```python
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

#%%
#                                                    "TEST ON THE INDEX OF DATA"
UDP_PORT = 49000

import socket
import struct
import datetime
import pandas as pd
import csv


counter = 0

def DecodeDataMessage(message):
  # Message consists of 4 byte type and 8 times a 4byte float value.
  # Write the results in a python dict.
  values = {}
  typelen = 4
  type = int.from_bytes(message[0:typelen], byteorder='little')
  data = message[typelen:]
  dataFLOATS = struct.unpack("<ffffffff",data)

  if type == 1:

      global counter
      counter = counter + 1

      currentDT = datetime.datetime.now()
      stamp = currentDT.strftime('%m/%d/%Y %H:%M:%S.%f')

      values['timestamp'] = stamp
      values['real, time']=dataFLOATS[0]
      values["totl, time"]=dataFLOATS[1]
      #values["missn, time"]=dataFLOATS[2]
```

```
    #values["timer, time"]=dataFLOATS[3]
    #values["zulu, time"]=dataFLOATS[4]
    #values["local, time"]=dataFLOATS[5]
    #values["hobbs, time"]=dataFLOATS[6]

elif type == 3:
    values['Vind, kias']=dataFLOATS[0] # airspeed indicator
    #values["Vind, keas"]=dataFLOATS[1]
    #values["Vtrue, ktas"]=dataFLOATS[2]
    #values["Vtrue, ktgs"]=dataFLOATS[3]
    #values["Vind, mph"]=dataFLOATS[4]
    #values["Vtrue, mphas"]=dataFLOATS[5]
    #values["Vtrue, mphgs"]=dataFLOATS[6]

elif type == 4:
    #values["Mach, ratio"]=dataFLOATS[0]
    values['VVI, fpm']=dataFLOATS[2] # VSI indicator
    #values["Gload, norml"]=dataFLOATS[2]
    #values["Gload, axial"]=dataFLOATS[3]
    #values["Gload, side"]=dataFLOATS[4]

elif type == 8:
    values['elev, yoke1']=dataFLOATS[0] # pull +1 , push -1
    values['ailrn, yoke1']=dataFLOATS[1] # right +1, left -1
    values['ruddr, yoke1']=dataFLOATS[2] # rudder right +1, left -1

elif type == 13:
    values['trim, elev']=dataFLOATS[0] # showes when the user apply pitch trim
    values['trim, ailrn']=dataFLOATS[1] # showes when the users apply aileron trim
    #values["trim, rudder"]=dataFLOATS[2]
    values['flap, handl']=dataFLOATS[3] # flap position,  0, 0.333, 1
    #values["flap position"]=dataFLOATS[4]
    #values["slat, ratio"]=dataFLOATS[5]

elif type == 17:
    values['pitch, deg']=dataFLOATS[0] # pitch deg - atitude indicator
    values['roll, deg']=dataFLOATS[1] # roll degree - atitude indicator
    #values["heading, true"]=dataFLOATS[2]
    values["heading mag"]=dataFLOATS[3] # heading indicator

elif type == 20:
```

```
        #values["lat, deg"]=dataFLOATS[0]
        #values["lon, deg"]=dataFLOATS[1]
        #values["alt, MSL"]=dataFLOATS[2]
        values['alt, ind']=dataFLOATS[5] # altitude indicator

    elif type == 21:
        values['X, m']=dataFLOATS[0] # X
        values['Y, m']=dataFLOATS[1] # Y
        values['Z, m']=dataFLOATS[2] # Z
        #values["vX, m/s"]=dataFLOATS[3]
        #values["vY, m/s"]=dataFLOATS[4]
        #values["vZ, m/s"]=dataFLOATS[5]

    elif type == 37:
        values['rpm n, engin']=dataFLOATS[0] # RPM

    #else:
        #print("  Type ", type, " not implemented: ",dataFLOATS)
    return values

def DecodePacket(data):
    # Packet consists of 5 byte header and multiple messages.
    valuesout = {}
    headerlen = 5
    header = data[0:headerlen]
    messages = data[headerlen:]
    if(header==b'DATA*'):
        # Divide into 36 byte messages
        messagelen = 36
        for i in range(0,int((len(messages))/messagelen)):
            message = messages[(i*messagelen) : ((i+1)*messagelen)]
            values = DecodeDataMessage(message)
            valuesout.update( values )
    else:
        print("Packet type not implemented. ")
        print("  Header: ", header)
        print("  Data: ", messages)
    return valuesout

def main():
```

```
# Open a Socket on UDP Port 49000
UDP_IP = '192.168.1.100'
sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP

sock.bind((UDP_IP, UDP_PORT))

while True:
  # Receive a packet
  data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes

  # Decode the packet. Result is a python dict (like a map in C) with values from X-Plane.
  # Example:
  # {'latitude': 47.72798156738281, 'longitude': 12.434000015258789,
  #   'altitude MSL': 1822.67, 'altitude AGL': 0.17, 'speed': 4.11,
  #   'roll': 1.05, 'pitch': -4.38, 'heading': 275.43, 'heading2': 271.84}
  values = DecodePacket(data)
  print(values['timestamp'])
  #(pd.DataFrame.from_dict(data=values, orient='index').to_csv('dict_file.csv', header=False))
  print (counter)

  #' N03-E-B-LDTS
  #' N03-E-A-LTDS
  #' N04-E-B-LTDS
  #' N04-E-A-LSDT
  #' N05-E-B-LTDS
  #' N05-E-A-TSDL
  #' N06-E-B-LTDS
  #' N06-E-A-LDTS

  with open('TEST-A.csv','a') as f1:
      writer=csv.writer(f1, delimiter='\t',lineterminator='\n')

      row =[]

      for key, value in values.items():

          if counter == 1:
              row.append(key)

          else:
```

```
                row.append(value)

        writer.writerow(row)

if __name__ == '__main__':
  main()
#%%

#%% DISSERTATION ==> For EYE MOVEMENTS
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# this import data from tobii pro glass 2 as .tsv file.
