Meeting #9

11/4/21

Omar Luna

Deliverables

- Function that makes dictionary containing epoch and pupil diameter info
  - Input
    - tobii_data
    - epoch_dict
  - Output
    - epochbm_dict
- Adjust json parsing function to include
  - Avg of left and right diameter

Methodology and Learnings

- Used
  - read_eeg,
  - decisionTree_epochDetection
  - Json_to_dataframe
- Converted tobii timestamps from numpy.datetime to datetime.datetime
- Added rows within each epoch to a list
- Created a tuple containing the epoch boundaries and the list of values from the dataframe

Results

```
from read_eeg import *
from decisionTree_epochDetection import *
from getEpochbm_dict import *
from js_pupil_diameter import *

import numpy as np

eeg_data = read_eeg("2020_06_04_T05_U00T_EEG01.vhdr")

X = (np.vstack((np.arange(len(eeg_data)), eeg_data["HR"]))).transpose()
epoch_dict = decisionTree_epochDetection(5 , X)

tobii_data = json_to_dataframe("participant.json", "livedata.json")
```

	pd		pd_avg
eye	left	right	
2020-06-04 23:13:42.000	0.00	4.50	2.250
2020-06-04 23:13:42.010	4.05	4.48	6.290
2020-06-04 23:13:42.020	0.00	4.51	2.255
2020-06-04 23:13:42.030	4.06	4.48	6.300
2020-06-04 23:13:42.040	0.00	4.52	2.260

```
# Converts initial datetime value to int
timestamp = int(round(p_ts.timestamp()))
# stores index values
index_vals = df_pd.index.values
# calculates the difference in seconds between values (1e6 = microseconds)
step = 1/100

# Converts int values to datetime values
for i in range(0, index_vals.size, 1):
  l_final_ts.append(dt.fromtimestamp(timestamp + (i*step)))

# Final dataframe
df_final = pd.DataFrame(df_pd.values, columns= df_pd.columns,  index=l_final_ts)

# Calculate avg value for left and right pupil diameter
l_avg = [0] * len(df_final.index)
for i, p_dia in enumerate(df_final.values):
  l_avg[i] = p_dia[0] + p_dia[1] / 2
# add avg column
df_final['pd_avg'] = l_avg

return df_final
```

```
def pd_epoch_dict(tobii_data, epoch_dict):
    # original datetime: 2020-06-04 23:45:19.217
    # altered for testing : 2020-06-04 23:10:19.217
    dt_timestamp = dt.strptime("2020-06-04 23:10:19.217", "%Y-%m-%d %H:%M:%S.%f" )
    y = int(round(dt_timestamp.timestamp()))
    # list containing converted epoch timestamps
    l_epoch_ts = []
    # list containing data from tobii dataframe
    l_list = []
    # list containing epoch and data tuples
    l_tps = []

    # Loop to traverse epoch values
    for k in epoch_dict.values():
        x = k
        # converts epoch boundaries to datetime and stores in l_epoch_ts
        # timestamp / 500 = conversion from 500hz to seconds
        x[0] = dt.fromtimestamp((x[0]/500)+y)
        x[1] = dt.fromtimestamp((x[1]/500)+y)
        l_epoch_ts.append(x)

        # Loop to traverse values in tobii_data
        for i, df_item in enumerate(tobii_data.index.values):

            # Converts from numpy.datetime to datetime.datetime
            data_df_ts = dt.utcfromtimestamp(df_item.astype('O')/1e9)

            # Compares the index timestamp to the epoch boundaries
            if data_df_ts >= x[0] and data_df_ts <= x[1] and i < 10:

                data_row = tobii_data.iloc[i,:]
                # stores rows of data into l_list
                l_list.append(data_row)

        # Create the epoch boundaries and dataframe values tuple
        tp = tuple(zip(l_epoch_ts + l_list))
        # add tuple to tuple list
        l_tps.append(tp)
        # clear lists used
        l_list.clear()
        l_epoch_ts.clear()
    epochbm_dict = dict(zip(epoch_dict.keys(), l_tps))

    # Returns dictionary
    return epochbm_dict
dict_f = pd_epoch_dict(tobii_data, epoch_dict)
print(dict_f)
```

```
(csproject) [omar@olA15 newFunctions]$ python pd_*
{'1': (([datetime.datetime(2020, 6, 4, 23, 10, 19), datetime.datetime(2020, 6, 4, 23, 11, 3, 520000)],),), '2': (([datetime.datetime(2020, 6, 4, 23, 11, 3, 520000), datetime.datetime(2020, 6, 4, 23, 13, 15, 182000)]
