from HumanDataExtraction import HumanDataExtraction  
from HumanDataPebl import HumanDataPebl  
from AnalysisData import AnalysisData  
  
  
# Press the green button in the gutter to run the script.  
if \_\_name\_\_ == '\_\_main\_\_':  
 DataPath = r'C:\Users\e3bom\Desktop\Human Bio Signals Analysis'  
 Make\_Trigger=False  
 Make\_DataSet=True  
 Analysis\_DataSet=True  
  
 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Make Trigger Table\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 if Make\_Trigger:  
 hdp = HumanDataPebl(DataPath)  
 hdp.Make\_Trigger\_Table(ID=None,rangeID=False)  
 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SortData\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 if Make\_DataSet:  
  
 PreProcessing = False  
 Dataset=True  
 Combine = True  
 HRV = True  
 RSP = False  
 EDA = False  
  
 hde = HumanDataExtraction(DataPath)  
 if PreProcessing:  
 hde.Check\_MedianFilter(ID=None,rangeID=False)  
  
 if Dataset:  
 hde.CleanData(ID=None,rangeID=False)  
 hde.CreateDataset(ID=None,rangeID=False)  
  
 if Combine:  
 hde.AX\_plot\_signals\_VAS(ID=None,rangeID=False)  
  
 if RSP:  
 hde.RSP\_Parts(ID=None, Group=None)  
  
 if EDA:  
 hde.AX\_plot\_3\_part\_HRV(ID = [19,20,21])  
 hde.AX\_plot\_3in1norm\_EDA(ID=[19, 20, 21])  
 hde.AX\_plot\_3in1\_EDA(ID=[19, 20, 21])  
 hde.AX\_plot\_3\_part\_EDA(ID=[19, 20, 21])  
 if HRV:  
 hde.HRV\_Window\_Feature(ID=None, Group=None)  
 hde.HRV\_Window\_Feature\_all()  
 hde.HRV\_Window\_2Features(ID = 27, Group = 'breath')  
 hde.AX\_plot\_3\_part\_HRV()  
  
 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SortData\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 if Analysis\_DataSet:  
 ad = AnalysisData(DataPath)  
 ad.ML\_models\_all()  
 # # Utilities.save\_dataframe(hde.sorted\_DATA, dataframe\_path, f'{Analysis\_Type}\_Sorted\_DATA')

from Utilities import Utilities  
import pandas as pd  
import os  
import bioread  
import re  
import datetime  
  
  
# Press the green button in the gutter to run the script.  
class HumanDataPebl():  
 def \_\_init\_\_(self,Directory):  
 self.path=Directory  
 self.parlist\_df=pd.DataFrame()  
 def structure\_data(self,df):  
 *"""  
 Function to structure task data from raw dataframe.  
 Handles three types of tasks: CS, PASAT, and TWOCOL.  
 Returns a new dataframe with columns: Sub, Task, Start, End  
 """* # Convert the Time column to datetime format  
 df["Time"] = pd.to\_datetime(df["Time"])  
  
 # Initialize the new structured list  
 structured\_data = []  
  
 # Iterate through the DataFrame to match START and END times  
 start\_times = {}  
  
 for \_, row in df.iterrows():  
 part = row["Part"]  
 sub = row["Sub"]  
 time = row["Time"]  
  
 if "Start\_" in part or "STARTED" in part:  
 task\_name = part.replace("Start\_", "").replace("PASAT\_STARTED ", "PASAT ").replace("TWOCOL\_STARTED ", "TWOCOL ")  
 start\_times[(sub, task\_name)] = time  
 elif "End\_" in part or "ENDED" in part:  
 task\_name = part.replace("End\_", "").replace("PASAT\_ENDED ", "PASAT ").replace("TWOCOL\_ENDED ", "TWOCOL ")  
 start\_time = start\_times.get((sub, task\_name), None)  
 if start\_time:  
 structured\_data.append([sub, f"{task\_name}", start\_time, time])  
 del start\_times[(sub, task\_name)]  
  
 # Convert structured data to DataFrame  
 final\_df = pd.DataFrame(structured\_data, columns=["Sub", "Task", "Start", "End"])  
 final\_df["Task"]=final\_df["Task"].str.strip()  
 # Merge CS\_B1 and CS\_B2 into CB\_easy and CB\_hard  
 final\_df.loc[(final\_df["Task"] == "CS\_B1 3000") | (final\_df["Task"] == "CS\_B2 3000"), "Task"] = "CB\_easy"  
 final\_df.loc[(final\_df["Task"] == "CS\_B1 650") | (final\_df["Task"] == "CS\_B2 650"), "Task"] = "CB\_hard"  
  
 # Merge consecutive CB\_easy and CB\_hard tasks  
 merged\_rows = []  
 previous\_row = None  
  
 for \_, row in final\_df.iterrows():  
 if previous\_row is not None and previous\_row["Task"] == row["Task"] and previous\_row["Sub"] == row["Sub"] and previous\_row["Task"]!='Break':  
 # Update the end time of the previous row  
 previous\_row["End"] = row["End"]  
 else:  
 # Add previous row to merged list before starting a new one  
 if previous\_row is not None:  
 merged\_rows.append(previous\_row)  
 previous\_row = row.copy() # Start a new merging process  
  
 # Append the last merged row  
 if previous\_row is not None:  
 merged\_rows.append(previous\_row)  
  
 # Convert back to DataFrame  
 final\_df = pd.DataFrame(merged\_rows)  
 # Merge PASAT and TWOCOL tasks into simplified names  
 final\_df.loc[final\_df["Task"] == "TWOCOL 12000", "Task"] = "TC\_easy"  
 final\_df.loc[final\_df["Task"] == "TWOCOL 7000", "Task"] = "TC\_hard"  
 final\_df.loc[final\_df["Task"] == "PASAT 3000", "Task"] = "PA\_easy"  
 final\_df.loc[final\_df["Task"] == "PASAT 2400", "Task"] = "PA\_medium"  
 final\_df.loc[final\_df["Task"] == "PASAT 1800", "Task"] = "PA\_hard"  
  
 # Adjust start and end times accordingly  
 # final\_df = final\_df.groupby(["Sub", "Task"]).agg({"Start": "min", "End": "max"}).reset\_index()  
  
 # Keep only the hour and minute part of the time columns  
 final\_df["Start"] = final\_df["Start"].dt.strftime("%H:%M:%S")  
 final\_df["End"] = final\_df["End"].dt.strftime("%H:%M:%S")  
  
 return final\_df  
  
 def Splite\_vas(self,df):  
 *"""  
 Function to structure VAS data from raw dataframe.  
 Returns a dataframe with columns: Sub, Task, Start, End, Score.  
 """* df["Timestamp"] = pd.to\_datetime(df["Timestamp"])  
 df["Timestamp.1"] = pd.to\_datetime(df["Timestamp.1"])  
  
 vas\_stress = df[["subnum", "Timestamp", "Stress"]].copy()  
 vas\_fatigue = df[["subnum", "Timestamp.1", "Fatigue"]].copy()  
  
  
 # Rename columns  
 vas\_stress.columns = ["Sub", "Start", "Score"]  
 vas\_fatigue.columns = ["Sub", "Start", "Score"]  
  
 # Add Task column  
 vas\_stress["Task"] = "VAS\_Stress"  
 vas\_fatigue["Task"] = "VAS\_Fatigue"  
  
 # Assign End as None (VAS has only one timestamp per measurement)  
 vas\_stress["End"] = None  
 vas\_fatigue["End"] = None  
  
 # Keep only hour and minute part of time  
 vas\_stress["Start"] = vas\_stress["Start"].dt.strftime("%H:%M:%S")  
 vas\_fatigue["Start"] = vas\_fatigue["Start"].dt.strftime("%H:%M:%S")  
 vas\_stress = vas\_stress[["Sub", "Task", "Start", "End", "Score"]]  
 vas\_fatigue = vas\_fatigue[["Sub", "Task", "Start", "End", "Score"]]  
 vas\_fatigue["Score"] = vas\_fatigue["Score"].round(2) # Round fatigue scores to 2 decimal places  
 vas\_stress["Score"] = vas\_stress["Score"].round(2) # Round fatigue scores to 2 decimal places  
  
 return vas\_stress, vas\_fatigue  
  
 def Make\_Trigger\_Table(self,ID,rangeID):  
 # -----------------------------------------------------START-----------------------------------------------------------------  
 pebl\_path = f'{self.path}\\PEBL2'  
 parlist\_path = f'{self.path}\\Participants\\participation management.csv'  
 parlist\_df = pd.read\_csv(parlist\_path, header=0)  
 parlist\_df = parlist\_df.dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 parlist\_df['code'] = pd.to\_numeric(parlist\_df['code'], errors='coerce').astype('Int64')  
 self.parlist\_df=parlist\_df  
 if ID is not None:  
 if rangeID:  
 parlist\_df = parlist\_df[parlist\_df['code'] >= ID]  
 else:  
 parlist\_df = parlist\_df[parlist\_df['code'] == ID]  
 CS\_path = f'{pebl\_path}\\battery\\stroop\\data\\CS-log.csv'  
 PA\_path = f'{pebl\_path}\\battery\\PASAT\\data\\PASAT-log.csv'  
 TC\_path = f'{pebl\_path}\\battery\\twocoladd\\data\\twocoladd-log.csv'  
 VAS\_path = f'{pebl\_path}\\battery\\scales\\data\\VAS-log.csv'  
 BR\_path = f'{pebl\_path}\\battery\\Break-log.csv'  
  
 CS\_df=pd.read\_csv(CS\_path)  
 PA\_df = pd.read\_csv(PA\_path)  
 TC\_df = pd.read\_csv(TC\_path)  
 VAS\_df = pd.read\_csv(VAS\_path)  
 BR\_df = pd.read\_csv(BR\_path)  
  
 CS\_df=self.structure\_data(CS\_df)  
 PA\_df=self.structure\_data(PA\_df)  
 TC\_df=self.structure\_data(TC\_df)  
 BR\_df=self.structure\_data(BR\_df)  
 VAS\_st,VAS\_fa=self.Splite\_vas(VAS\_df)  
  
 for index, row in parlist\_df.iterrows():  
 ID = row['code']  
 print(ID)  
 Group = row['Group']  
 # Task summary files  
 CS\_ID=CS\_df[CS\_df["Sub"]==ID]  
 TC\_ID=TC\_df[TC\_df["Sub"]==ID]  
 PA\_ID=PA\_df[PA\_df["Sub"]==ID]  
 BR\_ID=BR\_df[BR\_df["Sub"]==ID]  
 VAS\_faID=VAS\_fa[VAS\_fa["Sub"]==ID]  
 VAS\_stID=VAS\_st[VAS\_st["Sub"]==ID]  
 combined\_ID = pd.concat([CS\_ID, TC\_ID, PA\_ID,VAS\_faID,VAS\_stID,BR\_ID], ignore\_index=True)  
  
 # -----------------------------------------------------BioPac-Start\_time-----------------------------------------------------------------  
 participant\_path=fr'{self.path}\Participants\{Group}\_group\P\_{ID}'  
 participant\_acq=fr'{participant\_path}\P\_{ID}.acq'  
 participant\_Trigger=fr'{participant\_path}\Trigger\_{ID}.csv'  
 BioPac = bioread.read\_file(participant\_acq)  
 Start\_time = BioPac.event\_markers[00].text  
 Start\_time = pd.to\_datetime(Start\_time)  
 Start\_time = Start\_time.time().strftime('%H:%M:%S')  
 new\_row = {'Sub': ID,  
 'Task': 'Biopac record',  
 'Start': Start\_time,  
 'End': None}  
  
 # Append new row to DataFrame  
 combined\_ID = combined\_ID.\_append(new\_row, ignore\_index=True)  
 combined\_ID = combined\_ID.sort\_values(by='Start')  
 biopac\_time = combined\_ID.loc[combined\_ID["Task"] == "Biopac record", "Start"].min()  
 combined\_ID=combined\_ID[combined\_ID["Start"]>=biopac\_time]  
 combined\_ID = combined\_ID.reset\_index(drop=True)  
 combined\_ID["Start"] = pd.to\_datetime(combined\_ID["Start"], format="%H:%M:%S", errors="coerce").dt.time  
 combined\_ID["End"] = pd.to\_datetime(combined\_ID["End"], format="%H:%M:%S", errors='coerce').dt.time  
 combined\_ID["Start"] = combined\_ID["Start"].apply(lambda x: (pd.to\_datetime(str(x), format="%H:%M:%S") - pd.to\_datetime(str(biopac\_time), format="%H:%M:%S")).total\_seconds())  
 combined\_ID["End"] = combined\_ID["End"].apply(lambda x: (pd.to\_datetime(str(x), format="%H:%M:%S") - pd.to\_datetime(str(biopac\_time), format="%H:%M:%S")).total\_seconds())  
 combined\_ID.to\_csv(participant\_Trigger, index=False)  
  
 else:  
 CS\_ID = CS\_df[CS\_df["Sub"] == ID]  
 TC\_ID = TC\_df[TC\_df["Sub"] == ID]  
 PA\_ID = PA\_df[PA\_df["Sub"] == ID]  
 BR\_ID = BR\_df[BR\_df["Sub"] == ID]  
 VAS\_st = VAS\_st[VAS\_st["Sub"] == ID]  
 VAS\_fa = VAS\_fa[VAS\_fa["Sub"] == ID]  
  
 combined\_ID = pd.concat([CS\_ID, TC\_ID, PA\_ID, VAS\_st, VAS\_fa, BR\_ID], ignore\_index=True)  
  
 # Process BioPac start time  
 participant\_path = fr'{self.path}\Participants\{Group}\_group\P\_{ID}'  
 participant\_acq = fr'{participant\_path}\P\_{ID}.acq'  
 participant\_Trigger = fr'{participant\_path}\Trigger\_{ID}.csv'  
  
 BioPac = bioread.read\_file(participant\_acq)  
 Start\_time = pd.to\_datetime(BioPac.event\_markers[0].text).time().strftime('%H:%M:%S')  
  
 new\_row = {'Sub': ID, 'Task': 'Biopac record', 'Start': Start\_time, 'End': None}  
 combined\_ID = combined\_ID.\_append(new\_row, ignore\_index=True)  
  
 # Processing timestamps  
 combined\_ID = combined\_ID.sort\_values(by='Start')  
 biopac\_time = combined\_ID.loc[combined\_ID["Task"] == "Biopac record", "Start"].min()  
 combined\_ID = combined\_ID[combined\_ID["Start"] >= biopac\_time].reset\_index(drop=True)  
  
 combined\_ID["Start"] = pd.to\_datetime(combined\_ID["Start"], format="%H:%M:%S", errors="coerce").dt.time  
 combined\_ID["End"] = pd.to\_datetime(combined\_ID["End"], format="%H:%M:%S", errors="coerce").dt.time  
  
 combined\_ID["Start"] = combined\_ID["Start"].apply(lambda x: (  
 pd.to\_datetime(str(x), format="%H:%M:%S") - pd.to\_datetime(str(biopac\_time),  
 format="%H:%M:%S")).total\_seconds())  
 combined\_ID["End"] = combined\_ID["End"].apply(lambda x: (  
 pd.to\_datetime(str(x), format="%H:%M:%S") - pd.to\_datetime(str(biopac\_time),  
 format="%H:%M:%S")).total\_seconds())  
  
 combined\_ID.to\_csv(participant\_Trigger, index=False)

import matplotlib.pyplot as plt  
import bioread  
import neurokit2 as nk  
from scipy.signal import spectrogram  
import matplotlib.cm as cm  
import numpy as np  
from scipy.signal import medfilt  
import seaborn as sns  
import os  
from scipy.stats import linregress  
import itertools  
import re  
from pathlib import Path  
import pandas as pd  
  
  
# Press the green button in the gutter to run the script.  
class HumanDataExtraction():  
  
 def \_\_init\_\_(self,Directory):  
 self.path = Directory  
 self.sorted\_DATA = pd.DataFrame()  
  
 def Check\_MedianFilter(self,ID,rangeID):  
 Participants\_path = f'{self.path}\\Participants\\participation management.csv'  
 Participants\_df = pd.read\_csv(Participants\_path, header=0)  
 Participants\_df = Participants\_df.dropna(axis=1, how='all')  
 Participants\_df = Participants\_df.dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
  
 if ID is not None:  
 if rangeID:  
 Participants\_df = Participants\_df[Participants\_df['code'] >= ID]  
 else:  
 Participants\_df = Participants\_df[Participants\_df['code'] == ID]  
 num\_participants = len(Participants\_df)  
  
 medfilt\_window = [3,5,11,101]  
 for window in medfilt\_window:  
 fig, ax = plt.subplots(nrows=num\_participants, figsize=(15, 4 \* num\_participants), squeeze=False)  
 ax = ax.flatten() # להפוך לרשימה חד־ממדית גם אם רק משתתף אחד  
 for idx, (i, row) in enumerate(Participants\_df.iterrows()):  
 ID = row['code']  
 Group = row['Group']  
 directory = fr'{self.path}\Participants\{Group}\_group\P\_{ID}'  
 BioPac\_path = fr'{directory}\P\_{ID}.acq'  
 BioPac = bioread.read\_file(BioPac\_path)  
  
 eda\_data = BioPac.named\_channels['EDA'].data  
 eda\_time = BioPac.named\_channels['EDA'].time\_index  
 EDA\_Clean = medfilt(eda\_data, kernel\_size=window)  
  
 ax[i].plot(eda\_time, eda\_data, label=f'Participant {ID} Raw')  
 ax[i].plot(eda\_time, EDA\_Clean, label=f'Participant {ID} Filtered')  
 ax[i].set\_title(f'Participant {ID} - EDA Signal')  
 ax[i].legend()  
 ax[i].set\_xlabel('Time (s)')  
 ax[i].set\_ylabel('EDA (µS)')  
  
 save\_path = fr'D:\Human Bio Signals Analysis\Preprocessing\MedianFilter\EDA\_data\_{window}.png'  
 plt.savefig(save\_path)  
 def CleanData(self,ID,rangeID,plot=False):  
 Participants\_path = f'{self.path}\Participants\participation management.csv'  
 Participants\_df = pd.read\_csv(Participants\_path, header=0)  
 Participants\_df = Participants\_df.dropna(axis=1, how='all')  
 Participants\_df = Participants\_df.dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
 medfilt\_window=101  
 if ID is not None:  
 if rangeID:  
 Participants\_df = Participants\_df[Participants\_df['code'] >= ID]  
 else:  
 Participants\_df = Participants\_df[Participants\_df['code'] == ID]  
  
 for j, row in Participants\_df.iterrows():  
 ID = row['code']  
 Group = row['Group']  
 print(ID)  
 directory = fr'{self.path}\Participants\{Group}\_group\P\_{ID}'  
 Trigger\_path = fr'{directory}\Trigger\_{ID}.csv'  
 Trigger\_df = pd.read\_csv(Trigger\_path, header=0)  
 BioPac\_path = fr'{directory}\P\_{ID}.acq'  
 BioPac = bioread.read\_file(BioPac\_path)  
  
 eda\_data = BioPac.named\_channels['EDA'].data  
 eda\_time = BioPac.named\_channels['EDA'].time\_index  
 ecg\_data = BioPac.named\_channels['ECG'].data  
 ecg\_time = BioPac.named\_channels['ECG'].time\_index  
 ecg\_sampling\_rate = BioPac.named\_channels['ECG'].samples\_per\_second  
  
 ecg\_cleaned = nk.ecg\_clean(ecg\_data, sampling\_rate=ecg\_sampling\_rate)  
 r\_peaks, \_ = nk.ecg\_peaks(ecg\_cleaned, sampling\_rate=ecg\_sampling\_rate)  
 ecg\_signals, info = nk.ecg\_process(ecg\_cleaned, ecg\_sampling\_rate)  
 hrv\_rri = np.diff(info["ECG\_R\_Peaks"]) / ecg\_sampling\_rate \* 1000  
 time\_rri = info["ECG\_R\_Peaks"][1:] / ecg\_sampling\_rate  
 eda\_signals, info = nk.eda\_process(eda\_data, sampling\_rate=BioPac.named\_channels['EDA'].samples\_per\_second)  
 EDA\_Clean = medfilt(eda\_signals['EDA\_Clean'], kernel\_size=medfilt\_window)  
 EDA\_Clean = pd.DataFrame({  
 'Time': eda\_time,  
 'EDA': EDA\_Clean # adjust column index as needed  
 })  
 EDA\_Clean.to\_csv(fr'{directory}\EDA.csv')  
 RR\_cleaned = pd.DataFrame({  
 'Time': time\_rri,  
 'RR': hrv\_rri # adjust column index as needed  
 })  
 RR = pd.DataFrame({  
 'Time': time\_rri,  
 'RR': hrv\_rri # adjust column index as needed  
 })  
 invalid\_rr = (RR\_cleaned['RR'] < 300) | (RR\_cleaned['RR'] > 1500)  
 # הפיכת הערכים החריגים ל-NaN  
 RR\_cleaned.loc[invalid\_rr, 'RR'] = None  
  