# then create timestamps and interploate data on a 100 millisecond time interval
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import datetime as dt
import time
import math


filename_eye = 'N04-C-B-LDST.tsv'
fliename_flight = 'N04-C-B-LDST.csv'

df_eye = pd.read_csv(filename_eye, encoding = 'utf-16', delimiter='\t', low_memory=False,
                     parse_dates = {'time':['Recording date','Recording start time']})

df_eye['Recording timestamp'] = pd.to_timedelta(df_eye['Recording timestamp'], unit ='ms')
df_eye['time'] = df_eye['time'] + df_eye['Recording timestamp']

# in some cases, the eye trackers loges more than one time
```

```
#df_eye = df_eye.drop_duplicates(subset=['Recording timestamp'], keep='first').reset_index(drop=True)
#df_eye.to_csv('delete duplicate.csv', encoding='utf-8')

df_eye['Event']

# ********************************************************************************************
# I do not know why I rounded time here !
#df_eye['time'].dt.round('1s')
# ********************************************************************************************
df_eye['scenario duration'] = np.nan
df_eye['scenario duration'] = pd.to_timedelta(df_eye['scenario duration'])

list_extra = []
list_extra = df_eye['Event'][df_eye['Event'].notnull()].unique()
list_extra = list_extra.tolist()
list_extra.extend(['X', 'XX'])

list_events = ['Landing', 'Level Turn', 'Level Flight', 'Descent']

print('++++++++++++++++++++++ The Tobii Tags ++++++++++++++++++++++')
print(list_extra)
print('++++++++++++++++++++++ Desired Tags ++++++++++++++++++++++')
print(list_events)

for i in list_events:
    if i in list_extra:
        list_extra.remove(i)

print('++++++++++++++++++++++ Unwanted Tags ++++++++++++++++++++++')
print(list_extra)

for i in list_extra:
    df_eye['Event'] = df_eye['Event'].str.replace(i, '')

df_eye['Event'] = df_eye['Event'].fillna('')
df_eye['Event'] = df_eye['Event'].replace({'': np.NaN})


print('++++++++++++++++++++++ The Outcome ++++++++++++++++++++++')
print(df_eye['Event'][df_eye['Event'].notnull()].unique())
print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
```

```
df_eye = df_eye[~((df_eye['Recording timestamp'].duplicated(keep=False)) & (df_eye['Event'].isnull()))].reset_index(drop=True)

df_eye['Event'].astype(str)
flag_event = False

zero_point = 0

#fixing events
for index, row in df_eye.iterrows():
    if  len(df_eye['Event']) == index + 1:
        break

    elif pd.isnull(df_eye['Event'].iloc[index]) == True and pd.isnull(df_eye['Event'].iloc[index+1]) == True:
        flag_event = False

    elif pd.isnull(df_eye['Event'].iloc[index]) == False and pd.isnull(df_eye['Event'].iloc[index+1]) == True and flag_event == Fal

        if index == 0:
            if pd.isnull(df_eye['Event'].iloc[index]) == False and pd.isnull(df_eye['Event'].iloc[index+1]) == True:
                zero_point = index
        else:

            if pd.isnull(df_eye['Event'].iloc[index]) == False and pd.isnull(df_eye['Event'].iloc[index+1]) == True and pd.isnull(d
                zero_point = index

        #df_eye.set_value(index+1, 'Event', df_eye['Event'].iloc[index])
        df_eye.at[index+1, 'Event'] = df_eye['Event'].iloc[index]

#         time.sleep(0.001)
#         print(zero_point, '++++++++++++++++++++', df_eye['Event'].iloc[index])
#         print('========================================')
#         print(df_eye['Recording timestamp'].iloc[zero_point])
#         print('++++++++++++++++++++++++++++++++++++++++')
#         print(df_eye['Recording timestamp'].iloc[index])
#         print('****************************************')

        df_eye.at[index, 'scenario duration'] = df_eye['Recording timestamp'].iloc[index] - df_eye['Recording timestamp'].iloc[zero

        flag_event = False
```

```
    elif pd.isnull(df_eye['Event'].iloc[index]) == False and pd.isnull(df_eye['Event'].iloc[index+1]) == False:

        df_eye.at[index, 'scenario duration'] = df_eye['Recording timestamp'].iloc[index]
        df_eye.at[index+1, 'scenario duration'] = df_eye['Recording timestamp'].iloc[index+1]

        df_eye.at[index, 'scenario duration'] = df_eye['Recording timestamp'].iloc[index] - df_eye['Recording timestamp'].iloc[zero
        df_eye.at[index+1, 'scenario duration'] = df_eye['Recording timestamp'].iloc[index+1] - df_eye['Recording timestamp'].iloc[

#         time.sleep(5)
#         print(zero_point, '++++++++++++++++++++', df_eye['Event'].iloc[index])
#         print('========================================')
#         print(df_eye['Recording timestamp'].iloc[zero_point])
#         print('++++++++++++++++++++++++++++++++++++++++')
#         print(df_eye['Recording timestamp'].iloc[index])
#         print('****************************************')

        zero_point = 0
        flag_event = True

#df_eye.to_csv('df_at_test_2.csv', encoding='utf-8')

#df_eye['scenario duration'].iloc[1665:1675]
#df_eye['Recording timestamp'].iloc[1665:1675]
#print(df_to_append.head(10))
#print(df_to_append.tail(10))

df_eye['Number'] = df_eye['Participant name'].str[1:3]
df_eye['Group'] = np.where(df_eye['Participant name'].str[4:5]=='E', 'Experimental', 'Control')
df_eye['Training'] = np.where(df_eye['Participant name'].str[6:7]=='A', 'Before Treatment', 'After Treatment')

# if this is fixation or not ( 1 is yes and 0 is no)
df_eye['fixation'] = 0

df_eye = df_eye[['time', 'Recording timestamp', 'scenario duration',  'Group', 'Number', 'Training', 'Event', 'Eye movement type',
                 'Gaze event duration', 'Pupil diameter left', 'Pupil diameter right',
              'AOI hit [XXX - AI]', 'AOI hit [XXX - ALT]', 'AOI hit [XXX - ASI]',
       'AOI hit [XXX - CONT]', 'AOI hit [XXX - GAG]', 'AOI hit [XXX - GPS]',
