    if pdf_files:
        upload_pdfs(pdf_files, folder_id)


with tab5:
    if st.button("Rename Files"):
        renaming_files()

# Tab for Pre Evaluation
with tab0:
    if st.button("Pre Evaluation"):
        trigger_google_apps_script("PreEval")

# Tab for Checking Pending Evaluations
with tab1:
    if st.button("Reminder Mail"):
        trigger_google_apps_script("CheckEval")

# Tab for Post Evaluation
with tab2:
    if st.button("Post Evaluation"):
        trigger_google_apps_script("PostEval")
```

```
  # Tab for Generating Charts
  with tab3:
    if st.button("Generate Charts"):
      generate_charts()
      #trigger_google_apps_script("GenChart")

  with tab4:
    change_password_dashboard()



def student_dashboard():
  st.title("Student Dashboard")
  st.write(f"Welcome, {st.session_state['username'].split('.')[0].capitalize()}")
  #st.write(f"Welcome, {st.session_state['username']}")


  # Creating tabs
  tab1, tab2, tab3, tab4 = st.tabs(["Evaluation", "View Marks", "Answer Script",
"Change Password"])

  # with tab0:
  #   change_password_dashboard()
  #   if st.session_state["username"]:
  #     # Fetch marks, unique ID, and spreadsheet link using the session's
username
  #     marks, unique_id, folder_link, sheet_link =
get_student_details(st.session_state["username"])
  #   else:
  #     st.error("Username is Incorrect!")

  # Tab for opening the peer evaluation spreadsheet
  with tab1:
    if st.session_state["username"]:
      # Fetch marks, unique ID, and spreadsheet link using the session's
username
      marks, unique_id, folder_link, sheet_link =
get_student_details(st.session_state["username"])
    else:
      st.error("Username is Incorrect!")
```

```
        t1, t2 = st.tabs(["Evaluation Files", "Evaluation Sheet"])
        with t1:
            if folder_link:
                st.markdown(f"[Link to open Evaluation Files]({folder_link})",
unsafe_allow_html=True)
            else:
                st.error("Folder link not found,")
        with t2:
            if sheet_link:
                st.markdown(f"[Link to open Evaluation Sheet]({sheet_link})",
unsafe_allow_html=True)
            else:
                st.error("Spreadsheet link not found.")

    # Tab for viewing marks
    with tab2:
        if st.button("See Marks"):
            if marks and unique_id:
                st.write(f"Your evaluation marks are =  {marks}")
            else:
                st.error("No marks are available.")

    # Tab for downloading PDF
    with tab3:
        pdf_file, file_name = get_student_pdf(unique_id)
        if pdf_file:
            st.download_button(
                label="Download your Evaluation PDF",
                data=pdf_file,
                file_name=file_name,
                mime='application/pdf'
            )
        else:
            st.error("PDF not found.")

    with tab4:
        change_password_dashboard()
```

```
# Main Streamlit app
def main():
  # Initialize session state variables if not present
  if "login_status" not in st.session_state:
    st.session_state["login_status"] = False
  if "role" not in st.session_state:
    st.session_state["role"] = None
  if "username" not in st.session_state:
    st.session_state["username"] = None
  if "page" not in st.session_state:
    st.session_state["page"] = "login"
  if "message" not in st.session_state:
    st.session_state["message"] = None
  if "success_message" not in st.session_state:
    st.session_state["success_message"] = None


  # Set background color and input field styling using HTML
  st.markdown(
    """
    <style>
    .stApp {
      background-color: #1f3f49;  /* Light blue background */
    }
    .stTextInput>div>input, .stPasswordInput>div>input {
      background-color: white;  /* White background for text and password
inputs */
      color: black;  /* Text color for input fields */
    }
    .stButton>button {
      background-color: #007bff;  /* Optional: Style buttons with a color */
      color: white;
    }
    </style>
    """,
    unsafe_allow_html=True
  )
```

```
# Page routing based on session state
if st.session_state["page"] == "login":
    st.title("Peer Evaluation System")