 # אינטרפולציה ליניארית לפי עמודת הזמן  
 # RR\_cleaned['RR'] = RR\_cleaned['RR'].interpolate(method='linear', limit\_direction='both')  
 if plot:  
 fig,ax=plt.subplots(2)  
 ax[0].plot(RR['Time'],RR['RR'])  
 ax[1].plot(RR\_cleaned['Time'],RR\_cleaned['RR'])  
 plt.show()  
 RR\_cleaned.to\_csv(fr'{directory}\RR.csv')  
  
 def BoxPlot(self, ID, Group):  
 Participants\_path = f'{self.path}\\Participants\\participation management.csv'  
 Participants\_df = pd.read\_csv(Participants\_path)  
 Participants\_df = Participants\_df.dropna(axis=1, how='all')  
 Participants\_df = Participants\_df.dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
  
 if ID is not None:  
 Participants\_df = Participants\_df[Participants\_df['code'] == ID]  
  
 for \_, row in Participants\_df.iterrows():  
 ID = row['code']  
 Group = row['Group']  
 directory = fr'{self.path}\\Participants\\{Group}\_group\\P\_{ID}'  
  
 # Load data  
 Trigger\_df = pd.read\_csv(fr'{directory}\\Trigger\_{ID}.csv')  
 EDA\_df = pd.read\_csv(fr'{directory}\\EDA.csv')  
 RR\_df = pd.read\_csv(fr'{directory}\\RR.csv')  
  
 eda\_time = EDA\_df['Time']  
 eda\_signal = EDA\_df['EDA']  
  
 rr\_time = RR\_df['Time']  
 rr\_signal = RR\_df['RR']  
  
 # Ensure plot directory exists  
 plot\_dir = fr'{directory}\\plots'  
 os.makedirs(plot\_dir, exist\_ok=True)  
  
 # --- EDA Boxplot ---  
 eda\_by\_task = []  
  
 for \_, trigger in Trigger\_df.iterrows():  
 task = trigger['Task']  
 start = trigger['Start']  
 end = trigger['End']  
  
 segment = eda\_signal[(eda\_time >= start) & (eda\_time <= end)]  
 eda\_by\_task.extend([{'Task': task, 'EDA': val} for val in segment])  
  
 df\_eda = pd.DataFrame(eda\_by\_task)  
  
 plt.figure(figsize=(10, 6))  
 sns.boxplot(data=df\_eda, x='Task', y='EDA')  
 plt.title(f'Participant {ID} - EDA by Task Phase')  
 plt.xlabel('Task Phase')  
 plt.ylabel('EDA Amplitude')  
 plt.xticks(rotation=45)  
 plt.tight\_layout()  
 plt.savefig(fr'{plot\_dir}\\boxplot\_EDA.png')  
 plt.close()  
  
 # --- RR Boxplot ---  
 rr\_by\_task = []  
  
 for \_, trigger in Trigger\_df.iterrows():  
 task = trigger['Task']  
 start = trigger['Start']  
 end = trigger['End']  
  
 segment = rr\_signal[(rr\_time >= start) & (rr\_time <= end)]  
 rr\_by\_task.extend([{'Task': task, 'RR': val} for val in segment])  
  
 df\_rr = pd.DataFrame(rr\_by\_task)  
  
 plt.figure(figsize=(10, 6))  
 sns.boxplot(data=df\_rr, x='Task', y='RR')  
 plt.title(f'Participant {ID} - RR by Task Phase')  
 plt.xlabel('Task Phase')  
 plt.ylabel('R-R Interval')  
 plt.xticks(rotation=45)  
 plt.tight\_layout()  
 plt.savefig(fr'{plot\_dir}\\boxplot\_RR.png')  
 plt.close()  
 def plot\_with\_annotations(self, data, time\_index, Trigger\_df, feature, Channel, ID, Group, Insert\_Mistake):  
 plt.figure(figsize=(16, 6)) # Wider plot with adjusted dimensions  
 plt.plot(time\_index, data, label=f'{feature}')  
  
 max\_rate = max(data)  
 plt.ylim(0, max\_rate \* 1.5) # Adjust y-axis limit for better label placement  
  
 # Ensure 'Stress Report' column has no NaN values  
 Trigger\_df['Stress Report'] = Trigger\_df['Stress Report'].fillna('')  
  
 for i, row in Trigger\_df.iterrows():  
 start = row['Time-start-sec']  
 end = row['Time-end-sec']  
 task = row['Task']  
 stress\_report = row['Stress Report']  
 mid\_point = (start + end) / 2  
 y\_pos = max\_rate \* (1.1 + 0.1 \* (i % 2)) # Alternates text height for readability  
  
 if task == 'mistake':  
 line\_color = 'red'  
 plt.axvline(x=start, color=line\_color, linestyle='--', alpha=0.7)  
 else:  
 if task in ['Baseline', 'wait\_timer', 'breathing\_part\_1', 'breathing\_part\_2',  
 'breathing\_part\_3', 'breathing\_part\_4', 'music\_part\_1',  
 'music\_part\_2', 'music\_part\_3', 'music\_part\_4']:  
 # plt.axvspan(start, end, color='yellow', alpha=0.3, label='rest')  
 label\_text = f"{task}" # Consistent label structure  
 plt.text(mid\_point, y\_pos, label\_text, ha='center', va='bottom', fontsize=8, color='black',  
 rotation=0)  
 else:  
 # Use distinct colors for higher levels  
 plt.axvspan(start, end, color='orange', alpha=0.8, label=f'{task}')  
 label\_text = f"{task}\nStress:{stress\_report}" # Consistent label structure  
 plt.text(mid\_point, y\_pos, label\_text, ha='center', va='bottom', fontsize=8, color='black',  
 rotation=0)  
  
 # Add labels, title, and legend  
 plt.xlabel("Start Time (s)")  
 plt.ylabel(f'{feature}')  
 plt.title(f'{feature} For Participant {ID} Group {Group}')  
 plt.grid(True)  
 plt.tight\_layout() # Ensures everything fits nicely  
  
 # Save the plot  
 if Insert\_Mistake:  
 plt.savefig(  
 f"D:\\Human Bio Signals Analysis\\Analysis\\{Channel}\\{feature}\\{Group}\\Participant\_{ID}\_{feature}\_include\_mistake.png")  
 else:  
 plt.savefig(  
 f"D:\\Human Bio Signals Analysis\\Analysis\\{Channel}\\{feature}\\{Group}\\Participant\_{ID}\_{feature}.png")  
  
 plt.show()  
 def AX\_Spectrogram(self,ID,Group):  
 Participants\_path = f'{self.path}\\Participants\\participation management.csv'  
 Participants\_df = pd.read\_csv(Participants\_path, header=0)  
 Participants\_df = Participants\_df.dropna(axis=1, how='all')  
 Participants\_df = Participants\_df.dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
  
 if ID is not None:  
 Participants\_df = Participants\_df[Participants\_df['code'] == ID]  
 for \_, row in Participants\_df.iterrows():  
 ID = row['code']  
 Group = row['Group']  
 directory = fr'{self.path}\\Participants\\{Group}\_group\\P\_{ID}'  
 Trigger\_path = fr'{directory}\\Trigger\_{ID}.csv'  
 Trigger\_df = pd.read\_csv(Trigger\_path, header=0)  
 BioPac\_path = fr'{directory}\\P\_{ID}.acq'  
 BioPac = bioread.read\_file(BioPac\_path)  
  
 fig, ax = plt.subplots(2, figsize=(15, 12), sharex=True)  
  
 # Load EDA data  
 eda\_data = BioPac.named\_channels['EDA'].data  
 eda\_time = BioPac.named\_channels['EDA'].time\_index  
 eda\_sampling\_rate = BioPac.named\_channels['EDA'].samples\_per\_second  
  
 # Load ECG data  
 ecg\_data = BioPac.named\_channels['ECG'].data  
 ecg\_time = BioPac.named\_channels['ECG'].time\_index  
 ecg\_sampling\_rate = BioPac.named\_channels['ECG'].samples\_per\_second  
  
 # Process ECG to extract HRV (RR intervals)  
 ecg\_cleaned = nk.ecg\_clean(ecg\_data, sampling\_rate=ecg\_sampling\_rate)  
 r\_peaks, \_ = nk.ecg\_peaks(ecg\_cleaned, sampling\_rate=ecg\_sampling\_rate)  
 ecg\_signals, info = nk.ecg\_process(ecg\_cleaned, ecg\_sampling\_rate)  
 hrv\_rri = np.diff(info["ECG\_R\_Peaks"]) / ecg\_sampling\_rate \* 1000 # Convert to ms  
 time\_rri = info["ECG\_R\_Peaks"][1:] / ecg\_sampling\_rate # Time axis for HRV  
  
 # Process EDA  
 eda\_signals, info = nk.eda\_process(eda\_data, sampling\_rate=eda\_sampling\_rate)  
 eda\_Phasic = eda\_signals["EDA\_Phasic"]  
  
 # Compute spectrogram for HRV  
 f\_hrv, t\_hrv, Sxx\_hrv = spectrogram(hrv\_rri, fs=1 / np.mean(np.diff(time\_rri)), nperseg=20)  
  
 # Compute spectrogram for EDA  
 f\_eda, t\_eda, Sxx\_eda = spectrogram(eda\_data, fs=eda\_sampling\_rate, nperseg=256)  
  
 # Plot HRV spectrogram  
 hrv\_img = ax[0].pcolormesh(t\_hrv, f\_hrv, np.log(Sxx\_hrv), shading='gouraud') # Avoid log(0)  
 ax[0].set\_ylabel("HRV Frequency (Hz)")  
 ax[0].set\_title(f"HRV Spectrogram - Participant {ID}")  
 cbar\_hrv = plt.colorbar(hrv\_img, ax=ax[0])  
 cbar\_hrv.set\_label('Log Power')  
 y\_min\_ax0, y\_max\_ax0 = ax[0].get\_ylim()  
  
 # Plot EDA spectrogram  
 eda\_img = ax[1].pcolormesh(t\_eda, f\_eda, np.log(Sxx\_eda), shading='gouraud') # Avoid log(0)  
 ax[1].set\_ylabel("EDA Frequency (Hz)")  
 ax[1].set\_xlabel("Time (s)")  
 ax[1].set\_title(f"EDA Spectrogram - Participant {ID}")  
 cbar\_eda = plt.colorbar(eda\_img, ax=ax[1])  
 cbar\_eda.set\_label('Log Power')  
 y\_min\_ax1, y\_max\_ax1 = ax[1].get\_ylim()  
  
  
 # Add Task Labels & Event Markers  
 for \_, trigger in Trigger\_df.iterrows():  
 start\_time = trigger["Start"]  
 end\_time = trigger["End"]  
 task\_name = trigger["Task"]  
 if pd.notna(end\_time):  
 if task\_name=='CB\_easy' or task\_name=='CB\_hard':  
 ax[0].axvline(start\_time, color='white', alpha=0.3)  
 ax[0].axvline(end\_time, color='white', alpha=0.3)  
 ax[1].axvline(start\_time, color='white', alpha=0.3)  
 ax[1].axvline(end\_time, color='white', alpha=0.3)  
 ax[0].text((start\_time+end\_time)/2, y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name,  
 ha='center', fontsize=10, color='black')  
 elif task\_name == 'PA\_easy' or task\_name == 'PA\_medium'or task\_name == 'PA\_hard':  
 ax[0].axvline(start\_time, color='white', alpha=0.3)  
 ax[0].axvline(end\_time, color='white', alpha=0.3)  
 ax[1].axvline(start\_time, color='white', alpha=0.3)  
 ax[1].axvline(end\_time, color='white', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2,y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name,  
 ha='center', fontsize=8, color='black')  
 elif task\_name == 'TC\_easy' or task\_name == 'TC\_hard':  
 ax[0].axvline(start\_time, color='white', alpha=0.3)  
 ax[0].axvline(end\_time, color='white', alpha=0.3)  
 ax[1].axvline(start\_time, color='white', alpha=0.3)  
 ax[1].axvline(end\_time, color='white', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2,y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name,  
 ha='center', fontsize=8, color='black')  
 else:  
 ax[0].axvline(start\_time, color='white', alpha=0.3)  
 ax[0].axvline(end\_time, color='white', alpha=0.3)  
 ax[1].axvline(start\_time, color='white', alpha=0.3)  
 ax[1].axvline(end\_time, color='white', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2,y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name,  
 ha='center', fontsize=10, color='black')  
  
 plt.tight\_layout()  
 plot\_path = fr'{directory}\\plots\\Spectrogram\_Plot\_{ID}.png'  
 plt.savefig(plot\_path, dpi=300)  
 plt.show()  
 def AX\_Spectrogram\_4\_signals(self,ID,Group):  
 Participants\_path = f'{self.path}\\Participants\\participation management.csv'  
 Participants\_df = pd.read\_csv(Participants\_path, header=0)  
 Participants\_df = Participants\_df.dropna(axis=1, how='all')  
 Participants\_df = Participants\_df.dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
  
 if ID is not None:  
 Participants\_df = Participants\_df[Participants\_df['code'] == ID]  
 for \_, row in Participants\_df.iterrows():  
 ID = row['code']  
 Group = row['Group']  
 directory = fr'{self.path}\\Participants\\{Group}\_group\\P\_{ID}'  
 Trigger\_path = fr'{directory}\\Trigger\_{ID}.csv'  
 Trigger\_df = pd.read\_csv(Trigger\_path, header=0)  
 BioPac\_path = fr'{directory}\\P\_{ID}.acq'  
 BioPac = bioread.read\_file(BioPac\_path)  
  
 fig, ax = plt.subplots(4, figsize=(15, 12), sharex=True)  
 # Load EDA data  
 eda\_data = BioPac.named\_channels['EDA'].data  
 eda\_time = BioPac.named\_channels['EDA'].time\_index  
 eda\_sampling\_rate = BioPac.named\_channels['EDA'].samples\_per\_second  
 # eda\_signals, info = nk.eda\_process(eda\_data, sampling\_rate=eda\_sampling\_rate)  
 # eda\_data = eda\_signals['EDA\_Clean']  
  
 f\_eda, t\_eda, Sxx\_eda = spectrogram(eda\_data, fs=eda\_sampling\_rate, nperseg=256)  
  
 # Plot EDA spectrogram  
 eda\_img = ax[0].pcolormesh(t\_eda, f\_eda, np.log(Sxx\_eda), shading='gouraud')  
 ax[0].set\_ylabel("EDA Frequency (Hz)")  
 ax[0].set\_title(f"EDA Spectrogram - Participant {ID}")  
 # cbar\_eda = plt.colorbar(eda\_img, ax=ax[0])  
 # cbar\_eda.set\_label('Log Power')  
 y\_min\_ax0, y\_max\_ax0 = ax[0].get\_ylim()  
  
 ax[1].plot(eda\_time, eda\_data, label=f"Participant {ID}")  
  
 # Load ECG data  
 ecg\_data = BioPac.named\_channels['ECG'].data  
 ecg\_time = BioPac.named\_channels['ECG'].time\_index  
 ecg\_sampling\_rate = BioPac.named\_channels['ECG'].samples\_per\_second  
  
 # Process ECG to extract HRV (RR intervals)  
 ecg\_cleaned = nk.ecg\_clean(ecg\_data, sampling\_rate=ecg\_sampling\_rate)  
 r\_peaks, \_ = nk.ecg\_peaks(ecg\_cleaned, sampling\_rate=ecg\_sampling\_rate)  
 ecg\_signals, info = nk.ecg\_process(ecg\_cleaned, ecg\_sampling\_rate)  
 hrv\_rri = np.diff(info["ECG\_R\_Peaks"]) / ecg\_sampling\_rate \* 1000 # Convert to ms  
 time\_rri = info["ECG\_R\_Peaks"][1:] / ecg\_sampling\_rate # Time axis for HRV  
  
 f\_hrv, t\_hrv, Sxx\_hrv = spectrogram(hrv\_rri, fs=1 / np.mean(np.diff(time\_rri)), nperseg=20)  
  
 hrv\_img = ax[2].pcolormesh(t\_hrv, f\_hrv, np.log(Sxx\_hrv), shading='gouraud') # Avoid log(0)  
 ax[2].set\_ylabel("HRV Frequency (Hz)")  
 ax[2].set\_title(f"HRV Spectrogram - Participant {ID}")  
 # cbar\_hrv = plt.colorbar(hrv\_img, ax=ax[0])  
 # cbar\_hrv.set\_label('Log Power')  
  
 ax[3].plot(time\_rri, hrv\_rri, label=f"Participant {ID}")  
  