       'AOI hit [XXX - HI]', 'AOI hit [XXX - NAV]', 'AOI hit [XXX - OSW]',
       'AOI hit [XXX - RT]', 'AOI hit [XXX - TAC]', 'AOI hit [XXX - TC]',
       'AOI hit [XXX - VSI]']]
```

```
#df_interpolated['AOI'] = df_interpolated.apply(func=lambda row: AOI(row, 14, 22), axis=1)

#df_eye = df_eye.set_index(['time'])

df_eye['sum'] = df_eye.iloc[:, df_eye.columns.get_loc('AOI hit [XXX - AI]'):df_eye.columns.get_loc('AOI hit [XXX - VSI]')+1].apply(

df_eye = df_eye[['time','Recording timestamp', 'scenario duration', 'Group', 'Number', 'Training', 'Event', 'Eye movement type', 'f
                 'Gaze event duration', 'Pupil diameter left', 'Pupil diameter right',
              'AOI hit [XXX - AI]', 'AOI hit [XXX - ALT]', 'AOI hit [XXX - ASI]',
       'AOI hit [XXX - CONT]', 'AOI hit [XXX - GAG]', 'AOI hit [XXX - GPS]',
       'AOI hit [XXX - HI]', 'AOI hit [XXX - NAV]', 'AOI hit [XXX - OSW]',
       'AOI hit [XXX - RT]', 'AOI hit [XXX - TAC]', 'AOI hit [XXX - TC]',
       'AOI hit [XXX - VSI]']]


df_eye = df_eye.dropna(subset=['Event'])

# keeping the desired columns with desired order
df_eye = df_eye[['time','Recording timestamp', 'scenario duration', 'Group', 'Number', 'Training', 'Event', 'Eye movement type', 'f
                          'AOI hit [XXX - AI]', 'AOI hit [XXX - ALT]', 'AOI hit [XXX - ASI]',
       'AOI hit [XXX - CONT]', 'AOI hit [XXX - GAG]', 'AOI hit [XXX - GPS]',
       'AOI hit [XXX - HI]', 'AOI hit [XXX - NAV]', 'AOI hit [XXX - OSW]',
       'AOI hit [XXX - RT]', 'AOI hit [XXX - TAC]', 'AOI hit [XXX - TC]',
       'AOI hit [XXX - VSI]']]

df_eye.columns = ['time', 'timestamp', 'scenario time', 'Group', 'Number', 'Training', 'Event', 'Eye movement type', 'fixation', 'G
                      'AI', 'ALT', 'ASI',
                      'CONT', 'GAG', 'GPS',
                      'HI', 'NAV', 'OSW',
                      'RT', 'TAC', 'TC',
                      'VSI']

AOI_list =    ['AI',           'ALT',           'HI',          'TC',          'ASI' ,        'VSI',             'TAC',


# finding the column (AOI) that partcipants looked at
df_aoi = df_eye.iloc[:, df_eye.columns.get_loc('AI'):df_eye.columns.get_loc('VSI')+1].idxmax(axis=1)

# adding the AOI to the dataframe / AOI is the visited Area of Interst
df_eye['AOI'] =  df_aoi
```

```
# if the eye movement is not fixation, then AOI is 0
df_eye['AOI'] = df_eye.apply(lambda x: np.nan if x['Eye movement type'] != 'Fixation' else x['AOI'], axis=1)
df_eye['AOI'] = df_eye.apply(lambda x: np.nan if x['Eye movement type'] == 'Fixation' and x['sum'] == 0 else x['AOI'], axis=1)

#print('++++++++++++++++++++++++++++++++++++++++')
#print(df_eye_data.iloc[72460:72470, : ])
#print('++++++++++++++++++++++++++++++++++++++++')

df_eye_landing = df_eye.loc[df_eye['Event'] == 'Landing'].reset_index(drop=True)
df_eye_levelturn = df_eye.loc[df_eye['Event'] == 'Level Turn'].reset_index(drop=True)
df_eye_levelflight = df_eye.loc[df_eye['Event'] == 'Level Flight'].reset_index(drop=True)
df_eye_descent = df_eye.loc[df_eye['Event'] == 'Descent'].reset_index(drop=True)

df_eye_landing.name = 'df_eye_landing'
df_eye_levelturn.name = 'df_eye_levelturn'
df_eye_levelflight.name = 'df_eye_levelflight'
df_eye_descent.name = 'df_eye_descent'

#[df_eye_landing, df_eye_levelturn, df_eye_levelflight, df_eye_descent]

for scenario in [df_eye_levelturn]:
    #print (scenario.head(5))


    # defining two variables for counting number of fixation , fixation duration
    fixation_count = 0
    fixation_duration = 0
    # defining a variable as counter of rows of dataframe
    row_index = 0

    columns = ['Index','timestamp', 'scenario time', 'AOI', 'Duration']
    index = range(0,0)
    df_fixation = pd.DataFrame(index = index, columns = columns)


    # It counts number of fixation, fixation duration
    for index, row in scenario.iterrows():
        #print(index, row)
        counter_order = row_index
```

```
# fixing the AOI
while (counter_order != len(scenario) - 1  and scenario['Eye movement type'].iloc[row_index-1] == 'Fixation' and
       scenario['Eye movement type'].iloc[counter_order] == 'Fixation' and
       scenario['sum'].iloc[counter_order]  == 0 and
       scenario['sum'].iloc[row_index - 1 ] == 1 and
       scenario['Gaze event duration'].iloc[row_index - 1] == scenario['Gaze event duration'].iloc[counter_order]):

    print('===> forward', counter_order)
    scenario['AOI'].iloc[counter_order] = scenario['AOI'].iloc[row_index -1]
    scenario['sum'].iloc[counter_order] = 1
    scenario[scenario['AOI'].iloc[row_index -1]].iloc[counter_order] = scenario[scenario['AOI'].iloc[row_index -1]].iloc[ro
    counter_order = counter_order + 1

counter_reverse = row_index
while (counter_reverse != 0 and scenario['Eye movement type'].iloc[row_index + 1] == 'Fixation' and
       scenario['Eye movement type'].iloc[counter_reverse ] == 'Fixation' and
       scenario['sum'].iloc[counter_reverse]  == 0 and
       scenario['sum'].iloc[row_index + 1] == 1 and
       scenario['Gaze event duration'].iloc[row_index + 1] == scenario['Gaze event duration'].iloc[row_index]):

    print('===> backward', counter_reverse)
    scenario['AOI'].iloc[counter_reverse] = scenario['AOI'].iloc[row_index + 1]
    scenario['sum'].iloc[counter_reverse] = 1
    scenario[scenario['AOI'].iloc[row_index +1]].iloc[counter_reverse] = scenario[scenario['AOI'].iloc[row_index +1 ]].iloc

    counter_reverse = counter_reverse - 1


# 1st check: that it is not the last row of dataframe, when you get to one row to the last row, do a few tasks and then bre
# before that it says that if the cell (current row) is not fixation but the next one is, then it is a fixation

if len(scenario['Event'])-1 == row_index + 1 :