    # Tabs for Login and Registration
    tab1, tab2 = st.tabs(["Login", "Register"])

    with tab1:
        st.header("Login")

        with st.form(key='login_form'):
            username = st.text_input("Email ID")
            password = st.text_input("Password", type="password")
            submit_button = st.form_submit_button("Login")

            if submit_button:
                users = get_users_from_sheets()
                login(username, password, users)
                if st.session_state["login_status"]:
                    st.rerun()

    with tab2:
        st.header("Register")

        with st.form(key='register_form'):
            reg_username = st.text_input("Email ID", key='reg_username')
            reg_password = st.text_input("Password", type="password",
key='reg_password')
            register_button = st.form_submit_button("Register")

            if register_button:
                if not reg_username.endswith("@iitrpr.ac.in"):
                    st.error("Email ID must end with @iitrpr.ac.in")
                elif not validate_password(reg_password):
                    st.error("Password must include at least One: - \n1. Uppercase
letter. \n2. Lowercase letter. \n3. Special character. \n4. Numerical digit. \n5. Must
be at least 8 characters long.")
                else:
                    users = get_users_from_sheets()
                    if any(user['username'] == reg_username for user in users):
```

```
                st.error("Username already exists")
            else:
                role = register_user(reg_username, reg_password)
                st.success(f"User registered successfully with role: {role}")
                time.sleep(2)
                # Redirect to the login page
                st.session_state["page"] = "login"
                st.rerun()

    elif st.session_state["page"] == "dashboard":
        if st.session_state["role"] == "Admin":
            admin_dashboard()
        elif st.session_state["role"] == "Teacher":
            teacher_dashboard()
        elif st.session_state["role"] == "TA":
            ta_dashboard()
        elif st.session_state["role"] == "Student":
            student_dashboard()



        # Logout button
        if st.button("Logout"):
            logout()
            st.rerun()


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

2. **Rename_File.py -**

```
import os
import io
import cv2
import numpy as np
from googleapiclient.discovery import build
from google.oauth2.service_account import Credentials
from skimage.metrics import structural_similarity as ssim
from scipy.ndimage import center_of_mass
from googleapiclient.http import MediaIoBaseDownload

SCOPE = [
  "https://spreadsheets.google.com/feeds",
  "https://www.googleapis.com/auth/drive"
]
CREDENTIALS_FILE = "peer-evaluation-sem1-e2fcf8b5fc27.json"

# Authenticate Google Drive
def authenticate_drive():
  creds = Credentials.from_service_account_file(CREDENTIALS_FILE)
  service = build('drive', 'v3', credentials=creds)
  return service

# List files in a Google Drive folder
def list_files_in_folder(service, folder_id):
  results = service.files().list(
      q=f"'{folder_id}' in parents and mimeType !=
'application/vnd.google-apps.folder'",
      pageSize=1000, fields="files(id, name)").execute()
  return results.get('files', [])

# Read file content from Google Drive (read as image data)
def read_drive_image_file(service, file_id):
  request = service.files().get_media(fileId=file_id)
  file_bytes = io.BytesIO()
  downloader = MediaIoBaseDownload(file_bytes, request)
  done = False
  while not done:
      _, done = downloader.next_chunk()
```

```
    file_bytes.seek(0)  # Reset pointer to start
    file_array = np.frombuffer(file_bytes.read(), np.uint8)
    image = cv2.imdecode(file_array, cv2.IMREAD_GRAYSCALE)  # Decode as