 # Add Task Labels & Event Markers  
 for \_, trigger in Trigger\_df.iterrows():  
 start\_time = trigger["Start"]  
 end\_time = trigger["End"]  
 task\_name = trigger["Task"]  
 if pd.notna(end\_time):  
 if task\_name=='CB\_easy' or task\_name=='CB\_hard':  
 ax[0].axvline(start\_time, color='white', alpha=0.3)  
 ax[0].axvline(end\_time, color='white', alpha=0.3)  
 ax[1].axvline(start\_time, color='purple', alpha=0.3)  
 ax[1].axvline(end\_time, color='purple', alpha=0.3)  
 ax[2].axvline(start\_time, color='white', alpha=0.3)  
 ax[2].axvline(end\_time, color='white', alpha=0.3)  
 ax[3].axvline(start\_time, color='purple', alpha=0.3)  
 ax[3].axvline(end\_time, color='purple', alpha=0.3)  
 ax[0].text((start\_time+end\_time)/2, y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name, ha='center', fontsize=10, color='black')  
 elif task\_name == 'PA\_easy' or task\_name == 'PA\_medium'or task\_name == 'PA\_hard':  
 ax[0].axvline(start\_time, color='white', alpha=0.3)  
 ax[0].axvline(end\_time, color='white', alpha=0.3)  
 ax[1].axvline(start\_time, color='purple', alpha=0.3)  
 ax[1].axvline(end\_time, color='purple', alpha=0.3)  
 ax[2].axvline(start\_time, color='white', alpha=0.3)  
 ax[2].axvline(end\_time, color='white', alpha=0.3)  
 ax[3].axvline(start\_time, color='purple', alpha=0.3)  
 ax[3].axvline(end\_time, color='purple', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2,y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name, ha='center', fontsize=8, color='black')  
 elif task\_name == 'TC\_easy' or task\_name == 'TC\_hard':  
 ax[0].axvline(start\_time, color='white', alpha=0.3)  
 ax[0].axvline(end\_time, color='white', alpha=0.3)  
 ax[1].axvline(start\_time, color='purple', alpha=0.3)  
 ax[1].axvline(end\_time, color='purple', alpha=0.3)  
 ax[2].axvline(start\_time, color='white', alpha=0.3)  
 ax[2].axvline(end\_time, color='white', alpha=0.3)  
 ax[3].axvline(start\_time, color='purple', alpha=0.3)  
 ax[3].axvline(end\_time, color='purple', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2,y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name, ha='center', fontsize=8, color='black')  
 else:  
 ax[0].axvline(start\_time, color='white', alpha=0.3)  
 ax[0].axvline(end\_time, color='white', alpha=0.3)  
 ax[1].axvline(start\_time, color='purple', alpha=0.3)  
 ax[1].axvline(end\_time, color='purple', alpha=0.3)  
 ax[2].axvline(start\_time, color='white', alpha=0.3)  
 ax[2].axvline(end\_time, color='white', alpha=0.3)  
 ax[3].axvline(start\_time, color='purple', alpha=0.3)  
 ax[3].axvline(end\_time, color='purple', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2,y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name, ha='center', fontsize=10, color='black')  
  
 plt.tight\_layout()  
 plot\_path = fr'{directory}\\plots\\Spectrogram\_4\_signals\_Plot\_{ID}.png'  
 plt.savefig(plot\_path, dpi=300)  
 plt.show()  
 def AX\_Spectrogram\_signals(self,ID,Group):  
 Participants\_path = f'{self.path}\\Participants\\participation management.csv'  
 Participants\_df = pd.read\_csv(Participants\_path, header=0)  
 Participants\_df = Participants\_df.dropna(axis=1, how='all')  
 Participants\_df = Participants\_df.dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
  
 if ID is not None:  
 Participants\_df = Participants\_df[Participants\_df['code'] == ID]  
 for \_, row in Participants\_df.iterrows():  
 ID = row['code']  
 Group = row['Group']  
 directory = fr'{self.path}\\Participants\\{Group}\_group\\P\_{ID}'  
 Trigger\_path = fr'{directory}\\Trigger\_{ID}.csv'  
 Trigger\_df = pd.read\_csv(Trigger\_path, header=0)  
 BioPac\_path = fr'{directory}\\P\_{ID}.acq'  
 BioPac = bioread.read\_file(BioPac\_path)  
  
 fig, ax = plt.subplots(2, figsize=(15, 12), sharex=True)  
 Channels = ['EDA', 'HRV']  
 for channel in Channels:  
 if channel=='EDA':  
 # Load EDA data  
 eda\_data = BioPac.named\_channels['EDA'].data  
 eda\_time = BioPac.named\_channels['EDA'].time\_index  
 eda\_sampling\_rate = BioPac.named\_channels['EDA'].samples\_per\_second  
 eda\_signals, info = nk.eda\_process(eda\_data, sampling\_rate=eda\_sampling\_rate)  
 eda\_data = eda\_signals['EDA\_Clean']  
  
 f\_eda, t\_eda, Sxx\_eda = spectrogram(eda\_data, fs=eda\_sampling\_rate, nperseg=256)  
 # Plot EDA spectrogram  
 eda\_img = ax[0].pcolormesh(t\_eda, f\_eda, np.log(Sxx\_eda), shading='gouraud')  
 ax[0].set\_ylabel("EDA Frequency (Hz)")  
 ax[0].set\_xlabel("Time (s)")  
 ax[0].set\_title(f"EDA Spectrogram - Participant {ID}")  
 # cbar\_eda = plt.colorbar(eda\_img, ax=ax[0])  
 # cbar\_eda.set\_label('Log Power')  
 y\_min\_ax0, y\_max\_ax0 = ax[0].get\_ylim()  
  
 ax[1].plot(eda\_time, eda\_data, label=f"Participant {ID}")  
  
 if channel=='HRV':  
 # Load ECG data  
 ecg\_data = BioPac.named\_channels['ECG'].data  
 ecg\_time = BioPac.named\_channels['ECG'].time\_index  
 ecg\_sampling\_rate = BioPac.named\_channels['ECG'].samples\_per\_second  
  
 # Process ECG to extract HRV (RR intervals)  
 ecg\_cleaned = nk.ecg\_clean(ecg\_data, sampling\_rate=ecg\_sampling\_rate)  
 r\_peaks, \_ = nk.ecg\_peaks(ecg\_cleaned, sampling\_rate=ecg\_sampling\_rate)  
 ecg\_signals, info = nk.ecg\_process(ecg\_cleaned, ecg\_sampling\_rate)  
 hrv\_rri = np.diff(info["ECG\_R\_Peaks"]) / ecg\_sampling\_rate \* 1000 # Convert to ms  
 time\_rri = info["ECG\_R\_Peaks"][1:] / ecg\_sampling\_rate # Time axis for HRV  
 # Compute spectrogram for HRV  
 f\_hrv, t\_hrv, Sxx\_hrv = spectrogram(hrv\_rri, fs=1 / np.mean(np.diff(time\_rri)), nperseg=20)  
  
 # Plot HRV spectrogram  
 hrv\_img = ax[0].pcolormesh(t\_hrv, f\_hrv, np.log(Sxx\_hrv), shading='gouraud') # Avoid log(0)  
 ax[0].set\_ylabel("HRV Frequency (Hz)")  
 ax[0].set\_title(f"HRV Spectrogram - Participant {ID}")  
 cbar\_hrv = plt.colorbar(hrv\_img, ax=ax[0])  
 cbar\_hrv.set\_label('Log Power')  
 y\_min\_ax0, y\_max\_ax0 = ax[0].get\_ylim()  
  
 ax[1].plot(time\_rri, hrv\_rri, label=f"Participant {ID}")  
  
 # Add Task Labels & Event Markers  
 for \_, trigger in Trigger\_df.iterrows():  
 start\_time = trigger["Start"]  
 end\_time = trigger["End"]  
 task\_name = trigger["Task"]  
 if pd.notna(end\_time):  
 if task\_name=='CB\_easy' or task\_name=='CB\_hard':  
 # ax[0].axvline(start\_time, color='white', alpha=0.3)  
 # ax[0].axvline(end\_time, color='white', alpha=0.3)  
 ax[1].axvline(start\_time, color='purple', alpha=0.3)  
 ax[1].axvline(end\_time, color='purple', alpha=0.3)  
 ax[0].text((start\_time+end\_time)/2, y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name,  
 ha='center', fontsize=10, color='black')  
 elif task\_name == 'PA\_easy' or task\_name == 'PA\_medium'or task\_name == 'PA\_hard':  
 # ax[0].axvline(start\_time, color='white', alpha=0.3)  
 # ax[0].axvline(end\_time, color='white', alpha=0.3)  
 ax[1].axvline(start\_time, color='purple', alpha=0.3)  
 ax[1].axvline(end\_time, color='purple', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2,y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name,  
 ha='center', fontsize=8, color='black')  
 elif task\_name == 'TC\_easy' or task\_name == 'TC\_hard':  
 # ax[0].axvline(start\_time, color='white', alpha=0.3)  
 # ax[0].axvline(end\_time, color='white', alpha=0.3)  
 ax[1].axvline(start\_time, color='purple', alpha=0.3)  
 ax[1].axvline(end\_time, color='purple', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2,y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name,  
 ha='center', fontsize=8, color='black')  
 else:  
 # ax[0].axvline(start\_time, color='white', alpha=0.3)  
 # ax[0].axvline(end\_time, color='white', alpha=0.3)  
 ax[1].axvline(start\_time, color='purple', alpha=0.3)  
 ax[1].axvline(end\_time, color='purple', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2,y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name,  
 ha='center', fontsize=10, color='black')  
  
 plt.tight\_layout()  
 plot\_path = fr'{directory}\\plots\\Spectrogram\_{channel}\_Plot\_{ID}.png'  
 plt.savefig(plot\_path, dpi=300)  
 plt.show()  
 def AX\_plot\_3\_part\_HRV(self,ID):  
 # Load participant management data  
 Participants\_path = f'{self.path}\\Participants\\participation management.csv'  
 Participants\_df = pd.read\_csv(Participants\_path, header=0)  
 Participants\_df = Participants\_df.dropna(axis=1, how='all')  
 Participants\_df = Participants\_df.dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
  
 if ID is not None:  
 if isinstance(ID, list): # Handle multiple IDs  
 Participants\_df = Participants\_df[Participants\_df['code'].isin(ID)]  
 else: # Handle single ID  
 Participants\_df = Participants\_df[Participants\_df['code'] == ID]  
  
 fig, ax = plt.subplots(3, figsize=(15, 12), sharex=True)  
 fig.suptitle(f'HRV for particapents {ID}')  
  
 # Loop through participants  
 for i, row in Participants\_df.iterrows():  
 participant\_id = row['code']  
 participant\_group = row['Group']  
  
 # Define paths  
 directory = fr'{self.path}\\Participants\\{participant\_group}\_group\\P\_{participant\_id}'  
 Trigger\_path = fr'{directory}\\Trigger\_{participant\_id}.csv'  
 BioPac\_path = fr'{directory}\\P\_{participant\_id}.acq'  
  
  
 # Load data  
 Trigger\_df = pd.read\_csv(Trigger\_path, header=0)  
 BioPac = bioread.read\_file(BioPac\_path)  
  
 ecg\_data = BioPac.named\_channels['ECG'].data  
 ecg\_time = BioPac.named\_channels['ECG'].time\_index  
 ecg\_sampling\_rate = BioPac.named\_channels['ECG'].samples\_per\_second  
  
 # Process ECG  
 ecg\_cleaned = nk.ecg\_clean(ecg\_data, sampling\_rate=ecg\_sampling\_rate)  
 \_, r\_peaks = nk.ecg\_peaks(ecg\_cleaned, sampling\_rate=ecg\_sampling\_rate)  
 ecg\_signals, info = nk.ecg\_process(ecg\_cleaned, ecg\_sampling\_rate)  
 hrv\_rri = np.diff(info["ECG\_R\_Peaks"]) / ecg\_sampling\_rate \* 1000  
 time\_rri = info["ECG\_R\_Peaks"][1:] / ecg\_sampling\_rate  
  
 # Plot EDA and HRV  
 ax[i].plot(time\_rri, hrv\_rri, label=f"Participant {participant\_id}")  
 ax[i].legend(loc='upper right')  
  
 y\_min\_ax0, y\_max\_ax0 = ax[0].get\_ylim()  
 # Annotate tasks  
 for \_, trigger in Trigger\_df.iterrows():  
 start\_time, end\_time, task\_name, task\_score = trigger["Start"], trigger["End"], trigger["Task"], \  
 trigger["Score"]  
 if pd.notna(end\_time):  
 if task\_name == 'CB\_easy' or task\_name == 'CB\_hard':  
 ax[i].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 if i==0:  
 ax[0].text((start\_time + end\_time) / 2, y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name, ha='center', fontsize=8, color='black')  
 elif task\_name == 'PA\_easy' or task\_name == 'PA\_medium' or task\_name == 'PA\_hard':  
 ax[i].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 if i==0:  
 ax[0].text((start\_time + end\_time) / 2, y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name, ha='center', fontsize=8, color='black')  
 elif task\_name == 'TC\_easy' or task\_name == 'TC\_hard':  
 ax[i].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 if i==0:  
 ax[0].text((start\_time + end\_time) / 2, y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name, ha='center', fontsize=8, color='black')  
 else:  
 ax[i].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 if i==0:  
 ax[0].text((start\_time + end\_time) / 2, y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name, ha='center', fontsize=8, color='black')  
 else:  
 ax[i].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 plt.tight\_layout()  
 plot\_path = fr'{self.path}\Participants\plots\_data\HRV\_comparision.png'  
 plt.savefig(plot\_path, dpi=300)  
 plt.show()  
 def AX\_plot\_3in1norm\_EDA(self, ID):  
 # Load participant management data  
 Participants\_path = f'{self.path}\\Participants\\participation management.csv'  
 Participants\_df = pd.read\_csv(Participants\_path, header=0).dropna(axis=1, how='all')  
 Participants\_df = Participants\_df.dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
  
 if ID is not None:  
 Participants\_df = Participants\_df[Participants\_df['code'].isin(ID)] if isinstance(ID, list) else \  
 Participants\_df[Participants\_df['code'] == ID]  
  
 plt.figure(figsize=(12, 6))  
 plt.title(f'EDA for participants {ID}')  
  
 colors = cm.viridis(np.linspace(0, 1, len(Participants\_df))) # Dynamic color map  
  
 for i, row in Participants\_df.iterrows():  
 participant\_id, participant\_group = row['code'], row['Group']  
 directory = fr'{self.path}\\Participants\\{participant\_group}\_group\\P\_{participant\_id}'  
 Trigger\_path, BioPac\_path = fr'{directory}\\Trigger\_{participant\_id}.csv', fr'{directory}\\P\_{participant\_id}.acq'  
  
 # Load Data  
 Trigger\_df = pd.read\_csv(Trigger\_path, header=0)  
 BioPac = bioread.read\_file(BioPac\_path)  
  
 # Extract EDA signals  
 eda\_data = BioPac.named\_channels['EDA'].data  
 eda\_sampling\_rate = BioPac.named\_channels['EDA'].samples\_per\_second  
 eda\_signals, \_ = nk.eda\_process(eda\_data, sampling\_rate=eda\_sampling\_rate)  
 eda\_cleaned = eda\_signals['EDA\_Clean']  
 # eda\_cleaned = (eda\_cleaned - np.min(eda\_cleaned)) / (np.max(eda\_cleaned) - np.min(eda\_cleaned))  
 eda\_cleaned = (eda\_cleaned - np.mean(eda\_cleaned)) / np.std(eda\_cleaned)  
 eda\_time = np.arange(len(eda\_cleaned)) / eda\_sampling\_rate # Ensure correct time scaling  
  
 # Plot EDA  
 plt.plot(eda\_time, eda\_cleaned, label=f"Participant {participant\_id}", color=colors[i])  
  
 y\_min\_ax0, y\_max\_ax0 = plt.gca().get\_ylim()  
  
 # Annotate tasks  
 for \_, trigger in Trigger\_df.iterrows():  
 start\_time, end\_time, task\_name = trigger["Start"], trigger["End"], trigger["Task"]  
 if pd.notna(end\_time):  
 plt.axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 if i == 0: # Only label once  
 text\_y = max(y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), y\_min\_ax0 \* 0.95)  
 plt.text((start\_time + end\_time) / 2, text\_y, task\_name, ha='center', fontsize=8, color='black')  
  
 plt.legend(loc='upper right')  
 plt.tight\_layout()  
 plot\_path = fr'{self.path}\Participants\plots\_data\EDA\_3in1norm\_comparision.png'  
 plt.savefig(plot\_path, dpi=300)  
 plt.show()  
 def AX\_plot\_3in1\_EDA(self, ID):  
 # Load participant management data  
 Participants\_path = f'{self.path}\\Participants\\participation management.csv'  
 Participants\_df = pd.read\_csv(Participants\_path, header=0).dropna(axis=1, how='all')  
 Participants\_df = Participants\_df.dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
  
 if ID is not None:  
 Participants\_df = Participants\_df[Participants\_df['code'].isin(ID)] if isinstance(ID, list) else \  
 Participants\_df[Participants\_df['code'] == ID]  
  
 plt.figure(figsize=(12, 6))  
 plt.title(f'EDA for participants {ID}')  
  
 colors = cm.viridis(np.linspace(0, 1, len(Participants\_df))) # Dynamic color map  
  
 for i, row in Participants\_df.iterrows():  
 participant\_id, participant\_group = row['code'], row['Group']  
 directory = fr'{self.path}\\Participants\\{participant\_group}\_group\\P\_{participant\_id}'  
 Trigger\_path, BioPac\_path = fr'{directory}\\Trigger\_{participant\_id}.csv', fr'{directory}\\P\_{participant\_id}.acq'  
  
 # Load Data  
 Trigger\_df = pd.read\_csv(Trigger\_path, header=0)  
 BioPac = bioread.read\_file(BioPac\_path)  
  
 # Extract EDA signals  
 eda\_data = BioPac.named\_channels['EDA'].data  
 eda\_sampling\_rate = BioPac.named\_channels['EDA'].samples\_per\_second  
 eda\_signals, \_ = nk.eda\_process(eda\_data, sampling\_rate=eda\_sampling\_rate)  
 eda\_cleaned = eda\_signals['EDA\_Clean']  
 eda\_time = np.arange(len(eda\_cleaned)) / eda\_sampling\_rate # Ensure correct time scaling  
  
 # Plot EDA  
 plt.plot(eda\_time, eda\_cleaned, label=f"Participant {participant\_id}", color=colors[i])  
  
 y\_min\_ax0, y\_max\_ax0 = plt.gca().get\_ylim()  
  
 # Annotate tasks  
 for \_, trigger in Trigger\_df.iterrows():  
 start\_time, end\_time, task\_name = trigger["Start"], trigger["End"], trigger["Task"]  
 if pd.notna(end\_time):  
 plt.axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 if i == 0: # Only label once  
 text\_y = max(y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), y\_min\_ax0 \* 0.95)  
 plt.text((start\_time + end\_time) / 2, text\_y, task\_name, ha='center', fontsize=8, color='black')  
  
 plt.legend(loc='upper right')  
 plt.tight\_layout()  
 plot\_path = fr'{self.path}\Participants\plots\_data\EDA\_3in1\_comparision.png'  
 plt.savefig(plot\_path, dpi=300)  
 plt.show()  
 def AX\_plot\_3\_part\_EDA(self,ID):  
 # Load participant management data  
 Participants\_path = f'{self.path}\\Participants\\participation management.csv'  
 Participants\_df = pd.read\_csv(Participants\_path, header=0)  
 Participants\_df = Participants\_df.dropna(axis=1, how='all')  
 Participants\_df = Participants\_df.dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
  
 if ID is not None:  
 if isinstance(ID, list): # Handle multiple IDs  
 Participants\_df = Participants\_df[Participants\_df['code'].isin(ID)]  
 else: # Handle single ID  
 Participants\_df = Participants\_df[Participants\_df['code'] == ID]  
  
 fig, ax = plt.subplots(3, figsize=(15, 12), sharex=True)  
 fig.suptitle(f'EDA for particapents {ID}')  
  