    if  scenario['Eye movement type'].iloc[row_index] != 'Fixation' and scenario['Eye movement type'].iloc[row_index+1] ==
        fixation_count = fixation_count + 1
        scenario.at[row_index+1, 'fixation'] = 1

        fixation_duration = fixation_duration + scenario['Gaze event duration'].iloc[row_index+1]

    # counter_fixation + 1 is the index of fixation not counter_fixation
    # it addes the index of fixation, AOI, and gaze duration to a series, then it appened this to the dataframe
    #++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
            s = pd.Series([row_index+1, scenario['timestamp'].iloc[row_index+1], scenario['scenario time'].iloc[row_index+1], scena
                           scenario['Gaze event duration'].iloc[row_index+1]], index=['Index', 'timestamp', 'scenario time', 'A
            df_fixation = df_fixation.append(s, ignore_index=True)

            break

        #2nd check:if this is the first row and eye movement is fixation, it is a fixation
        elif row_index == 0 and scenario['Eye movement type'].iloc[row_index] == 'Fixation':
            #print('++++++++++++++++++++++++++++++++++++', fixation_count)

            fixation_count = fixation_count + 1
            scenario.at[row_index, 'fixation'] = 1
            fixation_duration = fixation_duration + scenario['Gaze event duration'].iloc[row_index+1]

            s = pd.Series([row_index+1, scenario['timestamp'].iloc[row_index+1], scenario['scenario time'].iloc[row_index+1], scena
                           scenario['Gaze event duration'].iloc[row_index+1]], index=['Index', 'timestamp', 'scenario time', 'A
            df_fixation = df_fixation.append(s, ignore_index=True)

        #3rd check:if this is not the first and last row, and this is not fixaion but then next one is, then it is a fixation
        elif scenario['Eye movement type'].iloc[row_index] != 'Fixation' and scenario['Eye movement type'].iloc[row_index+1] == 'Fi
            #print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~', fixation_count)
            fixation_count = fixation_count + 1
            scenario.at[row_index+1, 'fixation'] = 1
            fixation_duration = fixation_duration + scenario['Gaze event duration'].iloc[row_index+1]

            s = pd.Series([row_index+1, scenario['timestamp'].iloc[row_index+1], scenario['scenario time'].iloc[row_index+1], scena
                          index=['Index', 'timestamp', 'scenario time', 'AOI', 'Duration'])
            df_fixation = df_fixation.append(s, ignore_index=True)

        # it addes one to row counter
        row_index += 1

    #print('number of fixation', fixation_count)
    #print('fixation duration', fixation_duration)
    #print('ave fixatin duration', fixation_duration/fixation_count)

    df_fixation.name = 'df_landing_data'
    outfile_1 = str(scenario.name) + '.csv'
    outfile_2 = str(scenario.name) + '_summary.csv'
    scenario.to_csv(outfile_1, encoding='utf-8')
```

```
    df_fixation.to_csv(outfile_2, encoding='utf-8')
    #print('++++++++++++++++++++++++++++++++++++')

    columns = ['From','To']
    index = range(0, math.ceil(len(df_fixation)/2))

    df_from_to = pd.DataFrame(index = index, columns = columns)

    counter_index = 0
    time.sleep(5)

    #print(df_fixation.head(5))

    for i in range(0, math.ceil(len(df_fixation)/2)):
        time.sleep(0.01)
        #print(counter_index, i, df_fixation['AOI'].iloc[counter_index], df_fixation['AOI'].iloc[counter_index+1])
        df_from_to['From'].iloc[i] = df_fixation['AOI'].iloc[counter_index]
        df_from_to['To'].iloc[i] = df_fixation['AOI'].iloc[counter_index+1]

        counter_index += 1


df_from_to = df_from_to[df_from_to['From'] != df_from_to['To']]
df_from_to = df_from_to.reset_index(drop=True)
df_crosstab = pd.crosstab(df_from_to.From, df_from_to.To)
df_crosstab


for i in AOI_list:
    if df_from_to['From'].str.contains(i).any() == False and df_from_to['To'].str.contains(i).any() == False:
        s = pd.Series([i, i], index=['From', 'To'])
        df_from_to = df_from_to.append(s, ignore_index=True)

df_crosstab = pd.crosstab(df_from_to.From, df_from_to.To)

for i in AOI_list:
    print(i)
    df_crosstab[i][i] = 0

print('++++++++++++++++++++++++++++++++++++++++++++++++++')
print(df_crosstab)
```

```
print('++++++++++++++++++++++++++++++++++++++++++++++++++')

df_from_to['numeric_from'] = pd.Categorical(df_from_to.From)
df_from_to['numeric_from'] = df_from_to['numeric_from'].cat.codes
df_from_to['numeric_to'] = pd.Categorical(df_from_to.To)
df_from_to['numeric_to'] = df_from_to['numeric_to'].cat.codes

#transition_matrix = pd.DataFrame(df_crosstab)
#transition_matrix

movements_from= list(df_from_to['numeric_from'])
movements_to= list(df_from_to['numeric_to'])
movements_from.append(movements_to[-1])

transition_matrix_list = [[0]*len(df_crosstab) for _ in range(len(df_crosstab))]

for i in range(0, len(movements_from)):
    if i+1 < len(movements_from):
        transition_matrix_list[movements_from[i]][movements_from[i+1]] += 1

transition_matrix = pd.DataFrame(transition_matrix_list)

for i in range(13):
    transition_matrix[i][i]=0

transition_matrix

transition_matrix.to_csv('synchronized_data.csv', encoding='utf-8')

probability_matrix = transition_matrix.div(transition_matrix.values.sum())
log_probability_matrix = probability_matrix.copy()
log_probability_matrix.iloc[:,:] = 0

for i in range (0,13):
    log_probability_matrix[i] = probability_matrix[i].apply(lambda x: math.log2(x) if x > 0 else 0)

entropy_matrix = probability_matrix * log_probability_matrix
entropy = -entropy_matrix.values.sum()

print('ENTROPY: ', entropy)
#%%
```

```
probability_matrix

#%%
probability_matrix_transposed = probability_matrix.T