grayscale image
    return image

# Copy and rename a file in Google Drive to a new folder
def copy_and_rename_drive_file(service, file_id, new_name,
destination_folder_id):
    file_metadata = {
        'name': new_name,
        'parents': [destination_folder_id]  # Move file to this folder
    }
    copied_file = service.files().copy(fileId=file_id, body=file_metadata).execute()
    return copied_file

# Delete a file in Google Drive
def delete_drive_file(service, file_id):
    service.files().delete(fileId=file_id).execute()

# Preprocess image (resize and normalize)
def preprocess_image(image):
    image = cv2.resize(image, (256, 256))  # Resize
    image = image / 255.0  # Normalize pixel values
    return image

# Align signatures by centering the image based on center of mass
def align_images(image):
    cy, cx = center_of_mass(image)
    height, width = image.shape
    shift_x = int(width / 2 - cx)
    shift_y = int(height / 2 - cy)
    translation_matrix = np.float32([[1, 0, shift_x], [0, 1, shift_y]])
    aligned_image = cv2.warpAffine(image, translation_matrix, (width, height))
    return aligned_image

# Compare signatures using SSIM
def compare_signatures(stored_sig, uploaded_sig):
    aligned_stored_sig = align_images(stored_sig)
    aligned_uploaded_sig = align_images(uploaded_sig)
```

```
  score, _ = ssim(aligned_stored_sig, aligned_uploaded_sig, full=True,
data_range=1.0)
  return score

# Load stored signatures directly from Google Drive (in memory)
def load_stored_signatures(service, stored_signatures_folder_id):
  stored_signatures = []
  stored_filenames = []
  files = list_files_in_folder(service, stored_signatures_folder_id)

  for file in files:
    image = read_drive_image_file(service, file['id'])
    image = preprocess_image(image)
    stored_signatures.append(image)
    stored_filenames.append(file['name'])
    print(f"Loaded stored signature: {file['name']}")  # Debugging info
  return stored_signatures, stored_filenames

# Compare uploaded signature with stored signatures and copy file to a new
folder if matched
def process_signatures(service, uploaded_signatures_folder_id, stored_signatures,
stored_filenames, destination_folder_id, threshold=0.85):
  uploaded_files = list_files_in_folder(service, uploaded_signatures_folder_id)
  matched_files = []

  for uploaded_file in uploaded_files:
    uploaded_image = read_drive_image_file(service, uploaded_file['id'])
    uploaded_image = preprocess_image(uploaded_image)

    # Compare each uploaded signature with stored signatures
    matched_filename = None
    print(f"Comparing uploaded file: {uploaded_file['name']}")  # Debugging
info
    for stored_signature, stored_filename in zip(stored_signatures,
stored_filenames):
      similarity_score = compare_signatures(stored_signature, uploaded_image)
      print(f"Comparing with stored file: {stored_filename}, Similarity:
{similarity_score}")  # Debugging info
      if similarity_score > threshold:  # Match threshold
        matched_filename = stored_filename
```

```
            print(f"Match found: {uploaded_file['name']} -> {stored_filename}")  #
Debugging info
            break

        # If a match is found, copy and rename the file in the new folder
        if matched_filename:
            copied_file = copy_and_rename_drive_file(service, uploaded_file['id'],
matched_filename, destination_folder_id)
            # Delete original file after copying
            #delete_drive_file(service, uploaded_file['id'])
            print(f"Copied and renamed {uploaded_file['name']} to
{matched_filename} in destination folder")
            matched_files.append((uploaded_file['name'], matched_filename))
        else:
            print(f"No match found for {uploaded_file['name']}")  # Debugging info

    return matched_files
```