 # Loop through participants  
 for i, row in Participants\_df.iterrows():  
 participant\_id = row['code']  
 participant\_group = row['Group']  
  
 # Define paths  
 directory = fr'{self.path}\\Participants\\{participant\_group}\_group\\P\_{participant\_id}'  
 Trigger\_path = fr'{directory}\\Trigger\_{participant\_id}.csv'  
 BioPac\_path = fr'{directory}\\P\_{participant\_id}.acq'  
  
  
 # Load data  
 Trigger\_df = pd.read\_csv(Trigger\_path, header=0)  
 BioPac = bioread.read\_file(BioPac\_path)  
  
 # Create plots  
 eda\_data = BioPac.named\_channels['EDA'].data  
 print(len(eda\_data))  
 eda\_sampling\_rate = BioPac.named\_channels['EDA'].samples\_per\_second  
 eda\_signals, info = nk.eda\_process(eda\_data, sampling\_rate=eda\_sampling\_rate)  
 eda\_data=eda\_signals['EDA\_Clean']  
 eda\_time = BioPac.named\_channels['EDA'].time\_index  
 # Plot EDA and HRV  
 ax[i].plot(eda\_time, eda\_data, label=f"Participant {participant\_id}")  
 ax[i].legend(loc='upper right')  
  
 y\_min\_ax0, y\_max\_ax0 = ax[0].get\_ylim()  
 # Annotate tasks  
 for \_, trigger in Trigger\_df.iterrows():  
 start\_time, end\_time, task\_name, task\_score = trigger["Start"], trigger["End"], trigger["Task"], \  
 trigger["Score"]  
 if pd.notna(end\_time):  
 if task\_name == 'CB\_easy' or task\_name == 'CB\_hard':  
 ax[i].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 if i==0:  
 ax[0].text((start\_time + end\_time) / 2, y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name, ha='center', fontsize=8, color='black')  
 elif task\_name == 'PA\_easy' or task\_name == 'PA\_medium' or task\_name == 'PA\_hard':  
 ax[i].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 if i==0:  
 ax[0].text((start\_time + end\_time) / 2, y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name, ha='center', fontsize=8, color='black')  
 elif task\_name == 'TC\_easy' or task\_name == 'TC\_hard':  
 ax[i].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 if i==0:  
 ax[0].text((start\_time + end\_time) / 2, y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name, ha='center', fontsize=8, color='black')  
 else:  
 ax[i].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 if i==0:  
 ax[0].text((start\_time + end\_time) / 2, y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name, ha='center', fontsize=8, color='black')  
 else:  
 ax[i].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 plt.tight\_layout()  
 plot\_path = fr'{self.path}\Participants\plots\_data\EDA\_intervals\_comparision.png'  
 plt.savefig(plot\_path, dpi=300)  
 plt.show()  
 def AX\_plot\_signals(self,ID,rangeID):  
 # Load participant management data  
 Participants\_path = f'{self.path}\\Participants\\participation management.csv'  
 Participants\_df = pd.read\_csv(Participants\_path, header=0)  
 Participants\_df = Participants\_df.dropna(axis=1, how='all')  
 Participants\_df = Participants\_df.dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
  
 if ID is not None:  
 if rangeID:  
 Participants\_df = Participants\_df[Participants\_df['code'] >= ID]  
 else:  
 Participants\_df = Participants\_df[Participants\_df['code'] == ID]  
  
 # Loop through participants  
 for \_, row in Participants\_df.iterrows():  
 participant\_id = row['code']  
 participant\_group = row['Group']  
  
 # Define paths  
 directory = fr'{self.path}\\Participants\\{participant\_group}\_group\\P\_{participant\_id}'  
 Trigger\_path = fr'{directory}\\Trigger\_{participant\_id}.csv'  
 BioPac\_path = fr'{directory}\\P\_{participant\_id}.acq'  
  
 try:  
 # Load data  
 Trigger\_df = pd.read\_csv(Trigger\_path, header=0)  
 BioPac = bioread.read\_file(BioPac\_path)  
  
 # Create plots  
 fig, ax = plt.subplots(2, figsize=(15, 12), sharex=True)  
 eda\_data = BioPac.named\_channels['EDA'].data  
 eda\_sampling\_rate = BioPac.named\_channels['EDA'].samples\_per\_second  
 eda\_signals, info = nk.eda\_process(eda\_data, sampling\_rate=eda\_sampling\_rate)  
 eda\_data=eda\_signals['EDA\_Clean']  
 eda\_time = BioPac.named\_channels['EDA'].time\_index  
 ecg\_data = BioPac.named\_channels['ECG'].data  
 ecg\_time = BioPac.named\_channels['ECG'].time\_index  
 ecg\_sampling\_rate = BioPac.named\_channels['ECG'].samples\_per\_second  
  
 # Process ECG  
 ecg\_cleaned = nk.ecg\_clean(ecg\_data, sampling\_rate=ecg\_sampling\_rate)  
 \_, r\_peaks = nk.ecg\_peaks(ecg\_cleaned, sampling\_rate=ecg\_sampling\_rate)  
 ecg\_signals, info = nk.ecg\_process(ecg\_cleaned, ecg\_sampling\_rate)  
 hrv\_rri = np.diff(info["ECG\_R\_Peaks"]) / ecg\_sampling\_rate \* 1000  
 time\_rri = info["ECG\_R\_Peaks"][1:] / ecg\_sampling\_rate  
  
 # Plot EDA and HRV  
 ax[0].plot(eda\_time, eda\_data, label=f"Participant {participant\_id}")  
 ax[1].plot(time\_rri, hrv\_rri, label=f"Participant {participant\_id}")  
 y\_min\_ax0, y\_max\_ax0 = ax[0].get\_ylim()  
  
 # Annotate tasks  
 for \_, trigger in Trigger\_df.iterrows():  
 start\_time, end\_time, task\_name, task\_score = trigger["Start"], trigger["End"], trigger["Task"], \  
 trigger["Score"]  
 if pd.notna(end\_time):  
 if task\_name == 'CB\_easy' or task\_name == 'CB\_hard':  
 ax[0].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[1].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2, y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name,  
 ha='center', fontsize=10, color='black')  
 elif task\_name == 'PA\_easy' or task\_name == 'PA\_medium' or task\_name == 'PA\_hard':  
 ax[0].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[1].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2, y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name,  
 ha='center', fontsize=8, color='black')  
 elif task\_name == 'TC\_easy' or task\_name == 'TC\_hard':  
 ax[0].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[1].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2, y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name,  
 ha='center', fontsize=8, color='black')  
 else:  
 ax[0].axvspan(start\_time, end\_time, color='#C3E6CB', alpha=0.3)  
 ax[1].axvspan(start\_time, end\_time, color='#C3E6CB', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2, y\_min\_ax0 - 0.05 \* (y\_max\_ax0 - y\_min\_ax0), task\_name,  
 ha='center', fontsize=10, color='black')  
 else:  
 ax[0].axvline(start\_time, color='green', linestyle='--')  
 ax[1].axvline(start\_time, color='green', linestyle='--')  
  
 ax[0].set\_ylabel("EDA Signal")  
 ax[0].legend(loc="upper right", fontsize=8)  
 ax[1].set\_ylabel("ECG RR intervals")  
 ax[1].legend(loc="upper right", fontsize=8)  
 plt.tight\_layout()  
 plot\_path = fr'{directory}\plots\EDA\_RR\_intervals\_{ID}.png'  
 plt.savefig(plot\_path, dpi=300)  
 plt.show()  
  
 except Exception as e:  
 print(f"Error processing participant {participant\_id}: {e}")  
 # def Cor\_plot(self,Window,Overlap,ID, Group)  
 # Participants\_path = f'{self.path}\\Participants\\participation management.csv'  
 # Participants\_df = pd.read\_csv(Participants\_path, header=0)  
 # Participants\_df = Participants\_df.dropna(axis=1, how='all')  
 # Participants\_df = Participants\_df.dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 # Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
 # if ID is not None:  
 # Participants\_df = Participants\_df[Participants\_df['code'] == ID]  
 #  
 # for j, row in Participants\_df.iterrows():  
 # ID = row['code']  
 # print(ID)  
 # Group = row['Group']  
 # directory = fr'{self.path}\\Participants\\{Group}\_group\\P\_{ID}'  
 # Trigger\_path = fr'{directory}\\Trigger\_{ID}.csv'  
 # Trigger\_df = pd.read\_csv(Trigger\_path, header=0)  
 # BioPac\_path = fr'{directory}\\P\_{ID}.acq'  
 # BioPac = bioread.read\_file(BioPac\_path)  
 # EDA = pd.read\_csv(fr'{directory}\\EDA.csv')  
 # RR = pd.read\_csv(fr'{directory}\\RR.csv')  
 # RSP\_D = BioPac.named\_channels['Diaphragmatic Respiration'].data  
 # RSP\_D\_T = BioPac.named\_channels['Diaphragmatic Respiration'].time\_index  
 # RSP\_C = BioPac.named\_channels['Chest Respiration'].data  
 # RSP\_C\_T = BioPac.named\_channels['Chest Respiration'].time\_index  
 # eda\_data = EDA['EDA']  
 def AX\_plot\_signals\_VAS(self, ID, rangeID):  
 VAS\_plot=False  
 Cor\_plot=True  
 Participants\_path = f'{self.path}\\Participants\\participation management.csv'  
 Participants\_df = pd.read\_csv(Participants\_path, header=0)  
 Participants\_df = Participants\_df.dropna(axis=1, how='all')  
 Participants\_df = Participants\_df.dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
 if ID is not None:  
 if rangeID:  
 Participants\_df = Participants\_df[Participants\_df['code'] >= ID]  
 else:  
 Participants\_df = Participants\_df[Participants\_df['code'] == ID]  
 stress\_all = pd.DataFrame(columns=['ID', 'Group', 'HRV\_MeanNN', 'HRV\_SDNN', 'HRV\_RMSSD', 'HRV\_CVNN', 'HRV\_pNN20', 'HRV\_pNN50','Stress'])  
 fatigue\_all= pd.DataFrame(columns=['ID', 'Group', 'HRV\_MeanNN', 'HRV\_SDNN', 'HRV\_RMSSD', 'HRV\_CVNN', 'HRV\_pNN20', 'HRV\_pNN50','Fatigue'])  
 all\_corr\_records = [] # every (ID, Group, Target, Feature, r, p)  
 for j, row in Participants\_df.iterrows():  
 ID = row['code']  
 Group = row['Group']  
 directory = fr'{self.path}\\Participants\\{Group}\_group\\P\_{ID}'  
 Trigger\_path = fr'{directory}\\Trigger\_{ID}.csv'  
 Trigger\_df = pd.read\_csv(Trigger\_path, header=0)  
 BioPac\_path = fr'{directory}\\P\_{ID}.acq'  
 BioPac = bioread.read\_file(BioPac\_path)  
 EDA = pd.read\_csv(fr'{directory}\\EDA.csv')  
 RR = pd.read\_csv(fr'{directory}\\RR.csv')  
 RSP\_D = BioPac.named\_channels['Diaphragmatic Respiration'].data  
 RSP\_D\_T = BioPac.named\_channels['Diaphragmatic Respiration'].time\_index  
 RSP\_C = BioPac.named\_channels['Chest Respiration'].data  
 RSP\_C\_T = BioPac.named\_channels['Chest Respiration'].time\_index  
 eda\_data = EDA['EDA']  
 if VAS\_plot:  
 # Original Combined Plot  
 fig, ax = plt.subplots(6, figsize=(15, 18), sharex=True)  
  
 ax[0].plot(EDA['Time'], EDA['EDA'], label=f"Participant {ID}")  
 ax[1].scatter(RSP\_D\_T, RSP\_D, label=f"Participant {ID}")  
 ax[2].scatter(RSP\_C\_T, RSP\_C, label=f"Participant {ID}")  
 ax[3].scatter(RR['Time'], RR['RR'], label=f"Participant {ID}")  
 # Set y-axis limits to 500-1500 ms for consistent RR interval scaling  
 ax[3].set\_ylim([400, 1500])  
  
 stress\_data = Trigger\_df[Trigger\_df["Task"] == "VAS\_Stress"]  
 fatigue\_data = Trigger\_df[Trigger\_df["Task"] == "VAS\_Fatigue"]  
  
 ax[4].scatter(stress\_data["Start"], stress\_data["Score"], color='blue', label="VAS Stress")  
 ax[5].scatter(fatigue\_data["Start"], fatigue\_data["Score"], color='red', label="VAS Fatigue")  
  
 for \_, trigger in Trigger\_df.iterrows():  
 start\_time = trigger["Start"]  
 end\_time = trigger["End"]  
 task\_name = trigger["Task"]  
 task\_score = trigger["Score"]  
 if pd.notna(end\_time):  
 if task\_name == 'CB\_easy' or task\_name == 'CB\_hard':  
 ax[0].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[1].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[2].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[3].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[4].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[5].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2, min(eda\_data) - 0.10 \* max(eda\_data), task\_name,  
 ha='center', fontsize=10, color='black')  
 elif task\_name == 'PA\_easy' or task\_name == 'PA\_medium' or task\_name == 'PA\_hard':  
 ax[0].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[1].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[2].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[3].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[4].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[5].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2, min(eda\_data) - 0.10 \* max(eda\_data), task\_name,  
 ha='center', fontsize=10, color='black')  
 elif task\_name == 'TC\_easy' or task\_name == 'TC\_hard':  
 ax[0].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[1].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[2].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[3].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[4].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[5].axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2, min(eda\_data) - 0.10 \* max(eda\_data), task\_name,  
 ha='center', fontsize=10, color='black')  
 else:  
 ax[0].axvspan(start\_time, end\_time, color='#C3E6CB', alpha=0.3)  
 ax[1].axvspan(start\_time, end\_time, color='#C3E6CB', alpha=0.3)  
 ax[2].axvspan(start\_time, end\_time, color='#C3E6CB', alpha=0.3)  
 ax[3].axvspan(start\_time, end\_time, color='#C3E6CB', alpha=0.3)  
 ax[4].axvspan(start\_time, end\_time, color='#C3E6CB', alpha=0.3)  
 ax[5].axvspan(start\_time, end\_time, color='#C3E6CB', alpha=0.3)  
 ax[0].text((start\_time + end\_time) / 2, min(eda\_data) - 0.10 \* max(eda\_data), task\_name,  
 ha='center', fontsize=10, color='black')  
 else:  
 ax[0].axvline(start\_time, color='green', linestyle='--')  
 ax[1].axvline(start\_time, color='green', linestyle='--')  
 ax[2].axvline(start\_time, color='green', linestyle='--')  
 ax[3].axvline(start\_time, color='green', linestyle='--')  
 ax[4].axvline(start\_time, color='green', linestyle='--')  
 ax[5].axvline(start\_time, color='green', linestyle='--')  
  
 ax[0].set\_ylabel("EDA Signal")  
 ax[0].legend(loc="upper right", fontsize=8)  
 ax[1].set\_ylabel("RSP\_D")  
 ax[1].legend(loc="upper right", fontsize=8)  
 ax[2].set\_ylabel("RSP\_C")  
 ax[2].legend(loc="upper right", fontsize=8)  
 ax[3].set\_ylabel("ECG RR intervals")  
 ax[3].legend(loc="upper right", fontsize=8)  
 ax[4].set\_ylabel("VAS Stress Score")  
 ax[4].legend(loc="upper right", fontsize=8)  
 ax[5].set\_ylabel("VAS Fatigue Score")  
 ax[5].legend(loc="upper right", fontsize=8)  
  
 plt.tight\_layout()  
 plot\_path = fr'{directory}\\plots\\Trigger\_Plot\_{ID}\_vas.png'  
 plt.savefig(plot\_path, dpi=300)  
 plt.show()  
  
 # Create separate directories for each signal type if they don't exist  
 analysis\_base\_path = "C:\\Users\\e3bom\\Desktop\\Human Bio Signals Analysis\\Analysis"  
 eda\_path = f"{analysis\_base\_path}\\EDA"  
 rsp\_d\_path = f"{analysis\_base\_path}\\RSP\_D"  
 rsp\_c\_path = f"{analysis\_base\_path}\\RSP\_C"  
 rr\_path = f"{analysis\_base\_path}\\RR"  
  
 for path in [eda\_path, rsp\_d\_path, rsp\_c\_path, rr\_path]:  
 os.makedirs(path, exist\_ok=True)  
  
 # 1. Separate EDA Plot  
 fig\_eda = plt.figure(figsize=(12, 6))  
 ax\_eda = fig\_eda.add\_subplot(111)  
 ax\_eda.plot(EDA['Time'], EDA['EDA'], label=f"Participant {ID}")  
 ax\_eda.set\_title(f"EDA Signal - Participant {ID} Group:{Group}")  
 ax\_eda.set\_xlabel("Time (s)")  
 ax\_eda.set\_ylabel("EDA Signal")  
 ax\_eda.set\_ylim([0, 3]) # Set y-axis limits to 500-1200 ms for consistent RR interval scaling  
  
 # Add task regions to the EDA plot  
 for \_, trigger in Trigger\_df.iterrows():  
 start\_time = trigger["Start"]  
 end\_time = trigger["End"]  
 task\_name = trigger["Task"]  
  
 if pd.notna(end\_time):  
 if task\_name in ['CB\_easy', 'CB\_hard', 'PA\_easy', 'PA\_medium', 'PA\_hard', 'TC\_easy', 'TC\_hard']:  
 ax\_eda.axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 # Use transform for text positioning to avoid blended transform warning  
 # y\_pos = min(eda\_data) - 0.10 \* max(eda\_data)  
 # ax\_eda.text(0.5 \* (start\_time + end\_time), y\_pos,  
 # task\_name, ha='center', fontsize=10, color='black')  
 else:  
 ax\_eda.axvspan(start\_time, end\_time, color='#C3E6CB', alpha=0.3)  
 # y\_pos = min(eda\_data) - 0.10 \* max(eda\_data)  
 # ax\_eda.text(0.5 \* (start\_time + end\_time), y\_pos,  
 # task\_name, ha='center', fontsize=10, color='black')  
 else:  
 ax\_eda.axvline(start\_time, color='green', linestyle='--')  
  
 plt.legend(loc='upper right') # Specify legend location instead of 'best'  
 plt.tight\_layout()  
 plt.savefig(f"{eda\_path}\\EDA\_P{ID}.png", dpi=300)  
 plt.close()  
  