probability_matrix_transposed

#%%
probability_matrix_melt = probability_matrix_transposed.melt()
probability_matrix_melt

#%%
probability_matrix_melt ['to'] = 0
probability_matrix_melt ['zero'] = 1
#%%
probability_matrix_melt = probability_matrix_melt [['variable', 'to', 'value', 'zero']]
#%%
for i in range(0, len(probability_matrix_melt), 13):
    for j in range (0, 13):
        print(i, j)
        probability_matrix_melt['to'].iloc[i+j] = j

#%%
probability_matrix_melt['zero'] = probability_matrix_melt.apply(lambda x: 0 if x['variable'] == x['to'] else 1, axis=1)


#%%
probability_matrix_melt



#%%
probability_matrix_melt.to_csv('data.csv', encoding='utf-8')
probability_matrix_melt



#%%
#[df_eye_landing, df_eye_levelturn, df_eye_levelflight, df_eye_descent]
for scenario in [df_eye_levelturn]: # , df_eye_levelturn, df_eye_levelflight, df_eye_descent

    # this part should be added to dataframe of eye
```

```
    rng = pd.date_range(scenario['time'].dt.round('100ms').iloc[0],
                        periods=(scenario['time'].iloc[-1] - scenario['time'].iloc[0]).total_seconds()*10,
                        freq='100L')

    time.sleep(5)

    eye_to_append = pd.DataFrame(rng)
    eye_to_append.columns = ['time']
    # ********************************************************************************************
    # creating a new dataframe with the stamp of 0
    # ********************************************************************************************
    # create a list of df_eye's columns
    list_columns = [x for x in scenario.columns]
    dataframe_eye = scenario[list_columns].copy()
    dataframe_eye['stamped'] = 0
    #print(dataframe_eye.columns)
    dataframe_eye

    for col in dataframe_eye.columns[:]:
        if col != 'time':
            eye_to_append[col] = np.nan

    # RW: Mark the new (interpolated) data as "stamped"
    eye_to_append['stamped'] = 1
    eye_data = dataframe_eye.append(eye_to_append, ignore_index = True).sort_values('time').reset_index(drop=True) #.set_index('tim
    eye_data[['Group', 'Number', 'Training', 'Event', 'Eye movement type', 'Gaze event duration', 'sum', 'AI', 'ALT', 'ASI','CONT',

#    print(eye_data.head(5))
#    eye_data.interpolate(limit_direction='both', inplace = True)
#    print (eye_data.iloc[0:5, 0:10])
#    print (eye_data.iloc[0:5, 10:20])

    row_index = 0

    for index, row in eye_data.iterrows():

        counter_reverse = row_index

        while (counter_reverse > 0 and eye_data['Eye movement type'].iloc[row_index] == 'Fixation' and
               eye_data['Eye movement type'].iloc[counter_reverse - 1 ] == 'Fixation' and
               eye_data['stamped'].iloc[row_index] == 1 and
```

```
                pd.isnull(eye_data['fixation'].iloc[row_index]) == True):

            if eye_data['fixation'].iloc[counter_reverse - 1] == 2 and eye_data['stamped'].iloc[counter_reverse - 1] == 1:
                break

            elif eye_data['fixation'].iloc[counter_reverse - 1] == 1 and eye_data['stamped'].iloc[counter_reverse - 1] == 0:
                eye_data.at[row_index, 'fixation'] = 2

            counter_reverse = counter_reverse - 1

        if pd.isnull(row['timestamp']) == True:


            eye_data.at[index, 'timestamp'] = (eye_data['time'].iloc[index] - eye_data['time'].iloc[index-1]) + eye_data['timestamp
            eye_data.at[index, 'scenario time'] = (eye_data['time'].iloc[index] - eye_data['time'].iloc[index-1]) + eye_data['scena

        row_index += 1


        eye_data['fixation'] = eye_data['fixation'].replace('NaN', 0)

    if eye_data['stamped'].iloc[0] == 1:
        eye_data = eye_data.drop(eye_data.index[0]).reset_index(drop=True)

    # RW: This is the data you want
    eye_final = eye_data.query('stamped == 1').drop('stamped', 1).reset_index(drop=True)
    eye_final = eye_final.set_index('time')

                                    # THE END OF CODING FOR EYE MOVEMENTS
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# **********************************************************************************************
# **********************************************************************************************
                                   # START OF CODING FOR FLIGHT DATA

#IMPORTING FILE AND DEFINING THE COLUMN
df_flight = pd.read_csv(fliename_flight, delimiter='\t') # RW: added the `delimiter` parameter
df_flight['timestamp'] = pd.to_datetime(df_flight['timestamp'], format='%m/%d/%Y %H:%M:%S.%f').dt.strftime('%Y-%m-%d %H:%M:%S.%f')
df_flight['timestamp'] = pd.to_datetime(df_flight['timestamp'], format='%Y-%m-%d %H:%M:%S.%f')

rng = pd.date_range(df_flight['timestamp'].dt.round('100ms').iloc[0],
```

```
                    periods=(df_flight['timestamp'].iloc[-1] - df_flight['timestamp'].iloc[0]).total_seconds()*10,
                    freq='100L')

# RW: We will add new rows with the desired time stamps (`date`)
flight_to_append = pd.DataFrame(rng)
flight_to_append.columns = ['timestamp']

# RW: There were some duplicate column names here I had to delete
list_columns = [x for x in df_flight.columns]

dataframe_flight = df_flight[list_columns].copy()
# RW: Mark the existing data as "not stamped"
dataframe_flight['stamped'] = 0

# RW: Add all the necessary empty columns to the new rows
for col in dataframe_flight.columns[1:]:
    flight_to_append[col] = None

# RW: Mark the new (interpolated) data as "stamped"
flight_to_append['stamped'] = 1

# RW: Prepare a new dataframe for interpolation
flight_interpolated = dataframe_flight.append(
    flight_to_append, ignore_index = True
).sort_values('timestamp').set_index('timestamp')
flight_interpolated