,),), '3': (([datetime.datetime(2020, 6, 4, 23, 13, 15, 182000), datetime.datetime(2020, 6, 4, 23, 14, 6, 846000)],), (         eye
pd      left     0.00
        right    4.50
pd_avg           2.25
Name: 2020-06-04 23:13:42, dtype: float64,), (         eye
pd      left     4.05
        right    4.48
pd_avg           6.29
Name: 2020-06-04 23:13:42.010000, dtype: float64,), (         eye
pd      left     0.000
        right    4.510
pd_avg           2.255
Name: 2020-06-04 23:13:42.020000, dtype: float64,), (         eye
pd      left     4.06
        right    4.48
pd_avg           6.30
Name: 2020-06-04 23:13:42.030000, dtype: float64,), (         eye
pd      left     0.00
        right    4.52
pd_avg           2.26
Name: 2020-06-04 23:13:42.040000, dtype: float64,), (         eye
pd      left     4.080
        right    4.490
pd_avg           6.325
Name: 2020-06-04 23:13:42.050000, dtype: float64,), (         eye
pd      left     0.000
        right    4.530
pd_avg           2.265
Name: 2020-06-04 23:13:42.060000, dtype: float64,), (         eye
pd      left     4.09
        right    4.50
pd_avg           6.34
Name: 2020-06-04 23:13:42.070000, dtype: float64,), (         eye
pd      left     0.000
        right    4.530
pd_avg           2.265
Name: 2020-06-04 23:13:42.080000, dtype: float64,), (         eye
pd      left     4.10
        right    4.50
pd_avg           6.35
Name: 2020-06-04 23:13:42.090000, dtype: float64,)), '4': (([datetime.datetime(2020, 6, 4, 23, 14, 6, 846000), datetime.datetime(2020, 6, 4, 23, 14, 40, 844000)],),), '5': (([datetime.datetime(2020, 6, 4, 23, 14, 40
, 844000), datetime.datetime(2020, 6, 4, 23, 15, 12, 768000)],),)}
```

```
print(epochbm_dict.get("3")[1])
print(epochbm_dict.get("3")[2])
```

```
(([datetime.datetime(2020, 6, 4, 23, 10, 19), datetime.datetime(2020, 6, 4, 23, 11, 3, 520000)],),)
(         eye
pd       left     0.00
        right     4.50
pd_avg            2.25
Name: 2020-06-04 23:13:42, dtype: float64,)
```

```
(([datetime.datetime(2020, 6, 4, 23, 10, 19), datetime.datetime(2020, 6, 4, 23, 11, 3, 520000)],),)
(         eye
pd       left     4.05
        right     4.48
pd_avg            6.29
Name: 2020-06-04 23:13:42.010000, dtype: float64,)
```

```
def pd_epoch_dict(tobii_data, epoch_dict):
    # original datetime: 2020-06-04 23:45:19.217
    # altered for testing : 2020-06-04 23:10:19.217
    dt_timestamp = dt.strptime("2020-06-04 23:45:19.217", "%Y-%m-%d %H:%M:%S.%f" )
    y = int(round(dt_timestamp.timestamp()))
    # list containing converted epoch timestamps
    l_epoch_ts = []
    # list cointaining data from tobii dataframe
    l_list = []
    # list containing epoch and data tuples
    l_tps = []

    # Loop to traverse epoch values
    for k in epoch_dict.values():
        x = k
        # converts epoch boundaries to datetime and stores in l_epoch_ts
        # timestamp / 500 = conversion from 500hz to seconds
        x[0] = dt.fromtimestamp((x[0]/500)+y)
        x[1] = dt.fromtimestamp((x[1]/500)+y)
        l_epoch_ts.append(x)

        # Loop to traverse values in tobii_data
        for i, df_item in enumerate(tobii_data.index.values):

            # Converts from numpy.datetime to datetime.datetime
            data_df_ts = dt.utcfromtimestamp(df_item.astype('O')/1e9)

            # Compares the index timestamp to the epoch boundaries
            if data_df_ts >= x[0] and data_df_ts <= x[1]:

                data_row = tobii_data.iloc[i,:]
                # stores rows of data into l_list
                l_list.append(data_row)

        # Create the epoch boundaries and dataframe values tuple
        tp = tuple(zip(l_epoch_ts + l_list))
        # add tuple to tuple list
        l_tps.append(tp)
        # clear lists used
        l_list.clear()
        l_epoch_ts.clear()
    epochbm_dict = dict(zip(epoch_dict.keys(), l_tps))

    # Returns dictionary
    return epochbm_dict
dict_f = pd_epoch_dict(tobii_data, epoch_dict)
```