 # 2. Separate RSP\_D Plot  
 fig\_rsp\_d = plt.figure(figsize=(12, 6))  
 ax\_rsp\_d = fig\_rsp\_d.add\_subplot(111)  
 RSP\_D = RSP\_D - RSP\_D.mean()  
 ax\_rsp\_d.scatter(RSP\_D\_T, RSP\_D, label=f"Participant {ID}", s=1)  
 ax\_rsp\_d.set\_title(f"Diaphragmatic Respiration - Participant {ID} Group:{Group}")  
 ax\_rsp\_d.set\_xlabel("Time (s)")  
 ax\_rsp\_d.set\_ylabel("RSP\_D")  
 ax\_rsp\_d.set\_ylim([-8, 8]) # Set y-axis limits to 500-1200 ms for consistent RR interval scaling  
  
 # Add task regions to the RSP\_D plot  
 for \_, trigger in Trigger\_df.iterrows():  
 start\_time = trigger["Start"]  
 end\_time = trigger["End"]  
 task\_name = trigger["Task"]  
  
 if pd.notna(end\_time):  
 if task\_name in ['CB\_easy', 'CB\_hard', 'PA\_easy', 'PA\_medium', 'PA\_hard', 'TC\_easy', 'TC\_hard']:  
 ax\_rsp\_d.axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 else:  
 ax\_rsp\_d.axvspan(start\_time, end\_time, color='#C3E6CB', alpha=0.3)  
 else:  
 ax\_rsp\_d.axvline(start\_time, color='green', linestyle='--')  
  
 ax\_rsp\_d.legend(loc='upper right') # Specify legend location instead of 'best'  
 plt.tight\_layout()  
 plt.savefig(f"{rsp\_d\_path}\\RSP\_D\_P{ID}.png", dpi=300)  
 plt.close()  
  
 # 3. Separate RSP\_C Plot  
 fig\_rsp\_c = plt.figure(figsize=(12, 6))  
 ax\_rsp\_c = fig\_rsp\_c.add\_subplot(111)  
 RSP\_C=RSP\_C-RSP\_C.mean()  
 ax\_rsp\_c.scatter(RSP\_C\_T, RSP\_C, label=f"Participant {ID}", s=1)  
 ax\_rsp\_c.set\_title(f"Chest Respiration - Participant {ID} Group:{Group}")  
 ax\_rsp\_c.set\_xlabel("Time (s)")  
 ax\_rsp\_c.set\_ylabel("RSP\_C")  
 ax\_rsp\_c.set\_ylim([-8, 8]) # Set y-axis limits to 500-1200 ms for consistent RR interval scaling  
  
 # Add task regions to the RSP\_C plot  
 for \_, trigger in Trigger\_df.iterrows():  
 start\_time = trigger["Start"]  
 end\_time = trigger["End"]  
 task\_name = trigger["Task"]  
  
 if pd.notna(end\_time):  
 if task\_name in ['CB\_easy', 'CB\_hard', 'PA\_easy', 'PA\_medium', 'PA\_hard', 'TC\_easy', 'TC\_hard']:  
 ax\_rsp\_c.axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 else:  
 ax\_rsp\_c.axvspan(start\_time, end\_time, color='#C3E6CB', alpha=0.3)  
 else:  
 ax\_rsp\_c.axvline(start\_time, color='green', linestyle='--')  
  
 ax\_rsp\_c.legend(loc='upper right') # Specify legend location instead of 'best'  
 plt.tight\_layout()  
 plt.savefig(f"{rsp\_c\_path}\\RSP\_C\_P{ID}.png", dpi=300)  
 plt.close()  
  
 # 4. Separate RR Plot  
 fig\_rr = plt.figure(figsize=(15, 3)) # width 15″, height 3″  
 ax\_rr = fig\_rr.add\_subplot(111)  
 ax\_rr.scatter(RR['Time'], RR['RR'], label=f"Participant {ID}",  
 s=10) # Increased point size to 25 for better visibility  
 # Set y-axis limits to 500-1200 ms for consistent RR interval scaling  
 ax\_rr.set\_title(f"RR Intervals - Participant {ID} Group:{Group}" )  
 ax\_rr.set\_xlabel("Time (s)")  
 ax\_rr.set\_ylabel("RR Interval (ms)")  
 ax\_rr.set\_ylim([400, 1500]) # Set y-axis limits to 500-1200 ms for consistent RR interval scaling  
  
 # Add task regions to the RR plot  
 for \_, trigger in Trigger\_df.iterrows():  
 start\_time = trigger["Start"]  
 end\_time = trigger["End"]  
 task\_name = trigger["Task"]  
  
 if pd.notna(end\_time):  
 if task\_name in ['CB\_easy', 'CB\_hard', 'PA\_easy', 'PA\_medium', 'PA\_hard', 'TC\_easy', 'TC\_hard']:  
 ax\_rr.axvspan(start\_time, end\_time, color='#AEC6CF', alpha=0.3)  
 else:  
 ax\_rr.axvspan(start\_time, end\_time, color='#C3E6CB', alpha=0.3)  
 else:  
 ax\_rr.axvline(start\_time, color='green', linestyle='--')  
  
 ax\_rr.legend(loc='upper right')  
 plt.tight\_layout()  
 plt.savefig(f"{rr\_path}\\RR\_P{ID}.png", dpi=300)  
 plt.close()  
  
 if Cor\_plot:  
 # ──────────────────────────── 1. BEFORE the participant loop ─────────────────────────  
  
 # ── constants for this analysis -------------------------------------------------------  
 window, overlap = 30, 0.0  
 base\_dir = fr"{self.path}\Participants\{Group}\_group\P\_{ID}"  
 hrv\_file = fr"{base\_dir}\Features\HRV\HRV\_Time\_{window}\_{overlap}.csv"  
  
 if not os.path.exists(hrv\_file):  
 print(f"[WARN] HRV file not found for P\_{ID}: {hrv\_file}")  
 continue  
  
 # 0) read + drop unneeded columns ------------------------------------------------------  
 hrv\_df = (pd.read\_csv(hrv\_file)  
 .drop(columns=["Time", "Class"], errors="ignore"))  
  
 # 1) aggregates ------------------------------------------------------------------------  
 stress\_df = (hrv\_df  
 .dropna(subset=["Stress"])  
 .groupby(["ID", "Group", "Stress"], as\_index=False)  
 .mean()  
 .drop(columns=["Fatigue"], errors="ignore"))  
  
 fatigue\_df = (hrv\_df.dropna(subset=["Fatigue"])  
 .groupby(["ID", "Group", "Stress"], as\_index=False)  
 .mean()  
 .drop(columns=["Stress"], errors="ignore"))  
 stress\_all = pd.concat([stress\_all, stress\_df], ignore\_index=True)  
 fatigue\_all = pd.concat([fatigue\_all, fatigue\_df], ignore\_index=True)  
 fatigue\_df=fatigue\_df.drop(columns=['ID','Group'])  
 stress\_df=stress\_df.drop(columns=['ID','Group'])  
 # keep Stress rows for the big scatter later  
 tmp\_stress = stress\_df.copy()  
 tmp\_stress["ID"] = ID  
 tmp\_stress["Group"] = Group  
  
 # ── iterate over Stress + Fatigue -----------------------------------------------------  
 targets = [("Stress", stress\_df), ("Fatigue", fatigue\_df)]  
 plots\_dir\_stress = os.path.join(base\_dir, r"Features\HRV\Cor\Plot\Stress")  
 plots\_dir\_fatigue = os.path.join(base\_dir, r"Features\HRV\Cor\Plot\Fatighe")  
 plots\_dir\_stress\_analysis = r"C:\Users\e3bom\Desktop\Human Bio Signals Analysis\Analysis\RR\Plot\cor\Stress"  
 plots\_dir\_fatigue\_analysis = r"C:\Users\e3bom\Desktop\Human Bio Signals Analysis\Analysis\RR\Plot\cor\Fatigue"  
  
 os.makedirs(plots\_dir\_stress, exist\_ok=True)  
 os.makedirs(plots\_dir\_fatigue, exist\_ok=True)  
 os.makedirs(plots\_dir\_stress\_analysis, exist\_ok=True)  
 os.makedirs(plots\_dir\_fatigue\_analysis, exist\_ok=True)  
  
 corr\_rows\_participant = [] # for the per‑participant Excel sheet  
  
 for y\_label, tbl in targets:  
 if tbl.empty:  
 print(f"[WARN] {y\_label} table is empty for P\_{ID}; skipping plot")  
 continue  
  
 feature\_cols = [c for c in tbl.columns if c != y\_label]  
 n\_feats = len(feature\_cols)  
  
 fig, axes = plt.subplots(1, n\_feats,  
 figsize=(3 \* n\_feats, 4),  
 sharey=True)  
 axes = axes if isinstance(axes, (list, np.ndarray)) else [axes]  
  
 for ax, feat in zip(axes, feature\_cols):  
 x = tbl[feat].values  
 y = tbl[y\_label].values  
  
 # scatter  
 ax.scatter(x, y, alpha=0.8)  
  
 # regression line & correlation  
 slope, intercept, r, p, \_ = linregress(x, y)  
 x\_line = np.linspace(x.min(), x.max(), 100)  
 ax.plot(x\_line, intercept + slope \* x\_line, linestyle="--")  
  
 ax.set\_title(f"{feat} (r = {r:.2f})")  
 ax.set\_xlabel("mean value")  
 ax.grid(True)  
  
 # save correlation record  
 corr\_rows\_participant.append(  
 {"ID": ID, "Group": Group, "Target": y\_label,  
 "Feature": feat, "r": r, "p": p}  
 )  
 plot\_path = fr'{directory}\\plots\\Trigger\_Plot\_{ID}\_vas.png'  
 axes[0].set\_ylabel(y\_label)  
 fig.suptitle(f"P{ID} {Group}– {y\_label} vs HRV features (window {window}s)",  
 fontsize=14)  
 fig.tight\_layout(rect=[0, 0, 1, 0.96])  
 if y\_label=='Fatigue':  
 plot\_path1 = os.path.join(plots\_dir\_fatigue, f"{y\_label}\_vs\_HRV\_P{ID}\_{window}s.png")  
 plot\_path2= os.path.join(plots\_dir\_fatigue\_analysis, f"{y\_label}\_vs\_HRV\_P{ID}\_{window}s.png")  
 else:  
 plot\_path1 = os.path.join(plots\_dir\_stress, f"{y\_label}\_vs\_HRV\_P{ID}\_{window}s.png")  
 plot\_path2= os.path.join(plots\_dir\_stress\_analysis, f"{y\_label}\_vs\_HRV\_P{ID}\_{window}s.png")  
  
 fig.savefig(plot\_path1, dpi=300, bbox\_inches="tight")  
 fig.savefig(plot\_path2, dpi=300, bbox\_inches="tight")  
 plt.close(fig)  
 print(f"[INFO] subplot figure saved → {plot\_path}")  
  
 # ── write participant‑level correlation Excel sheet ------------------------------  
 corr\_dir = os.path.join(base\_dir, r"Features\HRV\Cor")  
 os.makedirs(corr\_dir, exist\_ok=True)  
 pd.DataFrame(corr\_rows\_participant).to\_excel(  
 os.path.join(corr\_dir, f"corr\_P{ID}\_{window}s.xlsx"), index=False  
 )  
 # also extend the global master list  
 all\_corr\_records.extend(corr\_rows\_participant)  
  
 # ─────────────────────────── END participant block ──────────────────────────────  
  
 # ──────────────────────────── 3. AFTER the participant loop ─────────────────────────  
 # (Place this once, at the very bottom of AX\_plot\_signals\_VAS)  
  
 # ── 3a. master correlation workbook ──────────────────────────────────────────────  
 master\_corr\_df = pd.DataFrame(all\_corr\_records)  
 master\_excel = os.path.join(  
 self.path, r"Participants\All\_HRV\_Correlations\_30s.xlsx"  
 )  
 # ── 1. pick the numeric column you want on the y-axis ───────────────  
 Y = "r" # or "p" if you prefer p-values  
 # ── 2. basic box-plot, grouped by Feature and coloured by Target ────  
 plt.figure(figsize=(12, 6))  
 ax = sns.boxplot(data=master\_corr\_df, x="Feature", y=Y, hue="Target",  
 palette="Set2", width=0.6, fliersize=2)  
 # cosmetic touches  
 ax.axhline(0, color="gray", lw=0.8, ls="--") # reference line for r=0  
 ax.set\_xlabel("") # feature names already below  
 ax.set\_ylabel("Pearson r" if Y == "r" else "p-value")  
 ax.set\_title("Distribution of correlations per HRV feature")  
 ax.legend(title="Target", loc="upper right")  
 plt.xticks(rotation=25, ha="right")  
 plt.tight\_layout()  
 plt.show()  
 master\_corr\_df.to\_excel(master\_excel, index=False)  
 print(f"[INFO] master correlation workbook saved → {master\_excel}")  
  
 # ── 3b. build Stress scatter matrices ────────────────────────────────────────────  
 master\_excel\_stress = os.path.join(  
 self.path, r"Participants\All\_HRV\_stress\_30s.xlsx"  
 )  
 stress\_all.to\_excel(master\_excel\_stress, index=False)  
 print(f"[INFO] master correlation workbook saved → {master\_excel}")  
  
 # ── 3b. build Fatigue scatter matrices ────────────────────────────────────────────  
 master\_excel\_fatigue = os.path.join(  
 self.path, r"Participants\All\_HRV\_fatigue\_30s.xlsx"  
 )  
 fatigue\_all.to\_excel(master\_excel\_fatigue, index=False)  
 print(f"[INFO] master correlation workbook saved → {master\_excel}")  
  
 def HRV\_Window\_Feature(self, ID,rangeID):  
  
 # ── 1. Load participant table ────────────────────────────────────  
 part\_csv = fr"{self.path}\Participants\participation management.csv"  
 part\_df = (pd.read\_csv(part\_csv)  
 .dropna(axis=1, how="all")  
 .dropna(subset=["participant", "Date", "departmant"], how="all"))  
 part\_df["code"] = pd.to\_numeric(part\_df["code"], errors="coerce").astype("Int64")  
  
 if ID is not None:  
 if rangeID:  
 part\_df = part\_df[part\_df['code'] >= ID]  
 else:  
 part\_df = part\_df[part\_df['code'] == ID]  
  
 # ── 2. Master record holder ──────────────────────────────────────  
 big\_records: list[dict] = []  
  
 # ── 3. Iterate over participants ────────────────────────────────  
 for \_, row in part\_df.iterrows():  
 ID = row["code"]  
 Group = row["Group"]  
  
 # 3.1 Paths  
 base\_path = fr"{self.path}\Participants\{Group}\_group\P\_{ID}"  
 feature\_dir = os.path.join(base\_path, "Features", "HRV")  
 time\_plot\_dir = os.path.join(base\_path, "plots", "HRV\_TIME\_WINDOW")  
 hist\_plot\_dir = os.path.join(base\_path, "plots", "HRV\_TIME\_HIST")  
 os.makedirs(time\_plot\_dir, exist\_ok=True)  
 os.makedirs(hist\_plot\_dir, exist\_ok=True)  
  
 fscore\_records: list[dict] = []  
  
 # 3.2 Loop over \*every\* HRV window file  
 for fname in os.listdir(feature\_dir):  
 if not (fname.startswith("HRV\_Time\_") and fname.endswith(".csv")):  
 continue  
  
 csv\_path = os.path.join(feature\_dir, fname)  
 df = pd.read\_csv(csv\_path)  
 if df.empty:  
 continue  
  
 # Parse window / overlap from file name: HRV\_Time\_<win>s\_<ovlp>percent.csv  
 parts = fname.replace(".csv", "").split("\_")  
 window = int(parts[2].rstrip("s"))  
 overlap = parts[3].rstrip("percent")  
  
 # ── 3.2.1 Class label normalisation ───────────────────  
 if "Class" not in df.columns:  
 continue  
 classified = df.dropna(subset=["Class"]).copy()  
 rest\_mask = classified["Class"].str.contains(  
 r"^(breath|natural|music)", case=False, na=False  
 )  
 classified.loc[rest\_mask, "Class"] = "rest"  
 classified = classified[classified["Class"].isin(["rest", "test"])]  
 if classified.empty:  
 continue  
  
 # ── 3.2.2 Histogram plot ──────────────────────────────  
 hrv\_feats = [  
 "HRV\_CVNN", "HRV\_RMSSD", "HRV\_SDNN",  
 "HRV\_MeanNN", "HRV\_pNN20", "HRV\_pNN50"  
 ]  
 n\_feats = len(hrv\_feats)  
 n\_cols = 2  
 n\_rows = int(np.ceil(n\_feats / n\_cols))  
  
 hist\_subdir = os.path.join(hist\_plot\_dir, f"{window}s\_{overlap}")  
 os.makedirs(hist\_subdir, exist\_ok=True)  
  
 fig, axes = plt.subplots(n\_rows, n\_cols,  
 figsize=(12, 4 \* n\_rows),  
 constrained\_layout=True)  
 axes = axes.flatten()  
  
 for ax, feat in zip(axes, hrv\_feats):  
 rest\_vals = classified.loc[classified["Class"] == "rest", feat].dropna()  
 test\_vals = classified.loc[classified["Class"] == "test", feat].dropna()  
  
 if rest\_vals.empty and test\_vals.empty:  
 ax.axis("off")  
 continue  
  
 ax.hist(rest\_vals, bins=30, alpha=0.6, density=True, label="rest")  
 ax.hist(test\_vals, bins=30, alpha=0.6, density=True, label="test")  
 ax.set\_title(feat)  
 ax.legend()  
  
 fig.suptitle(  
 f"Participant P\_{ID}\nWindow: {window}s Overlap: {overlap}",  
 fontsize=14,  
 ha="center"  
 )  
  
 out\_png = os.path.join(  
 hist\_subdir,  
 f"P\_{ID}\_{window}s\_{overlap}\_hist.png"  
 )  
 fig.savefig(out\_png, dpi=300)  
 plt.close(fig)  
  
 # ── 3.2.3 F-score calculation ─────────────────────────  
 for feat in hrv\_feats:  
 rest = classified.loc[classified["Class"] == "rest", feat].dropna()  
 test = classified.loc[classified["Class"] == "test", feat].dropna()  
 if rest.empty or test.empty:  
 continue  
  
 diff2 = (rest.mean() - test.mean()) \*\* 2  
 var\_sum = rest.var(ddof=1) + test.var(ddof=1)  
 f\_val = diff2 / var\_sum if var\_sum else np.nan  
  
 fscore\_records.append({  
 "Participant": ID,  
 "Group": Group,  
 "Window\_s": window,  
 "Overlap\_pc": overlap,  
 "Feature": feat,  
 "F\_score": f\_val  
 })  
  
 # 3.3 Per-participant Excel --------------------------------  
 if fscore\_records:  
 p\_df = pd.DataFrame(fscore\_records).sort\_values("F\_score", ascending=False)  
 p\_xlsx = os.path.join(base\_path, f"F\_scores\_P{ID}.xlsx")  
 p\_df.to\_excel(p\_xlsx, index=False)  
 print(f"✅ F-scores saved → {p\_xlsx}")  
  
 big\_records.extend(fscore\_records)  
  
 # ── 4. Master workbook -------------------------------------------  
 if big\_records:  
 big\_df = pd.DataFrame(big\_records).sort\_values(  
 ["Window\_s", "Overlap\_pc", "F\_score"],  
 ascending=[True, True, False])  
  
 summary = (big\_df  
 .groupby(["Window\_s", "Overlap\_pc"], as\_index=False)["F\_score"]  
 .mean()  
 .rename(columns={"F\_score": "F\_score\_mean"})  
 .sort\_values(["Window\_s", "Overlap\_pc"]))  
  
 master\_xlsx = os.path.join(self.path, "Participants", "F\_scores\_All.xlsx")  
 with pd.ExcelWriter(master\_xlsx, engine="openpyxl") as writer:  
 big\_df.to\_excel(writer, sheet\_name="Raw", index=False)  
 summary.to\_excel(writer, sheet\_name="Summary", index=False)  
  
 print(f"✅ Master workbook saved → {master\_xlsx}")  
 else:  
 print("⚠️ No F-scores produced – check your input files.")  
  