# RW: Interpolate! (i.e. fill in data values at the desired moments in time)
# RW: The options used here are a matter of choice, but
#     seemed like a good idea when I examined the data
flight_interpolated.interpolate(limit_direction='both', inplace = True)

# RW: This is the data you want
df_final = flight_interpolated.query('stamped == 1').drop('stamped', 1)

########################################
## VISUAL CHECK OF WHAT WE'VE DONE!!! ##

plt.rcParams['figure.figsize'] = 15,6
########################################
plt.rcParams['font.size'] = 18
```

```
# RW: For each column, visualize the actual/original
#     values and the interpolated values we've created
for col in flight_interpolated.columns[1:-1]:
    flight_interpolated.query('stamped == 0')[col].plot(marker = 'o', lw = 0, markersize = 10)
    flight_interpolated[col].plot(marker = 'o', lw = 0, markersize = 3)
    plt.legend(['Actual values', 'Interpolated values'])
    plt.ylabel(col)
    plt.show()


synchronized_data = eye_final.merge(flight_interpolated, how='left', left_index=True, right_index=True)

synchronized_data = synchronized_data.reset_index()

synchronized_data = synchronized_data.set_index('scenario time')

synchronized_data.head(5)

# ********************************************************************************************************
# ********************************************************************************************************
# ********************************************************************************************************
# ********************************************************************************************************
# ********************************************************************************************************
# ********************************************************************************************************
# ********************************************************************************************************
# ********************************************************************************************************
#                                  END OF CODING FOR FLIGHT DATA
list_interval = ['00:00:00.000000', '00:00:10.000000', '00:00:20.000000', '00:00:30.000000', '00:00:40.000000', '00:00:50.000000',
                 '00:01:00.000000', '00:01:10.000000', '00:01:20.000000', '00:01:30.000000', '00:01:40.000000', '00:01:50.000000',
                 '00:02:00.000000', '00:02:10.000000', '00:02:20.000000', '00:02:30.000000', '00:02:40.000000', '00:02:50.000000',
                 '00:03:00.000000', '00:03:10.000000', '00:03:20.000000', '00:03:30.000000', '00:03:40.000000', '00:03:50.000000',
                 '00:04:00.000000']

AOI_list =    ['AI',           'ALT',           'HI',          'TC',          'ASI' ,         'VSI',             'TAC',
flight_list = [                'pitch, deg',    'alt, ind',      'heading mag',  'roll, deg',   'Vind, kias',   'VVI, fpm',


synchronized_data['re_entry'] = np.nan
synchronized_data = synchronized_data.reset_index()
```

```
#%%

transition_matrix_list

#%%
df_from_to
#%%

df_test = pd.crosstab(df_from_to.From, df_from_to.To)
df_test

#%%
for i in range(len(AOI_list)):
    print (((synchronized_data.loc[(synchronized_data['fixation'] == 2) & (synchronized_data['AOI'] == AOI_list[i])]).index.shape[0
    print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    print(AOI_list[i])
    print('BEFORE ENTERING')

    for m in range(0, (synchronized_data.loc[(synchronized_data['fixation'] == 2) & (synchronized_data['AOI'] == AOI_list[i])]).ind
        print('M ==> ', m)
        print((synchronized_data.loc[(synchronized_data['fixation'] == 2) & (synchronized_data['AOI'] == AOI_list[i])]).index.shape
        print((synchronized_data.loc[(synchronized_data['fixation'] == 2) & (synchronized_data['AOI'] == AOI_list[i])].shape[0] !=
        print((m < (synchronized_data.loc[(synchronized_data['fixation'] == 2) & (synchronized_data['AOI'] == AOI_list[i])]).index.

        if (m < (synchronized_data.loc[(synchronized_data['fixation'] == 2) & (synchronized_data['AOI'] == AOI_list[i])]).index.sha
            list_index = (synchronized_data.loc[(synchronized_data['fixation'] == 2) & (synchronized_data['AOI'] == AOI_list[i])]).

            print(list_index)
            print('========================')
            print('m   :', m , '======   :', list_index[m])

            synchronized_data.at[list_index[m], 're_entry'] = (synchronized_data['scenario time'].iloc[list_index[m+1]] - synchroni
            print('++++++++++++++++++INSIDE++++++++++++++++++++')
            print(list_index, '==', AOI_list[i])

            time.sleep(0.25)
            print('check ====================================================')
            print(synchronized_data[['AOI', 're_entry','fixation', 'timestamp']].loc[(synchronized_data['AOI'] == AOI_list[i]) & (s

synchronized_data = synchronized_data.set_index('scenario time')
```

```
##%%
#(synchronized_data.loc[(synchronized_data['fixation'] == 2) & (synchronized_data['AOI'] == 'AI')]).index.shape[0]
##%%
#(synchronized_data['AOI'].loc[(synchronized_data['fixation'] == 2)]).value_counts()
##%%
#synchronized_data[['AOI', 're_entry','fixation', 'timestamp']].loc[(synchronized_data['AOI'] == 'RT') & (synchronized_data['fixati

synchronized_data = synchronized_data.reset_index()
synchronized_data['rate turn' ] = 0.0

i = 0

while i < synchronized_data.index.shape[0]-1:
    synchronized_data.at[i, 'rate turn' ] = (synchronized_data['heading mag'].iloc[i+1] - synchronized_data['heading mag'].iloc[i])
    i = i + 1

#%%

#%%
synchronized_data = synchronized_data.set_index('scenario time')

plt.rcParams.update({'font.size': 11})
plt.rcParams["font.weight"] = "bold"
plt.rcParams["axes.labelweight"] = "bold"