3. **Javascript Code: -**

```
function sendFinalMarksToAllStudents() {
  //mapCumulativeMarks()
  sendMarksByEmail()
}

function mapCumulativeMarks() {
  var sourceSheetName = "PeerEval"; // Source worksheet name
  var targetSheetName = "Cumulative Marks"; // Target worksheet name (to be
created if not present)

  var spreadsheet = SpreadsheetApp.getActiveSpreadsheet();

  // Get the source sheet
  var sourceSheet = spreadsheet.getSheetByName(sourceSheetName);
  if (!sourceSheet) {
    Logger.log("Source sheet not found: " + sourceSheetName);
    return;
  }
```

```
  // Check if the target sheet exists, if not, create it
  var targetSheet = spreadsheet.getSheetByName(targetSheetName);
  if (!targetSheet) {
    Logger.log("Target sheet not found. Creating a new one: " + targetSheetName);
    targetSheet = spreadsheet.insertSheet(targetSheetName);
  }

  // Get data from the source sheet
  var sourceData = sourceSheet.getDataRange().getValues();

  // Find the "EMail ID" column in the source sheet
  var sourceHeaders = sourceData[0]; // First row of the source sheet
  var emailIdColIndexSource = sourceHeaders.indexOf("EMail ID");

  if (emailIdColIndexSource === -1) {
    Logger.log('EMail ID column not found in the source sheet.');
    return;
  }

  // Find the "Average Marks" column in the source sheet
  var averageMarksColIndexSource = sourceHeaders.indexOf("Average Marks");

  if (averageMarksColIndexSource === -1) {
    Logger.log("Average Marks column not found in the source sheet.");
    return;
  }

  // Retrieve the relevant columns for Peer Emails and Peer Average Marks from
the source sheet
  var peerEmails = sourceData.slice(1).map(row =>
row[emailIdColIndexSource]); // Get email IDs from the column
  var peerAverageMarks = sourceData.slice(1).map(row =>
row[averageMarksColIndexSource]); // Get Average Marks from the column

  // Get the data from the target sheet (Cumulative Marks)
  var targetData = targetSheet.getDataRange().getValues();

  // Check if "Email_Id" column exists in the target sheet, if not, create it
  var targetHeaders = targetData.length ? targetData[0] : [];
  var emailIdColIndexTarget = targetHeaders.indexOf("Email_Id");
```

```
  if (emailIdColIndexTarget === -1) {
    Logger.log("Email_Id column not found in the target sheet. Adding it.");

    // If "Email_Id" column is not found, add it as the first column
    emailIdColIndexTarget = 0;
    targetSheet.insertColumnBefore(1);
    targetSheet.getRange(1, 1).setValue("Email_Id");
    targetHeaders = ["Email_Id"].concat(targetHeaders); // Update the header
  }

  // Add a new column for Average Marks with a unique label
  var timestamp = new Date().toLocaleString(); // Get a timestamp to create a
unique column header
  var newColHeader = "Marks_" + timestamp;
  var newColIndex = targetHeaders.length + 1;
  targetSheet.getRange(1, newColIndex).setValue(newColHeader); // Add the new
header in the target sheet

  // Iterate over the peerEmails from the source sheet and map the marks
  for (var i = 0; i < peerEmails.length; i++) {
    var peerEmail = peerEmails[i];
    if (!peerEmail) continue; // Skip if there's no email ID in the source sheet

    var peerAverage = peerAverageMarks[i];
    var foundInTarget = false;

    // Search for the email ID in the target sheet
    for (var j = 1; j < targetData.length; j++) {
      var targetEmailId = targetData[j][emailIdColIndexTarget];
      if (targetEmailId === peerEmail) {
        targetSheet.getRange(j + 1, newColIndex).setValue(peerAverage); // Update
marks if email found
        foundInTarget = true;
        break;
      }
    }

    // If the email ID was not found in the target sheet, append it at the end
    if (!foundInTarget) {
```

```
      var newRowIndex = targetSheet.getLastRow() + 1;
      targetSheet.getRange(newRowIndex, emailIdColIndexTarget +
1).setValue(peerEmail); // Add new email
      targetSheet.getRange(newRowIndex, newColIndex).setValue(peerAverage); //
Add corresponding average mark
    }
  }

  Logger.log("Marks mapping completed with a new column added: " +
newColHeader);
}



function sendMarksByEmail() {
  var targetSheetName = "Cumulative Marks"; // Target sheet name
  var spreadsheet = SpreadsheetApp.getActiveSpreadsheet();

  // Get the target sheet
  var targetSheet = spreadsheet.getSheetByName(targetSheetName);
  if (!targetSheet) {
    Logger.log("Target sheet not found: " + targetSheetName);
    return;
  }

  // Get the target sheet data
  var targetData = targetSheet.getDataRange().getValues();
  if (targetData.length === 0) {
    Logger.log("Target sheet is empty.");
    return;
  }

  // Get the headers (first row) of the target sheet
  var headers = targetData[0];

  // Find the "Email_Id" column index
  var emailIdColIndex = headers.indexOf("Email_Id");
  if (emailIdColIndex === -1) {
    Logger.log("Email_Id column not found in the target sheet.");
    return;
```

```
  }

  // Find the columns with "Marks_timestamp" format
  var marksColumns = [];
  for (var i = 0; i < headers.length; i++) {
    if (headers[i].startsWith("Marks_")) {
      marksColumns.push(i);
    }
  }

  if (marksColumns.length === 0) {
    Logger.log("No Marks_timestamp columns found in the target sheet.");
    return;
  }

  // Loop through each row and calculate total marks, then send an email
  for (var i = 1; i < targetData.length; i++) {
    var emailId = targetData[i][emailIdColIndex];

    if (emailId) {
      var totalMarks = 0;

      // Sum all the marks from the "Marks_timestamp" columns
      for (var j = 0; j < marksColumns.length; j++) {
        var mark = parseFloat(targetData[i][marksColumns[j]]);
        if (!isNaN(mark)) {
          totalMarks += mark;
        }
      }

      // Send an email with the total marks
      var subject = "Your Total Marks in the Course";
      var body = "Dear Student,\n\nYour total marks in the course are: - " +
totalMarks + ".\n\nThanks & Regards,\nCSE, IIT Ropar";

      MailApp.sendEmail(emailId, subject, body);
      Logger.log("Email sent to: " + emailId + " with total marks: " + totalMarks);
    }
  }
}
```