 # -------------------------------------------------------------------  
 def RSP\_Parts(self,ID,Group):  
 Participants\_path = f'{self.path}\Participants\participation management.csv'  
 Participants\_df = pd.read\_csv(Participants\_path, header=0)  
 Participants\_df = Participants\_df.dropna(axis=1, how='all')  
 Participants\_df = Participants\_df.dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
 if ID is not None:  
 Participants\_df = Participants\_df[Participants\_df['code'] == ID]  
 for \_, row in Participants\_df.iterrows():  
 ID = row['code']  
 Group = row['Group']  
 directory = fr'{self.path}\Participants\{Group}\_group\P\_{ID}'  
 directory\_Breath = fr'{self.path}\Participants\{Group}\_group\P\_{ID}\plot\breath'  
  
 Trigger\_path = fr'{directory}\Trigger\_{ID}.csv'  
 Trigger\_df = pd.read\_csv(Trigger\_path, header=0)  
  
 BioPac\_path = fr'{directory}\P\_{ID}.acq'  
 BioPac = bioread.read\_file(BioPac\_path)  
 sample\_rate = BioPac.named\_channels['Chest Respiration'].samples\_per\_second  
 if Group.lower() == 'control':  
 task\_pattern = r'break2|PA\_easy|PA\_medium|PA\_hard|break3'  
 elif Group.lower() == 'music':  
 task\_pattern = r'music2|PA\_easy|PA\_medium|PA\_hard|music3'  
 elif Group.lower() == 'breath':  
 task\_pattern = r'breath2|PA\_easy|PA\_medium|PA\_hard|breath3'  
 else:  
 task\_pattern = r'' # fallback if needed  
  
 mask = Trigger\_df['Task'].str.contains(task\_pattern, case=False, na=False)  
 filtered\_df = Trigger\_df[mask].reset\_index(drop=True) # mask = Trigger\_df['Task'].str.contains(r'breath ?[1-4]|star', case=False, na=False)  
 Itterations=[0,1,2,3,4]  
 # fig, ax = plt.subplots(3, figsize=(10, 14))  
 for i in Itterations:  
 start=int(filtered\_df.iloc[i]['Start']\*sample\_rate)  
 end=int(filtered\_df.iloc[i]['End']\*sample\_rate)  
 part\_data = BioPac.named\_channels['Chest Respiration'].data[start:end]  
 part\_time=BioPac.named\_channels['Chest Respiration'].time\_index[start:end]  
 plt.plot(part\_time,part\_data)  
 # ax[i].plot(part\_time,part\_data)  
 plt.savefig(fr'{directory\_Breath}\breath')  
 plt.show()  
 def HRV\_Window\_Feature\_all(self):  
 window\_sizes = [5, 10, 30, 60] # seconds  
 overlaps = [0.0, 0.5] # no overlap and 50%  
  
 time\_window\_hist\_dir = fr'D:\Human Bio Signals Analysis\Participants\Dataset\HRV\_TIME\_HIST'  
 os.makedirs(time\_window\_hist\_dir, exist\_ok=True)  
  
 for window\_size in window\_sizes:  
 for overlap in overlaps:  
 try:  
 dataset\_path = fr'D:\Human Bio Signals Analysis\Participants\Dataset\HRV\_AllParticipants\_{window\_size}s\_{int(overlap \* 100)}percent.csv'  
 results = pd.read\_csv(dataset\_path)  
  
 plot\_base = fr'D:\Human Bio Signals Analysis\Participants\Dataset\Plots\{window\_size}s\_{int(overlap \* 100)}percent'  
 hist\_dir = os.path.join(plot\_base, "HRV\_TIME\_HIST")  
 os.makedirs(hist\_dir, exist\_ok=True)  
  
 hrv\_features = ['HRV\_CVNN', 'HRV\_RMSSD', 'HRV\_SDNN', 'HRV\_MeanNN', 'HRV\_pNN20', 'HRV\_pNN50']  
  
 fig, ax = plt.subplots(len(hrv\_features), figsize=(12, 20))  
 for i, feature in enumerate(hrv\_features):  
 if feature not in results.columns:  
 continue  
  
 rest\_data = results[results["Class"].isin(["breath", "music", "natural"])][feature].dropna()  
 test\_data = results[results["Class"] == "test"][feature].dropna()  
  
 ax[i].hist(rest\_data, bins=30, alpha=0.6, label="Rest", density=True)  
 ax[i].hist(test\_data, bins=30, alpha=0.6, label="Test", density=True)  
 ax[i].set\_title(  
 f"{feature} Histogram\nWindow={window\_size}s, Overlap={int(overlap \* 100)}%")  
 ax[i].set\_xlabel(feature)  
 ax[i].set\_ylabel("Density")  
 ax[i].legend()  
  
 plt.tight\_layout()  
 hist\_path = os.path.join(hist\_dir,  
 f"HRV\_Histogram\_win{window\_size}s\_overlap{int(overlap \* 100)}.png")  
 plt.savefig(hist\_path, dpi=300)  
 plt.close()  
  
 print(f"✅ Saved histogram for {window\_size}s / {int(overlap \* 100)}%: {hist\_path}")  
  
 except Exception as e:  
 print(f"❌ Error for {window\_size}s, {overlap \* 100:.0f}%: {e}")  
  
 def HRV\_Window\_2Features(self, ID, Group):  
 Participants\_path = f'{self.path}\Participants\participation management.csv'  
 Participants\_df = pd.read\_csv(Participants\_path, header=0)  
 Participants\_df = Participants\_df.dropna(axis=1, how='all')  
 Participants\_df = Participants\_df.dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
  
 if ID is not None:  
 Participants\_df = Participants\_df[Participants\_df['code'] == ID]  
  
 for \_, row in Participants\_df.iterrows():  
 ID = row['code']  
 Group = row['Group']  
 directory = fr'{self.path}\Participants\{Group}\_group\P\_{ID}'  
  
 Trigger\_path = fr'{directory}\Trigger\_{ID}.csv'  
 Trigger\_df = pd.read\_csv(Trigger\_path, header=0)  
  
 BioPac\_path = fr'{directory}\P\_{ID}.acq'  
 BioPac = bioread.read\_file(BioPac\_path)  
  
 EDA = pd.read\_csv(fr'{directory}\EDA.csv')  
 eda\_data = EDA['EDA']  
  
 sample\_rate = BioPac.named\_channels['ECG'].samples\_per\_second  
 part\_data = BioPac.named\_channels['ECG'].data  
  
 window\_sizes = [5, 10, 30, 60] # seconds  
 overlaps = [0.0, 0.5] # no overlap and 50%  
 # window\_sizes = [10] # seconds  
 # overlaps = [0.5] # no overlap and 50%  
 time\_window\_plot\_dir = fr'{directory}\plots\HRV\_TIME\_scatter'  
 os.makedirs(time\_window\_plot\_dir, exist\_ok=True)  
  
 def label\_window(time, trigger\_df):  
 for \_, row in trigger\_df.iterrows():  
 if pd.notna(row["End"]) and row["Task"].startswith("Breath"):  
 if row["Start"] <= time <= row["End"]:  
 return "breath"  
 elif pd.notna(row["End"]) and not row["Task"].startswith("VAS") and not row["Task"].startswith(  
 "Breath"):  
 if row["Start"] <= time <= row["End"]:  
 return "test"  
 return None  
  
 for window\_size in window\_sizes:  
 for overlap in overlaps:  
 results = pd.DataFrame()  
 window\_samples = int(window\_size \* sample\_rate)  
 step = int(window\_samples \* (1 - overlap))  
  
 for j, i in enumerate(range(0, len(part\_data) - window\_samples + 1, step)):  
 segment\_data = part\_data[i:i + window\_samples]  
 try:  
 ecg\_cleaned = nk.ecg\_clean(segment\_data, sampling\_rate=sample\_rate)  
 \_, rpeaks = nk.ecg\_peaks(ecg\_cleaned, sampling\_rate=sample\_rate)  
 if len(rpeaks["ECG\_R\_Peaks"]) > 2:  
 hrv = nk.hrv\_time(rpeaks, sampling\_rate=sample\_rate, show=False)  
 center\_time = (i + window\_samples / 2) / sample\_rate  
 new\_row = {  
 'Time': center\_time,  
 'HRV\_CVNN': hrv.get('HRV\_CVNN', [None])[0],  
 'HRV\_RMSSD': hrv.get('HRV\_RMSSD', [None])[0],  
 'HRV\_SDNN': hrv.get('HRV\_SDNN', [None])[0],  
 'HRV\_MeanNN': hrv.get('HRV\_MeanNN', [None])[0],  
 'HRV\_pNN20': hrv.get('HRV\_pNN20', [None])[0],  
 'HRV\_pNN50': hrv.get('HRV\_pNN50', [None])[0]  
 }  
 results = pd.concat([results, pd.DataFrame([new\_row])], ignore\_index=True)  
 except Exception as e:  
 print(f"Error in window {j}: {e}")  
  
  
 # Plot HRV time series  
 hrv\_features = ['HRV\_CVNN', 'HRV\_RMSSD', 'HRV\_SDNN', 'HRV\_MeanNN', 'HRV\_pNN20', 'HRV\_pNN50']  
  
 # Label and create histogram for this setting  
 results['Class'] = results['Time'].apply(lambda t: label\_window(t, Trigger\_df))  
 results.to\_csv(fr'{directory}\Features\HRV\_Time\_{window\_size}\_{overlap}.csv')  
 classified = results.dropna(subset=['Class'])  
 # Define color mapping  
 color\_map = {'breath': 'blue', 'test': 'orange'}  
  
 # Drop rows with missing class values  
 filtered\_results = results.dropna(subset=['Class'])  
  
 # Map colors  
 colors = filtered\_results['Class'].map(color\_map)  
  
 # Create scatter plot  
 plt.scatter(filtered\_results['HRV\_CVNN'], filtered\_results['HRV\_RMSSD'], c=colors)  
  
 # Add labels and show plot  
 plt.xlabel('HRV\_CVNN')  
 plt.ylabel('HRV\_RMSSD')  
 plt.title('HRV\_CVNN vs HRV\_RMSSD by Class')  
 plt.savefig(fr'{time\_window\_plot\_dir}\Scatter\_{window\_size}\_{overlap}.png')  
 plt.show()  
 def CreateDataset(self, ID,rangeID):  
 *"""Build time‑windowed feature datasets and add Stress, Fatigue,  
 and FromTrigger columns drawn from the participant's Trigger CSV."""* # ── Config ──────────────────────────────────────────  
 interpolate\_rr = False # RR interpolation toggle  
 # window\_sizes = [30] # sec  
 # overlaps = [0.0]  
 window\_sizes = [5, 10, 30, 60] # sec  
 overlaps = [0.0, 0.5]  
 # fraction  
 total\_dataset\_dir = r'C:\Users\e3bom\Desktop\Human Bio Signals Analysis\Participants\Dataset'  
 os.makedirs(total\_dataset\_dir, exist\_ok=True)  
  
 # ── Participant table ───────────────────────────────  
 Participants\_path = f'{self.path}\\Participants\\participation management.csv'  
 Participants\_df = (  
 pd.read\_csv(Participants\_path)  
 .dropna(axis=1, how='all')  
 .dropna(subset=['participant', 'Date', 'departmant'], how='all')  
 )  
 Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
 if ID is not None:  
 if rangeID:  
 Participants\_df = Participants\_df[Participants\_df['code'] >= ID]  
 else:  
 Participants\_df = Participants\_df[Participants\_df['code'] == ID]  
  
 # ── Accumulators for group datasets ─────────────────  
 total\_HRV\_dict = {}  
 total\_EDA\_dict = {}  
 total\_RSP\_chest\_dict = {}  
 total\_RSP\_diaph\_dict = {}  
  
 # ── Iterate participants ────────────────────────────  
 for \_, row in Participants\_df.iterrows():  
 ID = row['code']  
 Group = row['Group']  
 directory = fr'{self.path}\Participants\{Group}\_group\P\_{ID}'  
  
 try:  
 Trigger\_df = pd.read\_csv(fr'{directory}\Trigger\_{ID}.csv')  
 # ── Ratings table: keep only rows that actually carry stress / fatigue values ──  
 # create two new columns, NaN everywhere except the correct VAS rows  
  
 Trigger\_df["Stress"] = np.where(Trigger\_df["Task"] == "VAS\_Stress",  
 Trigger\_df["Score"],  
 np.nan)  
  
 Trigger\_df["Fatigue"] = np.where(Trigger\_df["Task"] == "VAS\_Fatigue",  
 Trigger\_df["Score"],  
 np.nan)  
  
 # now your original line works  
 rating\_df = (Trigger\_df[(~Trigger\_df["Stress"].isna()) | (~Trigger\_df["Fatigue"].isna())]  
 .sort\_values("Start")  
 .reset\_index(drop=True))  
 rating\_df = rating\_df.drop(columns=['Score', 'End']) # ← preferred, explicit  
 BioPac= bioread.read\_file(fr'{directory}\P\_{ID}.acq')  
 sample\_rate = BioPac.named\_channels['ECG'].samples\_per\_second  
  
 # ── Helper: label per timepoint ─────────────  
 def label\_window(t):  
 for \_, tr in Trigger\_df.iterrows():  
 task = str(tr["Task"]).lower()  
 if pd.notna(tr["End"]) and tr["Start"] <= t <= tr["End"]:  
 if "breath" in task: return "breath"  
 if "music" in task: return "music"  
 if "baseline" in task or "break" in task: return "natural"  
 if not task.startswith("vas"): return "test"  
 return None  
  
 # ── Helper: stress & fatigue ───────────  
  
 first\_iteration = True  
 def trigger\_attrs(t):  
 *"""Return Stress, Fatigue and forward gap (Δt) to the \*\*next\*\*  
 VAS rating starting at or after \*t\*. Because the experiment  
 begins and ends with a rating there is always such a future  
 score, so no NaNs will appear.  
  
 Δt = rating\_start − t (always ≥ 0).  
 """* # Check if first or second rating is before or at time t  
  
 if rating\_df["Start"].iloc[0] >= t or rating\_df["Start"].iloc[1] >= t:  
 future\_rows = rating\_df # Use all rows  
 else:  
 future\_rows = rating\_df[rating\_df["Start"] >= t] # Filter for rows after t  
 # Not enough rows? Use the full table as a safe fallback  
 if len(future\_rows) < 2: # handles 0 or 1 row  
 future\_rows = rating\_df  
 if future\_rows.empty:  
 # Should nt raot happen, but fall back to the very lasting  
 tr\_s = future\_rows.iloc[-2] # first future rating (closest)  
 tr\_f = future\_rows.iloc[-1] # first future rating (closest)  
 else:  
 val = future\_rows.iloc[0]["Stress"]  
 if pd.isna(val): # first future rating (closest)  
 tr\_s = future\_rows.iloc[1]  
 tr\_f = future\_rows.iloc[0] # first future rating (closest)  
 else:  
 tr\_s = future\_rows.iloc[0] # first future rating (closest)  
 tr\_f = future\_rows.iloc[1] # first future rating (closest)  
 stress = tr\_s.get("Stress", np.nan)  
 fatigue = tr\_f.get("Fatigue", np.nan)  
 return stress, fatigue  
  
 # ── Helper: HRV calculation ────────────────  
 def calculate\_hrv\_features(rr\_values):  
 RR = pd.DataFrame({"RR": rr\_values})  
 RR.loc[(RR['RR'] < 300) | (RR['RR'] > 1500), 'RR'] = np.nan  
 if interpolate\_rr:  
 RR['RR'] = RR['RR'].interpolate(limit\_direction='both')  
 rr = RR['RR'].dropna().values  
 if len(rr) < 3:  
 return {k: None for k in [  
 'HRV\_MeanNN','HRV\_SDNN','HRV\_RMSSD',  
 'HRV\_CVNN','HRV\_pNN20','HRV\_pNN50']}  
 mean\_nn = np.mean(rr)  
 sdnn = np.std(rr, ddof=1)  
 diff\_rr = np.diff(rr)  
 return {  
 'HRV\_MeanNN': mean\_nn,  
 'HRV\_SDNN' : sdnn,  
 'HRV\_RMSSD' : np.sqrt(np.mean(diff\_rr \*\* 2)),  
 'HRV\_CVNN' : sdnn / mean\_nn \* 100,  
 'HRV\_pNN20' : np.sum(np.abs(diff\_rr) > 20) / len(diff\_rr) \* 100,  
 'HRV\_pNN50' : np.sum(np.abs(diff\_rr) > 50) / len(diff\_rr) \* 100,  
 }  
  
 # ── Window sweep ────────────────────────────  
 for window\_size in window\_sizes:  
 for overlap in overlaps:  
 window\_samples = int(window\_size \* sample\_rate)  
 step = int(window\_samples \* (1 - overlap))  
 suffix = f'Time\_{window\_size}\_{overlap}.csv'  
  
 HRV\_df = pd.DataFrame()  
 EDA\_df = pd.DataFrame()  
 RSP\_df\_chest = pd.DataFrame()  
 RSP\_df\_diaph = pd.DataFrame()  
  
 for signal\_type in BioPac.named\_channels:  
 signal\_type='ECG'  
 part\_data = BioPac.named\_channels[signal\_type].data  
 for j, i in enumerate(range(0, len(part\_data) - window\_samples + 1, step)):  
 segment= part\_data[i:i + window\_samples]  
 center\_time = (i + window\_samples / 2) / sample\_rate  
 stress, fatigue = trigger\_attrs(center\_time)  
 cls = label\_window(center\_time)  
  
 try:  
 # ── ECG → HRV ────────────────────────  
 if signal\_type == 'ECG':  
 ecg\_cleaned = nk.ecg\_clean(segment, sampling\_rate=sample\_rate)  
 \_, rpeaks = nk.ecg\_peaks(ecg\_cleaned, sampling\_rate=sample\_rate)  
 if len(rpeaks["ECG\_R\_Peaks"]) > 2:  
 RR = np.diff(rpeaks["ECG\_R\_Peaks"]) / sample\_rate \* 1000  
 hrv\_features = calculate\_hrv\_features(RR)  
 if hrv\_features['HRV\_MeanNN'] is not None:  
 HRV\_df = pd.concat([  
 HRV\_df,  
 pd.DataFrame([{\*\*{  
 'ID': ID,  
 'Group': Group,  
 'Time': center\_time,  
 \*\*hrv\_features,  
 'Class': cls,  
 'Stress': stress,  
 'Fatigue': fatigue  
 }}])  
 ], ignore\_index=True)  
  