# It plots the data in each 10 time interval.


x = 0
while (x < len(list_interval)-1 and (synchronized_data[list_interval[x]:list_interval[x+1]]).empty == False):

    fig, axes = plt.subplots(nrows=10, ncols=1, figsize=(15, 25), sharex=True)

    for i in range(0, 10):
        if i <=6:

            synchronized_data[list_interval[x]:list_interval[x+1]][AOI_list[i]].plot(ax=axes[i+1], drawstyle='steps', color=['dodge
            axes[i+1].set_ylim(0, 1)
            axes[i+1].set_ylabel(AOI_list[i])
            axes[i+1].set_yticks(range(0,2))
```

```
        synchronized_data[list_interval[x]:list_interval[x+1]][flight_list[i]].plot(ax=axes[i+1], legend=False, secondary_y=Tru

        plt.ylabel(flight_list[i])
        axes[i+1].xaxis.grid(True)

    elif i == 7:

        synchronized_data[list_interval[x]:list_interval[x+1]][AOI_list[i]].plot(ax=axes[i-7], drawstyle='steps', color=['dodge
        axes[i-7].set_ylim(0, 1)
        axes[i-7].set_ylabel(AOI_list[i])
        axes[i-7].set_yticks(range(0,2))

        axes[i-7].xaxis.grid(True)

        #axes[i-7].fill_between(synchronized_data[list_interval[x]:list_interval[x+1]]['time_fill_between'].dt.to_pydatetime(),

    elif i == 8:

        synchronized_data[list_interval[x]:list_interval[x+1]][AOI_list[i]].plot(ax=axes[i], drawstyle='steps', color=['dodgerb
        axes[i].set_ylim(0, 1)
        axes[i].set_ylabel(AOI_list[i])
        axes[i].set_yticks(range(0,2))

        synchronized_data[list_interval[x]:list_interval[x+1]][AOI_list[i+1]].plot(ax=axes[i], legend=False, secondary_y=True,
        plt.ylabel(AOI_list[i+1])
        axes[i].right_ax.set_ylim(0,1)
        axes[i].right_ax.set_yticks(range(0,2))
        axes[i].xaxis.grid(True)

    elif i == 9:

        synchronized_data[list_interval[x]:list_interval[x+1]][AOI_list[i+1]].plot(ax=axes[i], drawstyle='steps', color=['dodge
        axes[i].set_ylim(0, 1)
        axes[i].set_ylabel(AOI_list[i+1])
        axes[i].set_yticks(range(0,2))

        synchronized_data[list_interval[x]:list_interval[x+1]][AOI_list[i+2]].plot(ax=axes[i], legend=False, secondary_y=True,
        plt.ylabel(AOI_list[i+2])
        axes[i].right_ax.set_ylim(0,1)
        axes[i].right_ax.set_yticks(range(0,2))
```

```
            axes[i].set_xlabel('The Look Points of The Pilot and The Flight Paramters')
            axes[i].xaxis.grid(True)

    plt.savefig("chart_" + str(x) + ".png")

    x += 1

    plt.show()

#%%
fixation_count_plot = synchronized_data.loc[synchronized_data['fixation'] == 2]
fixation_count_plot = fixation_count_plot.sort_values('AOI')
fixation_count_plot

fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(10, 15), sharex=True)

synchronized_data.loc[synchronized_data['fixation'] == 2].groupby('AOI').size().sort_index(ascending=True).plot(kind='bar', ax=axes

axes[0].set_ylabel('Fixation Count')
axes[0].set_xlabel('Areas of Interest (AOI)')
axes[0].get_yaxis().set_label_coords(-0.1,0.5)


synchronized_data['AOI'].loc[synchronized_data['fixation'] == 2].value_counts(normalize = True).sort_index(ascending=True).plot(kin
                secondary_y=True)

axes[0].yaxis.grid(True)
plt.ylabel('Fixation Count (%)')

synchronized_data.loc[synchronized_data['fixation'] == 2].groupby('AOI')['Gaze event duration'].mean().plot(kind = 'bar', ax=axes[1
axes[1].set_ylabel('Dwell Time (ms)')
axes[1].set_xlabel('Areas of Interest (AOI)')
axes[1].get_yaxis().set_label_coords(-0.1,0.5)

fixation_duration_plot = (synchronized_data.loc[synchronized_data['fixation'] == 2].groupby('AOI')['Gaze event duration'].mean() /
                          secondary_y=True)
axes[1].yaxis.grid(True)
plt.ylabel('Dwell Time (%)')

synchronized_data.loc[(synchronized_data['fixation'] == 2)].groupby('AOI')['re_entry'].mean().plot(kind = 'bar', ax=axes[2])
axes[2].set_ylabel('Re-entry Time (S)')
```

```
axes[2].set_xlabel('Areas of Interest (AOI)')
axes[2].get_yaxis().set_label_coords(-0.1,0.5)
axes[2].yaxis.grid(True)


plt.savefig("bar_plot.png")

#%%

#%%
synchronized_data.head(10)

#%%

synchronized_data['AOI'].loc[synchronized_data['fixation'] == 2].value_counts(normalize = True).sort_index(ascending=True)

#%%
flight_list = ['pitch, deg', 'alt, ind', 'heading mag', 'roll, deg', 'Vind, kias', 'VVI, fpm', 'rpm n, engin']
#%%
synchronized_data[['AOI', 're_entry', 'fixation']].loc[synchronized_data['fixation'] == 2]

#%%
synchronized_data.to_csv('synchronized_data.csv', encoding='utf-8')
#%%



synchronized_data[['pitch, deg', 'alt, ind', 'heading mag', 'roll, deg', 'Vind, kias', 'VVI, fpm', 'rpm n, engin']].resample('10S')







#%%
fixation_count_plot

import pandas as pd
from sklearn import preprocessing

data = {'score': [234,24,14,27,-74,46,73,-18,59,160]}
```

```
df = pd.DataFrame(data)
df