 # ── EDA ──────────────────────────────  
 elif signal\_type == 'EDA':  
 eda\_signals, info = nk.eda\_process(segment, sampling\_rate=sample\_rate)  
 eda\_cleaned = medfilt(eda\_signals['EDA\_Clean'], kernel\_size=101)  
 EDA\_df = pd.concat([  
 EDA\_df,  
 pd.DataFrame([{\*\*{  
 'ID': ID,  
 'Group': Group,  
 'Time': center\_time,  
 'EDA\_Tonic\_Mean' : eda\_signals['EDA\_Tonic'].mean(),  
 'EDA\_Phasic\_Mean' : eda\_signals['EDA\_Phasic'].mean(),  
 'SCR\_Peaks\_Count' : len(info['SCR\_Peaks']),  
 'SCR\_Amplitude\_Mean' : eda\_signals['EDA\_Phasic'].max() - eda\_signals['EDA\_Phasic'].min(),  
 'EDA\_Clean\_Median' : eda\_cleaned.mean(),  
 'Class': cls,  
 'Stress': stress,  
 'Fatigue': fatigue  
 }}])  
 ], ignore\_index=True)  
  
 # ── Chest Respiration ────────────────  
 elif signal\_type == 'Chest Respiration':  
 rsp\_signals, \_ = nk.rsp\_process(segment, sampling\_rate=sample\_rate)  
 RSP\_df\_chest = pd.concat([  
 RSP\_df\_chest,  
 pd.DataFrame([{\*\*{  
 'ID': ID,  
 'Group': Group,  
 'Time': center\_time,  
 'RSP\_Rate' : rsp\_signals.get('RSP\_Rate', [None])[0],  
 'RSP\_Amplitude' : rsp\_signals.get('RSP\_Amplitude', [None])[0],  
 'RSP\_Symmetry\_PeakTrough' : rsp\_signals.get('RSP\_Symmetry\_PeakTrough', [None])[0],  
 'Class': cls,  
 'Stress': stress,  
 'Fatigue': fatigue  
 }}])  
 ], ignore\_index=True)  
  
 # ── Diaphragmatic Respiration ────────  
 elif signal\_type == 'Diaphragmatic Respiration':  
 rsp\_signals, \_ = nk.rsp\_process(segment, sampling\_rate=sample\_rate)  
 RSP\_df\_diaph = pd.concat([  
 RSP\_df\_diaph,  
 pd.DataFrame([{\*\*{  
 'ID': ID,  
 'Group': Group,  
 'Time': center\_time,  
 'RSP\_Rate' : rsp\_signals.get('RSP\_Rate', [None])[0],  
 'RSP\_Amplitude' : rsp\_signals.get('RSP\_Amplitude', [None])[0],  
 'RSP\_Symmetry\_PeakTrough' : rsp\_signals.get('RSP\_Symmetry\_PeakTrough', [None])[0],  
 'Class': cls,  
 'Stress': stress,  
 'Fatigue': fatigue  
 }}])  
 ], ignore\_index=True)  
 except Exception as e:  
 print(f"⛔ Error in window {j} for {signal\_type} - ID {ID}: {e}")  
  
 # ── Save participant‑level CSVs ───────────────  
 base\_path = fr'{directory}\Features'  
 os.makedirs(base\_path, exist\_ok=True)  
  
 if not HRV\_df.empty:  
 p = os.path.join(base\_path, 'HRV', f'HRV\_{suffix}')  
 os.makedirs(os.path.dirname(p), exist\_ok=True)  
 HRV\_df.to\_csv(p, index=False)  
 total\_HRV\_dict.setdefault((window\_size, overlap), pd.DataFrame())  
 total\_HRV\_dict[(window\_size, overlap)] = pd.concat([  
 total\_HRV\_dict[(window\_size, overlap)], HRV\_df])  
  
 if not EDA\_df.empty:  
 p = os.path.join(base\_path, 'EDA', f'EDA\_{suffix}')  
 os.makedirs(os.path.dirname(p), exist\_ok=True)  
 EDA\_df.to\_csv(p, index=False)  
 total\_EDA\_dict.setdefault((window\_size, overlap), pd.DataFrame())  
 total\_EDA\_dict[(window\_size, overlap)] = pd.concat([  
 total\_EDA\_dict[(window\_size, overlap)], EDA\_df])  
  
 if not RSP\_df\_chest.empty:  
 p = os.path.join(base\_path, 'RSP\_chest', f'RSP\_Chest\_{suffix}')  
 os.makedirs(os.path.dirname(p), exist\_ok=True)  
 RSP\_df\_chest.to\_csv(p, index=False)  
 total\_RSP\_chest\_dict.setdefault((window\_size, overlap), pd.DataFrame())  
 total\_RSP\_chest\_dict[(window\_size, overlap)] = pd.concat([  
 total\_RSP\_chest\_dict[(window\_size, overlap)], RSP\_df\_chest])  
  
 if not RSP\_df\_diaph.empty:  
 p = os.path.join(base\_path, 'RSP\_diaph', f'RSP\_Diaph\_{suffix}')  
 os.makedirs(os.path.dirname(p), exist\_ok=True)  
 RSP\_df\_diaph.to\_csv(p, index=False)  
 total\_RSP\_diaph\_dict.setdefault((window\_size, overlap), pd.DataFrame())  
 total\_RSP\_diaph\_dict[(window\_size, overlap)] = pd.concat([  
 total\_RSP\_diaph\_dict[(window\_size, overlap)], RSP\_df\_diaph])  
  
 print(f"✅ Saved P\_{ID} | W={window\_size}s | O={overlap \* 100:.0f}%")  
  
 except Exception as e:  
 print(f"❌ Error processing P\_{ID}: {e}")  
  
 # ── Save combined datasets ─────────────────────────  
 for (window\_size, overlap), df in total\_HRV\_dict.items():  
 df.to\_csv(os.path.join(total\_dataset\_dir,  
 f'Dataset\_HRV\_window{window\_size}s\_{int(overlap \* 100)}.csv'), index=False)  
 for (window\_size, overlap), df in total\_EDA\_dict.items():  
 df.to\_csv(os.path.join(total\_dataset\_dir,  
 f'Dataset\_EDA\_window{window\_size}s\_{int(overlap \* 100)}.csv'), index=False)  
 for (window\_size, overlap), df in total\_RSP\_chest\_dict.items():  
 df.to\_csv(os.path.join(total\_dataset\_dir,  
 f'Dataset\_RSP\_Chest\_window{window\_size}s\_{int(overlap \* 100)}.csv'), index=False)  
 for (window\_size, overlap), df in total\_RSP\_diaph\_dict.items():  
 df.to\_csv(os.path.join(total\_dataset\_dir,  
 f'Dataset\_RSP\_Diaph\_window{window\_size}s\_{int(overlap \* 100)}.csv'), index=False)

import seaborn as sns  
import statsmodels.formula.api as smf  
from sklearn.model\_selection import train\_test\_split, GroupKFold, GridSearchCV  
from sklearn.tree import DecisionTreeClassifier, plot\_tree  
from sklearn.ensemble import RandomForestClassifier  
from xgboost import XGBClassifier  
import matplotlib.pyplot as plt  
from scipy.stats import linregress  
import os  
from sklearn.model\_selection import TimeSeriesSplit  
import pandas as pd  
import numpy as np  
from sklearn.ensemble import RandomForestRegressor  
from pathlib import Path  
from sklearn.preprocessing import StandardScaler  
from sklearn.pipeline import Pipeline  
from sklearn.metrics import (accuracy\_score, precision\_score,  
 recall\_score, f1\_score,confusion\_matrix,  
 mean\_absolute\_error, mean\_squared\_error,  
 r2\_score)  
from sklearn.linear\_model import LogisticRegression, LinearRegression  
from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis  
from sklearn.tree import DecisionTreeRegressor  
  
class AnalysisData():  
 def \_\_init\_\_(self,Directory):  
 self.path = Directory  
 # self.sorted\_DATA = sorted\_DATA  
 # self.sampling\_frequency = sampling\_frequency  
 self.segment\_DATA = pd.DataFrame()  
 self.preprocessed\_DATA = pd.DataFrame()  
 self.window\_samples = 0  
  
 def \_binary\_metrics(y\_true, y\_pred):  
 *"""Return accuracy, precision, recall (== sensitivity) and specificity."""* acc = accuracy\_score(y\_true, y\_pred)  
 prec = precision\_score(y\_true, y\_pred, zero\_division=0)  
 sens = recall\_score(y\_true, y\_pred, zero\_division=0) # sensitivity  
 tn, fp, fn, tp = confusion\_matrix(y\_true, y\_pred).ravel()  
 spec = tn / (tn + fp) if (tn + fp) else np.nan # specificity  
 return acc, prec, sens, spec  
  
 @staticmethod  
 def chrono\_split(df, train\_pct=0.6, val\_pct=0.2, time\_col="Time"):  
 *"""  
 Chronological split → (train\_df, val\_df, test\_df).  
  
 val\_df is returned even if you do not use it later, so the signature  
 stays general.  
 """* df = df.sort\_values(time\_col).reset\_index(drop=True)  
 n = len(df)  
 t\_end = int(train\_pct \* n)  
 v\_end = int((train\_pct + val\_pct) \* n)  
 return df.iloc[:t\_end], df.iloc[t\_end:v\_end], df.iloc[v\_end:]  
  
 # ── 2. main routine ────────────────────────────────────────────────  
 def \_feature\_importance(self,model, feat\_cols):  
 if hasattr(model, "feature\_importances\_"):  
 return pd.Series(model.feature\_importances\_, index=feat\_cols)  
 if hasattr(model, "coef\_"):  
 w = np.abs(model.coef\_).ravel()  
 return pd.Series(w / w.sum(), index=feat\_cols)  
 raise ValueError("Unsupported model for importance extraction.")  
  
 @staticmethod  
 def \_best\_cutoff(y\_true: np.ndarray, y\_prob: np.ndarray, step: float = 0.01):  
 *"""  
 Scan thresholds ∈ (0,1] and return the one with maximal F1.  
 Returns (best\_threshold, metrics\_dict)  
 """* best\_thr, best\_f1 = 0.5, -1  
 best\_scores = {}  
 for thr in np.arange(step, 1.0, step):  
 y\_pred = (y\_prob >= thr).astype(int)  
 f1 = f1\_score(y\_true, y\_pred, zero\_division=0)  
 if f1 > best\_f1:  
 best\_f1 = f1  
 best\_thr = thr  
 best\_scores = {  
 "Accuracy": accuracy\_score(y\_true, y\_pred),  
 "Precision": precision\_score(y\_true, y\_pred, zero\_division=0),  
 "Recall": recall\_score(y\_true, y\_pred, zero\_division=0),  
 "F1": f1  
 }  
 return best\_thr, best\_scores  
  
 # ──────────────────── main routine ───────────────────────────────  
 def ML\_models\_particapent(self, ID: int | None = None, rangeID: bool = False):  
  
 # ── participants list ────────────────────────────────────────  
 pp\_csv = f"{self.path}\\Participants\\participation management.csv"  
 p\_df = (pd.read\_csv(pp\_csv)  
 .dropna(axis=1, how="all")  
 .dropna(subset=["participant", "Date", "departmant"], how="all"))  
 p\_df["code"] = pd.to\_numeric(p\_df["code"], errors="coerce").astype("Int64")  
  
 if ID is not None:  
 p\_df = p\_df[p\_df["code"] >= ID] if rangeID else p\_df[p\_df["code"] == ID]  
  
 win\_sizes, overlaps = [5, 10, 30, 60], [0.0, 0.5]  
  
 out\_root = Path(r"C:\Users\e3bom\OneDrive - post.bgu.ac.il\מחקר ביורפואי"  
 r"\Participants\Dataset\ML\Per\_Particapent")  
 out\_root.mkdir(parents=True, exist\_ok=True)  
  
 summary\_rows: list[dict] = []  
  
 # ── iterate participants ─────────────────────────────────────  
 for \_, prow in p\_df.iterrows():  
 pid, group = int(prow["code"]), prow["Group"]  
 out\_xlsx = out\_root / f"P{pid}\_ML\_Results.xlsx"  
 plot\_dir = out\_root / f"P{pid}\_Importance\_Plots"  
 plot\_dir.mkdir(exist\_ok=True)  
  
 cls\_metrics, reg\_metrics, import\_rows = [], [], []  
  
 # ── iterate window × overlap ────────────────────────────  
 for win, ov in [(w, o) for w in win\_sizes for o in overlaps]:  
 fpath = (f"{self.path}\\Participants\\{group}\_group\\P\_{pid}\\Features"  
 f"\\HRV\\HRV\_Time\_{win}\_{ov}.csv")  
 print(f"📁 Processing {fpath}")  
 if not os.path.exists(fpath):  
 print(" ❌ file not found"); continue  
  
 df = pd.read\_csv(fpath).dropna().reset\_index(drop=True)  
  
 df["y\_cls"] = df["Class"].map({"test": 1, "music": 0,  
 "breath": 0, "natural": 0})  
 feat\_cols = [c for c in df.columns  
 if c not in ["ID", "Group", "Time", "Class",  
 "Stress", "Fatigue", "y\_cls"]]  
  
 tscv = TimeSeriesSplit(n\_splits=5)  
  
 # ── classifiers ─────────────────────────────────────  
 cls\_models = {  
 "LogReg": LogisticRegression(max\_iter=1000, solver="liblinear",  
 class\_weight="balanced"),  
 "LDA": LinearDiscriminantAnalysis(),  
 "DT": DecisionTreeClassifier(random\_state=42),  
 "RF": RandomForestClassifier(n\_estimators=300, random\_state=42),  
 "XGB": XGBClassifier(n\_estimators=400, learning\_rate=0.05,  
 subsample=0.8, colsample\_bytree=0.8,  
 objective="binary:logistic",  
 eval\_metric="logloss",  
 use\_label\_encoder=False,  
 random\_state=42)  
 }  
  
 # ── regressors ──────────────────────────────────────  
 reg\_models = {  
 "LinReg": LinearRegression(),  
 "DT": DecisionTreeRegressor(random\_state=42),  
 "RF": RandomForestRegressor(n\_estimators=200, random\_state=42)  
 }  
  
 # ───────── CLASSIFICATION LOOP ───────────────────  
 for mname, est in cls\_models.items():  
 pipe = Pipeline([("sc", StandardScaler()), ("m", est)])  
  
 fold\_probs, fold\_true = [], []  
  
 for tr\_idx, val\_idx in tscv.split(df):  
 X\_tr, y\_tr = df.iloc[tr\_idx][feat\_cols], df.iloc[tr\_idx]["y\_cls"]  
 X\_val, y\_val = df.iloc[val\_idx][feat\_cols], df.iloc[val\_idx]["y\_cls"]  
  
 pipe.fit(X\_tr, y\_tr)  
 y\_prob = pipe.predict\_proba(X\_val)[:, 1] # prob class 1  
  
 fold\_probs.append(y\_prob)  
 fold\_true.append(y\_val.values)  
  
 y\_val\_all = np.concatenate(fold\_true)  
 y\_prob\_all = np.concatenate(fold\_probs)  
  
 cutoff, sc = self.\_best\_cutoff(y\_val\_all, y\_prob\_all)  
  
 cls\_metrics.append({  
 "Participant": pid, "Group": group,  
 "Window": win, "Overlap": ov, "Model": mname,  
 "Cutoff": round(cutoff, 3),  
 \*\*sc  
 })  
  
 # importance from full-data fit  
 pipe.fit(df[feat\_cols], df["y\_cls"])  
 imp = self.\_feature\_importance(pipe["m"], feat\_cols)  
 for feat, val in imp.items():  
 import\_rows.append({  
 "Participant": pid, "Group": group,  
 "Window": win, "Overlap": ov,  
 "Model": mname, "Feature": feat,  
 "Importance": val  
 })  
 imp.sort\_values().plot(kind="barh", figsize=(6, 3))  
 plt.title(f"P{pid} – {mname} ({win}s / ov={ov})")  
 plt.tight\_layout()  
 plt.savefig(plot\_dir / f"P{pid}\_{mname}\_{win}s\_{ov}\_CLS.png",  
 dpi=300); plt.close()  
  
 # ───────── REGRESSION LOOP ───────────────────────  
 for mname, est in reg\_models.items():  
 pipe = Pipeline([("sc", StandardScaler()), ("m", est)])  
  
 maes, rmses, r2s = [], [], []  
  
 for tr\_idx, val\_idx in tscv.split(df):  
 X\_tr, y\_tr = df.iloc[tr\_idx][feat\_cols], df.iloc[tr\_idx]["Stress"]  
 X\_val, y\_val = df.iloc[val\_idx][feat\_cols], df.iloc[val\_idx]["Stress"]  
  
 pipe.fit(X\_tr, y\_tr)  
 y\_hat = pipe.predict(X\_val)  
  
 maes.append(mean\_absolute\_error(y\_val, y\_hat))  
 rmses.append(np.sqrt(mean\_squared\_error(y\_val, y\_hat)))  
 r2s.append(r2\_score(y\_val, y\_hat))  
  
 reg\_metrics.append({  
 "Participant": pid, "Group": group,  
 "Window": win, "Overlap": ov, "Model": mname,  
 "MAE": np.mean(maes),  
 "RMSE": np.mean(rmses),  
 "R2": np.mean(r2s)  
 })  
  
 pipe.fit(df[feat\_cols], df["Stress"])  
 imp = self.\_feature\_importance(pipe["m"], feat\_cols)  
 for feat, val in imp.items():  
 import\_rows.append({  
 "Participant": pid, "Group": group,  
 "Window": win, "Overlap": ov,  
 "Model": mname, "Feature": feat,  
 "Importance": val  
 })  
 imp.sort\_values().plot(kind="barh", figsize=(6, 3))  
 plt.title(f"P{pid} – {mname} ({win}s / ov={ov})")  
 plt.tight\_layout()  
 plt.savefig(plot\_dir / f"P{pid}\_{mname}\_{win}s\_{ov}\_REG.png",  
 dpi=300); plt.close()  
 # ── end window/overlap loop ─────────────────────────────  
  
 # build sheets  
 cls\_df = (pd.DataFrame(cls\_metrics).sort\_values("F1", ascending=False))  
  
 reg\_df = (pd.DataFrame(reg\_metrics)  
 .sort\_values("RMSE", ascending=True))  
 imp\_df = (pd.DataFrame(import\_rows)  
 .sort\_values("Importance", ascending=False))  
  
 with pd.ExcelWriter(out\_xlsx, engine="openpyxl") as xlw:  
 cls\_df.to\_excel(xlw, sheet\_name="Classification", index=False)  
 reg\_df.to\_excel(xlw, sheet\_name="Regression", index=False)  
 imp\_df.to\_excel(xlw, sheet\_name="Importance", index=False)  
  
 print(f"✅ Excel + plots finished for P{pid}")  
  
 if not cls\_df.empty:  
 summary\_rows.append({\*\*cls\_df.iloc[0].to\_dict(), "Sheet": "Classification"})  
 if not reg\_df.empty:  
 summary\_rows.append({\*\*reg\_df.iloc[0].to\_dict(), "Sheet": "Regression"})  
 if not imp\_df.empty:  
 summary\_rows.append({\*\*imp\_df.iloc[0].to\_dict(), "Sheet": "Importance"})  
  