min_max_scaler = preprocessing.MinMaxScaler()
np_scaled = min_max_scaler.fit_transform(df)
df_normalized = pd.DataFrame(np_scaled)
df_normalized
#%%
#%%
synchronized_data['scenario time'] = pd.to_datetime(synchronized_data['scenario time'], format='%Y-%m-%d %H:%M:%S.%f')

synchronized_data = synchronized_data.reset_index()

#%%
plt.figure(figsize=(15,10))
plt.fill_between(synchronized_data['scenario time'].dt.to_pydatetime(), synchronized_data['AI']
                 ,facecolor='green', alpha=0.2, interpolate=True)

#%%
synchronized_data.dtypes

#%%
synchronized_data['AI'].plot()
#%%

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

N = 300
dates = pd.date_range('2000-1-1', periods=N, freq='D')
x = np.linspace(0, 2*np.pi, N)
data = pd.DataFrame({'A': np.sin(x), 'B': np.cos(x),
                'Date': dates})
plt.plot_date(data['Date'], data['A'], '-')
plt.plot_date(data['Date'], data['B'], '-')

d = data['Date'].values
plt.fill_between(d, data['A'], data['B'],
                 where=data['A'] >= data['B'],
                 facecolor='green', alpha=0.2, interpolate=True)
```

```
plt.xticks(rotation=25)
plt.show()

#%%


#%%
# lolliplot

df_data = pd.read_excel('color.xlsx')
df_data.columns

ordered_df = df_data.sort_values(by='observation')
ordered_df

import seaborn as sns
sns.set(font_scale=1.5)
plt.figure(figsize=(15,15))

#my_color=np.where((ordered_df['group']=='experimental') , 'orange', 'skyblue')
#my_size=np.where(ordered_df ['group']=='B', 70, 30)

plt.hlines(y=ordered_df['observation'].loc[ordered_df['status'] =='before'],
           xmin=ordered_df['value'].loc[ordered_df['status'] =='before'],
           xmax=ordered_df['value'].loc[ordered_df['status'] =='after'],
           color='grey', alpha=0.4)
# ordered_df['before'] ==> value
# ordered_df['after'] ==> value

#Experimental
plt.scatter(ordered_df['value'].loc[(ordered_df['status'] =='before') & (ordered_df['group'] =='experimental')],
            ordered_df['observation'].loc[(ordered_df['status'] =='before') & (ordered_df['group'] =='experimental')],
            color=ordered_df['color'].loc[(ordered_df['status'] =='before') & (ordered_df['group'] =='experimental')],
            alpha=0.6, label='Exp. - Prior Training')

plt.scatter(ordered_df['value'].loc[(ordered_df['status'] =='after') & (ordered_df['group'] =='experimental')],
            ordered_df['observation'].loc[(ordered_df['status'] =='after') & (ordered_df['group'] =='experimental')],
            color=ordered_df['color'].loc[(ordered_df['status'] =='after') & (ordered_df['group'] =='experimental')],
            alpha=1, label='Exp. - Post Training')

plt.scatter(ordered_df['value'].loc[(ordered_df['status'] =='before') & (ordered_df['group'] =='control')],
```

```
            ordered_df['observation'].loc[(ordered_df['status'] =='before') & (ordered_df['group'] =='control')],
            color=ordered_df['color'].loc[(ordered_df['status'] =='before') & (ordered_df['group'] =='control')],
            alpha=1, label='Con. - Prior Training', marker='s')

plt.scatter(ordered_df['value'].loc[(ordered_df['status'] =='after') & (ordered_df['group'] =='control')],
            ordered_df['observation'].loc[(ordered_df['status'] =='after') & (ordered_df['group'] =='control')],
            color=ordered_df['color'].loc[(ordered_df['status'] =='after') & (ordered_df['group'] =='control')],
            alpha=1, label='Con. - Post Training', marker='s')

#Control
#plt.scatter(ordered_df['value'].loc[ordered_df['status'] =='before'], ordered_df['observation'].loc[ordered_df['status'] =='before
#plt.scatter(ordered_df['value'].loc[ordered_df['status'] =='after'], ordered_df['observation'].loc[ordered_df['status'] =='before'

plt.legend()

# Add title and axis names
plt.yticks(ordered_df['observation'], ordered_df['group'])
plt.title("Comparison of the value 1 and the value 2", loc='left')
plt.xlabel('Value of the variables')
plt.ylabel('Group')

#%% test
df_1 = pd.DataFrame(data = {'Date': pd.to_datetime(['2012-01-01', '2012-01-04', '2012-01-05', '2012-01-06', '2012-01-11', '2012-01-
        'Close': [11.13, 11.30, 11.59, 11.71, 11.80, 16]})

df_2 = pd.DataFrame(data = {'Date': pd.to_datetime(['2012-01-03', '2012-01-02', '2012-01-05', '2012-01-06', '2012-01-12', '2012-01-
        'Close': [21.05, 21.15, 22.17, 22.92, 22.84,14,18]})

print('++++++++++++++++++++++++++++++++++')
print(df_1)
print('++++++++++++++++++++++++++++++++++')
print(df_2)
print('++++++++++++++++++++++++++++++++++')
df_1 = df_1.set_index('Date')
df_2 = df_2.set_index('Date')
print(df_1)
print('++++++++++++++++++++++++++++++++++')
print(df_2)
print('++++++++++++++++++++++++++++++++++')

df_1 = df_1.merge(df_2, how='left', left_index=True, right_index=True)
```

```
print('=====================================')
df_1
#%%
```