 # ── master summary ──────────────────────────────────────────  
 summary\_df = pd.DataFrame(summary\_rows)  
 main\_cols = ["Sheet", "Participant", "Group", "Window", "Overlap", "Model"]  
 summary\_df = summary\_df[main\_cols + [c for c in summary\_df.columns  
 if c not in main\_cols]]  
 master\_xlsx = out\_root / "ML\_Best\_Summary.xlsx"  
 summary\_df.to\_excel(master\_xlsx, index=False)  
 print(f"🏆 Overall best summary saved to {master\_xlsx}")  
  
 def ML\_models\_all(self, n\_repeats=9, plot=False):  
 # Define window sizes and overlap percentages  
 window\_sizes = [5, 10, 30, 60]  
 overlaps = [0.0, 0.5]  
  
 # Define base models without hyperparameter tuning  
 base\_models = {  
 'DecisionTree': DecisionTreeClassifier(random\_state=42),  
 'RandomForest': RandomForestClassifier(random\_state=42),  
 'XGBoost': XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss', random\_state=42)  
 }  
  
 # Define parameter grids for hyperparameter tuning  
 param\_grids = {  
 'DecisionTree': {  
 'max\_depth': [None, 10, 20],  
 'min\_samples\_split': [2, 5, 10]  
 },  
 'RandomForest': {  
 'n\_estimators': [100, 200],  
 'max\_depth': [None, 10, 20],  
 'min\_samples\_split': [2, 5]  
 },  
 'XGBoost': {  
 'n\_estimators': [100, 200],  
 'max\_depth': [3, 6],  
 'learning\_rate': [0.01, 0.1]  
 }  
 }  
  
 # Load participant IDs from management file  
 participants\_csv = os.path.join(self.path, 'Participants', 'participation management.csv')  
 participants = pd.read\_csv(participants\_csv)  
 all\_ids = participants['code'].dropna().astype(int).unique()  
  
 results = []  
  
 for repeat in range(n\_repeats):  
 # Iteration 1: Split 80% train / 20% test by participant IDs  
 train\_ids, test\_ids = train\_test\_split(  
 all\_ids, test\_size=0.2, random\_state=42 + repeat  
 )  
  
 # Prepare best window info placeholders for each model  
 best\_ws = {name: {'window': None, 'overlap': None, 'f1': -np.inf} for name in base\_models}  
  
 # Iteration 2: Select optimal window size and overlap via 5-fold GroupKFold  
 for ws in window\_sizes:  
 for ov in overlaps:  
 file\_path = fr'{self.path}\Participants\Dataset\Dataset\_window{ws}s\_{int(ov \* 100)}.csv'  
 if not os.path.exists(file\_path):  
 continue  
  
 df = pd.read\_csv(file\_path).dropna().reset\_index(drop=True)  
 df\_train = df[df['ID'].isin(train\_ids)]  
 feature\_cols = [c for c in df.columns if c not in ['Time', 'ID', 'Group', 'Class']]  
 y = df\_train['Class'].map({'test': 1, 'music': 0, 'breath': 0, 'natural': 0})  
 groups = df\_train['ID']  
  
 gkf = GroupKFold(n\_splits=5)  
 for name, model in base\_models.items():  
 f1\_scores = []  
 for tr\_idx, val\_idx in gkf.split(df\_train, y, groups):  
 X\_tr = df\_train.iloc[tr\_idx][feature\_cols]  
 y\_tr = y.iloc[tr\_idx]  
 X\_val = df\_train.iloc[val\_idx][feature\_cols]  
 y\_val = y.iloc[val\_idx]  
  
 model.fit(X\_tr, y\_tr)  
 y\_pred = model.predict(X\_val)  
 f1\_scores.append(f1\_score(y\_val, y\_pred, zero\_division=0))  
  
 mean\_f1 = np.mean(f1\_scores)  
 if mean\_f1 > best\_ws[name]['f1']:  
 best\_ws[name].update({'window': ws, 'overlap': ov, 'f1': mean\_f1})  
  
 # Iteration 3: Tune hyperparameters for selected window and overlap  
 tuned\_models = {}  
 for name, base\_model in base\_models.items():  
 ws = best\_ws[name]['window']  
 ov = best\_ws[name]['overlap']  
 file\_path = fr'{self.path}\Participants\Dataset\Dataset\_window{ws}s\_{int(ov \* 100)}.csv'  
 df = pd.read\_csv(file\_path).dropna().reset\_index(drop=True)  
 df\_train = df[df['ID'].isin(train\_ids)]  
 feature\_cols = [c for c in df.columns if c not in ['Time', 'ID', 'Group', 'Class']]  
 X\_tr = df\_train[feature\_cols]  
 y\_tr = df\_train['Class'].map({'test': 1, 'music': 0, 'breath': 0, 'natural': 0})  
 groups = df\_train['ID']  
  
 gkf = GroupKFold(n\_splits=5)  
 grid = GridSearchCV(  
 base\_model,  
 param\_grid=param\_grids[name],  
 cv=gkf,  
 scoring='f1',  
 n\_jobs=-1  
 )  
 grid.fit(X\_tr, y\_tr, groups=groups)  
 tuned\_models[name] = grid.best\_estimator\_  
  
 # Iteration 4: Evaluate tuned models on the external test set  
 for name, model in tuned\_models.items():  
 ws = best\_ws[name]['window']  
 ov = best\_ws[name]['overlap']  
 file\_path = fr'{self.path}\Participants\Dataset\Dataset\_window{ws}s\_{int(ov \* 100)}.csv'  
 df = pd.read\_csv(file\_path).dropna().reset\_index(drop=True)  
 df\_test = df[df['ID'].isin(test\_ids)]  
 feature\_cols = [c for c in df.columns if c not in ['Time', 'ID', 'Group', 'Class']]  
 X\_te = df\_test[feature\_cols]  
 y\_te = df\_test['Class'].map({'test': 1, 'music': 0, 'breath': 0, 'natural': 0})  
  
 y\_pred = model.predict(X\_te)  
 results.append({  
 'Repeat': repeat + 1,  
 'Model': name,  
 'Window (s)': ws,  
 'Overlap (%)': int(ov \* 100),  
 'Accuracy': accuracy\_score(y\_te, y\_pred),  
 'Precision': precision\_score(y\_te, y\_pred, zero\_division=0),  
 'Recall': recall\_score(y\_te, y\_pred, zero\_division=0),  
 'F1': f1\_score(y\_te, y\_pred, zero\_division=0)  
 })  
  
 # Optional: save feature importance plots  
 if plot and hasattr(model, 'feature\_importances\_'):  
 out\_dir = fr"{self.path}\Participants\Dataset\ML\{name}\Repeat{repeat + 1}"  
 os.makedirs(out\_dir, exist\_ok=True)  
 imp = pd.Series(model.feature\_importances\_, index=feature\_cols).sort\_values(ascending=False)  
 fig, ax = plt.subplots(figsize=(8, 5))  
 imp.plot.bar(ax=ax)  
 ax.set\_title(f"{name} Importances (Repeat {repeat + 1}, {ws}s/{int(ov \* 100)}%)")  
 fig.tight\_layout()  
 fig.savefig(os.path.join(out\_dir, f"{name}\_Importance\_R{repeat + 1}.png"))  
 plt.close(fig)  
  
 # Save summary of results to CSV  
 results\_df = pd.DataFrame(results)  
 out\_path = fr'{self.path}\Participants\Dataset\ML\NestedCV\_Results.csv'  
 results\_df.to\_csv(out\_path, index=False)  
 print(f"Nested CV complete. Results saved to {out\_path}")  
  
 def Cor(self):  
 # ── 1. load data ──────────────────────────────────────────────  
 stress\_all = pd.read\_excel(r"C:\Users\e3bom\Desktop\Human Bio Signals Analysis\Participants\All\_HRV\_stress\_30s.xlsx"  
 )  
  
 hrv\_feats = [  
 "HRV\_MeanNN", "HRV\_SDNN", "HRV\_RMSSD",  
 "HRV\_CVNN", "HRV\_pNN20", "HRV\_pNN50",  
 ]  
  
 # ── 2. helper to build one full scatter-matrix figure ─────────  
 def \_plot(df, y\_col, title, out\_png):  
 n\_feats = len(hrv\_feats)  
 fig, axes = plt.subplots(n\_feats, 1, figsize=(6, 3 \* n\_feats), sharey=True)  
  
 if not isinstance(axes, (list, np.ndarray)):  
 axes = [axes]  
  
 for ax, feat in zip(axes, hrv\_feats):  
 # coloured dots  
 sns.scatterplot(  
 data=df, x=feat, y=y\_col, hue="Group",  
 palette="Set2", s=40, ax=ax, legend=False  
 )  
 # regression line  
 sns.regplot(  
 data=df, x=feat, y=y\_col,  
 scatter=False, ci=95, line\_kws=dict(lw=1.5, alpha=0.8), ax=ax  
 )  
 # Pearson r  
 r, p = linregress(df[feat], df[y\_col])[:2]  
 ax.set\_title(f"{feat} (r = {r:.2f}, p = {p:.3g})")  
 ax.set\_xlabel("Mean value")  
 ax.grid(True)  
  
 axes[0].set\_ylabel(y\_col)  
 fig.suptitle(title, fontsize=14)  
 fig.tight\_layout(rect=[0, 0, 1, 0.97])  
 fig.savefig(out\_png, dpi=300)  
 plt.close(fig) # keep memory footprint small  
  
 # ── 3. figure #1 – raw stress ─────────────────────────────────  
 \_plot(  
 stress\_all,  
 y\_col="Stress",  
 title="Stress vs HRV features (raw)",  
 out\_png=r"C:\Users\e3bom\Desktop\Human Bio Signals Analysis\Participants\All\_particapents\stress\_HRV\_scatter.png"  
 )  
  
 # ── 4. create z-scored Stress inside each participant ─────────  
 stress\_all["Stress\_z"] = (  
 stress\_all.groupby("ID")["Stress"]  
 .transform(lambda s: (s - s.mean()) / s.std(ddof=0))  
 )  
  
 # ── 5. figure #2 – normalized stress ──────────────────────────  
 \_plot(  
 stress\_all,  
 y\_col="Stress\_z",  
 title="Stress (z-score within participant) vs HRV features",  
 out\_png=r"C:\Users\e3bom\Desktop\Human Bio Signals Analysis\Participants\All\_particapents\stress\_HRV\_scatter\_norm.png")  
  
  
 def Analysis\_per\_particitenpt(self):  
 dataset\_path = f'{self.path}\Participants\Dataset\Dataset.csv'  
 Participants\_path = f'{self.path}\Participants\participation management.xlsx'  
 Participants\_df = pd.read\_excel(Participants\_path, header=1)  
 Participants\_df = Participants\_df.dropna(axis=1, how='all')  
 Participants\_df['code'] = pd.to\_numeric(Participants\_df['code'], errors='coerce').astype('Int64')  
 TotalCorr=pd.DataFrame()  
 for j, row in Participants\_df.iterrows():  
 ID = row['code']  
 Group = row['Group']  
 print(ID)  
 # ID = 9  
 # Group = 'music'  
 # directory = fr'{self.path}\Participants\{Group}\_group\P\_{ID}'  
 # dataParticipent\_path = fr'{directory}\data\_{ID}.csv'  
 data=pd.read\_csv(dataset\_path)  
 data=data[data['participant']==ID]  
 data=data.drop(columns=['participant'])  
 data=data.drop(columns=['Part'])  
 # data.replace('-', np.nan, inplace=True)  
 # Replace invalid entries like 'nane' with NaN  
 data.replace('nane', np.nan, inplace=True)  
 # Convert all columns to numeric where possible, forcing errors to NaN  
 data = data.apply(pd.to\_numeric, errors='coerce')  
  
 # sns.pairplot(data)  
 # plt.suptitle("Scatter Plot Matrix of Features vs. Stress Report", y=1.02)  
 # plt.show()  
  
 # g = sns.pairplot(data, diag\_kind="kde")  
 # g.map\_lower(sns.kdeplot, levels=4, color=".2")  
 # g\_path=fr'{directory}\pairplot\_{ID}.png'  
 # plt.savefig(g\_path, dpi=300, bbox\_inches='tight')  
 # plt.show()  
  
 # Correlation matrix  
 correlation\_matrix = data.corr()  
 Corr\_path = fr'{self.path}\Participants\{Group}\_group\P\_{ID}\Corr\_{ID}.csv'  
 correlation\_matrix.to\_csv(Corr\_path)  
 first\_row\_corr = correlation\_matrix.iloc[0, :]  
 features\_df = pd.DataFrame(first\_row\_corr).T # Transpose to match participant as a row  
 features\_df['Participant\_ID'] = ID # Add participant ID to track  
 cols = ['Participant\_ID'] + [col for col in features\_df if col != 'Participant\_ID']  
 features\_df = features\_df[cols]  
 # Concatenate with TotalCorr to accumulate results  
 TotalCorr = pd.concat([TotalCorr, features\_df], axis=0, ignore\_index=True)  
 # plt.figure(figsize=(10, 8))  
 # sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f')  
 # plt.title('Correlation Matrix')  
 # plt.savefig(fr'{directory}\Correlation Matrix\_{ID}.png', dpi=300, bbox\_inches='tight')  
 # plt.show()  
  
 # X = data[['ECG\_Rate\_Mean', 'HRV\_MeanNN', 'HRV\_SDNN', 'HRV\_RMSSD', 'HRV\_pNN50', 'HRV\_pNN20']]  
 # y = data[['Stress Report']]  
 #  
 #  
 # # Add a constant to the model (intercept)  
 # model = LinearRegression()  
 # sfs = SequentialFeatureSelector(model, n\_features\_to\_select=3, cv=5, scoring='neg\_mean\_squared\_error')  
 # sfs.fit(X, y)  
 # Selected\_Features = sfs.get\_feature\_names\_out()  
 # X\_selected = X[Selected\_Features]  
 # # Add a constant (intercept) to the model  
 # X\_selected\_with\_const = sm.add\_constant(X\_selected)  
 # X\_with\_const = sm.add\_constant(X)  
 #  
 # # Fit the model with statsmodels  
 # model\_selected = sm.OLS(y, X\_selected\_with\_const).fit()  
 # model\_full = sm.OLS(y, X\_with\_const).fit()  
 #  
 # # Print the summary of the selected model  
 # print(model\_selected.summary())  
 #  
 # # Print the summary of the full model  
 # print(model\_full.summary())  
  
  
 # Create a 3D scatter plot  
 # fig = plt.figure(figsize=(10, 8))  
 # ax = fig.add\_subplot(111, projection='3d')  
 #  
 # # Plot the selected features against y  
 # ax.scatter(X\_selected['ECG\_Rate\_Mean'], X\_selected['HRV\_MeanNN'], X\_selected['HRV\_SDNN'], c=y,  
 # cmap='viridis', marker='o')  
 #  
 # # Set labels and title  
 # ax.set\_xlabel('ECG\_Rate\_Mean')  
 # ax.set\_ylabel('HRV\_MeanNN')  
 # ax.set\_zlabel('HRV\_SDNN')  
 # ax.set\_title('3D Scatter Plot: Selected Features vs Target')  
 #  
 # # Show the plot  
 # plt.show()  
  
 # summary\_str = model.summary().as\_text()  
 # # Split the summary string into lines  
 # summary\_lines = summary\_str.split('\n')  
 # # Convert summary lines into a DataFrame  
 # summary\_df = pd.DataFrame({'Summary': summary\_lines})  
 # # Save the DataFrame to a CSV file  
 # summary\_df.to\_csv(fr'{directory}\Reggresion\_summary\_{ID}.png', index=False)  
 # # Print out the results  
 datasetCorr\_path = f'{self.path}\Participants\Dataset\Corr\_all.csv'  
 TotalCorr.to\_csv(datasetCorr\_path)  
  
 # Example of loading your dataset (replace with your data)  
 # data = pd.read\_csv('your\_data.csv')  
  
 # Here, 'dependent\_variable' is the outcome variable, 'fixed\_effects\_variable' is the fixed effect,  
 # and 'random\_effect\_grouping' is the random effect grouping (e.g., participant IDs).  
  
 # Mixed Linear Model  
 # Replace 'dependent\_variable' with the column name of the outcome,  
 # 'fixed\_effects\_variable' with your fixed effect predictor, and  
 # 'random\_effect\_grouping' with the grouping factor for the random effect.  
 # Load your data  
  
  
 def Linear\_Mixed\_Effects\_Models(self):  
  
 # Load the dataset  
 data = pd.read\_csv(r'D:\Human Bio Signals Analysis\Participants\Dataset\Dataset\_1\_EDA.csv')  
  
 # Perform mean imputation for all columns except 'participant' and 'Part'  
 data\_imputed = data.copy()  
 numeric\_columns = data\_imputed.columns.difference(['participant', 'Part'])  
 data\_imputed[numeric\_columns] = data\_imputed[numeric\_columns].fillna(data\_imputed[numeric\_columns].mean())  
  
 # Fit the Linear Mixed-Effects Model (LMM)  
 # Using 'Stress\_Report' as the dependent variable and 'participant' as the random effect  
 # The other physiological signals will be the fixed effects  
  
 model = smf.mixedlm(  
 "Stress\_Report ~ ECG\_Rate\_Mean + HRV\_MeanNN + HRV\_SDNN + HRV\_RMSSD + HRV\_pNN50 + HRV\_pNN20 + "  
 "HRV\_VHF + HRV\_VLF + HRV\_LF + HRV\_HF + HRV\_LFHF + HRV\_LFn + HRV\_HFn + HRV\_LnHF + HRV\_TP + "  
 "HRV\_ShanEn + Diaph\_RSP\_Rate\_Mean + Diaph\_BRV + Diaph\_BRavNN + Diaph\_RSP\_Phase\_Duration\_Expiration + "  
 "Diaph\_RSP\_Phase\_Duration\_Inspiration + Diaph\_RSP\_Phase\_Duration\_Ratio + Diaph\_RSP\_RVT + "  
 "Diaph\_RSP\_Symmetry\_PeakTrough + Diaph\_RSP\_Symmetry\_RiseDecay + Chest\_RSP\_Rate\_Mean + Chest\_BRV + "  
 "Chest\_BRavNN + Chest\_RSP\_Phase\_Duration\_Expiration + Chest\_RSP\_Phase\_Duration\_Inspiration + "  
 "Chest\_RSP\_Phase\_Duration\_Ratio + Chest\_RSP\_RVT + Chest\_RSP\_Symmetry\_PeakTrough + Chest\_RSP\_Symmetry\_RiseDecay + "  
 "EDA\_Tonic\_Mean + EDA\_Phasic\_Mean + SCR\_Peaks\_Count + SCR\_Amplitude\_Mean",  
 data\_imputed,  
 groups=data\_imputed["participant"])  
  
 # Fit the model  
 result = model.fit()  
  
 # Print the model summary  
 print(result.summary())