[ ]:
```python
# imports
import pyxdf
import pandas as pd
import datetime
import os
import matplotlib.pyplot as plt
import numpy as np
# from IPython.display import display
import seaborn as sns
sns.set_style("white")
```

Load data

Test files with different durations are stored in the folder "data". Before loading them, we want to access their path and save them in a dictionary.

Task 1:

- Get all files from "data" folder, sort them alphabetically and save them to a variable "files" Hint: os.listdir()

[ ]:
```python
# get all files from the folder "data"
files = os.listdir("data")
files = [f for f in files if '.xdf' in str.lower(f)]
# sort them alphabetically
files.sort()
files
```

[ ]:
```python
# save all files to an empty dictionary "recordings"
recordings = {}

for i, file in enumerate(files):  # store and display all files
    created = os.path.getmtime(f"data/{file}")  # creation timestamp
    created = datetime.datetime.fromtimestamp(created)  # translate as datetime
    created = created.strftime("%d.%m.%Y %H:%M")  # arrange it
    recordings[i] = {"file": file, "created": created}

files = [f.split(".")[0] for f in files]
# display the recordings and metadata
display(recordings)
```

Extensible Data Format (XDF)

.XDF is a format to store multiple channels of time series data with specific meta information. For the latency test, we stored 3 streams - Unity: 'Visual', - EEG: 'openvibeMarkers', 'openvibeSignal'

The loaded .XDF file results in a list of dictionaries with two main components:

1. **'time_series':** the information sent from Unity and EEG to via LSL
2. **'time_stamps':** the time stamps for each datapoint received

Task 2:

Load data streams for test 1 Hint: look at .XDF website info

[ ]:
```python
# load data
streams, _ = pyxdf.load_xdf(f"data/{recordings[0]['file']}")
display(streams)
```

Accessing streams

Once we load the streams, we can access each list in the dictionary by indicating for example the component we want to access such as stream (an index), 'info', time_stamps or time_series, etc. For specific information: - streams[2]['info']['name']

For a particular stream component: - streams[2]['time_stamps']

For a specific eeg channel - streams[2]["time_series"][:,64]

[ ]:
```
# access the second stream's time_stamps
streams[2]['time_stamps']
```

[ ]:
```
# print all stream names and index in stream list
s_names = {streams[i]["info"]["name"][0]: i for i in range(len(streams))}
s_names
```

Task 3: Defining functions

Function 1 Define a function that given the .xdf streams, returns the relevant streams: - The Unity samples were sent via a stream called *Visual* - The EEG light sensor data was sent via a stream called *openvibeSignal*

[ ]:
```
# define a function to select "Visual streams  and openvibeSignal from streams
def select_streams(streams):
    # stream names
    u_ch_name = "Visual"
    e_ch_name = "openvibeSignal"

    # get all current streams with their positions on the recording
    # example: {'Diode': 0, 'Audio': 1, 'openvibeSignal': 2}
    s_names = {streams[i]["info"]["name"][0]: i for i in range(len(streams))}

    # store and return their positions
    u = s_names[u_ch_name]  # unity stream channel
    e = s_names[e_ch_name]  # eeg stream channel (photodiode)
    return u, e
```

[ ]:
```
# select the Unity and EEG streams
u_ch, e_ch = select_streams(streams)
```

Function 2

Define a function that returns the computer "hostname" from the "info" dictionary given the streams data and the selected streams

[ ]:
```
# retrieve computer hostname
def stream_host(streams, u_ch, e_ch):
    # unity hostname
    u_host = streams[u_ch]["info"]["hostname"]
    # eeg hostname
    e_host = streams[e_ch]["info"]["hostname"]
    return u_host, e_host
```

Function 3

Define a function that returns the "time_stamps" for each selected stream in the streams data

[ ]:
```
def streams_ts(streams, u_ch, e_ch):
     # select timestamps for each stream
     u_ts = streams[u_ch]["time_stamps"]
     e_ts = streams[e_ch]["time_stamps"]
```

```
    return u_ts, e_ts
```

Function 4

Define a function that corrects the Unity and EEG timestamps to start at zero Hint: return a numpy array containing all corrected timestamps

```
[ ]: def corrected_ts(time_stamps):
         ts_norm = np.array([time_stamps[i] - time_stamps[0] for i in␣
      ↪range(len(time_stamps))])
         return ts_norm
```

Function 5

Define a function that calculates the latency of one sample to the next using the normalized timestamps. Hint: latency defined as the time difference between a timestamp and the next one

```
[ ]: def calculate_latency(ts_norm):
         ts_latency = np.array([ts_norm[i + 1] - ts_norm[i] for i in range(len(ts_norm)␣
      ↪- 1)])
         return ts_latency
```

Function 6

Define a function that retrieves the EEG and Unity streams data stored in *"time_series"* Hint: "time_series" contains the data sample that corresponds to each timestamp.

For Unity, we saved 0, 1, and 2 every time it switched from black to gray to white (channel 1) EEG contains the changes in light intensity for each of the colors collected in channel 64

```
[ ]: def streams_data(streams, u_ch, e_ch):
         # Data from streams
         u_data = streams[u_ch]["time_series"][:,1]
         # eeg only channel 64 where the diode was connected
         e_data = streams[e_ch]["time_series"][:,64]
         return u_data, e_data
```

Function 7

Define a function that calculates the test duration in minutes.

```
[ ]: def test_duration(time_stamps):
         # calculate recording's duration
         t_duration =  (time_stamps[-1] - time_stamps[0]) / 60
         return t_duration
```

Function 8

Define a function that calculates the average sampling rate

```
[ ]: def sampling_fps(t_duration, time_stamps):
         # calculate sampling rate (FPS)
         sampled_fps = 1 / (t_duration * 60 / len(time_stamps))
         return sampled_fps
```

Access all tests data

Now we can use the above functions to iterate over all the recordings and retrieve the corresponding data.

```
[ ]: # extract streams info for all files
     overview_df = pd.DataFrame()
     recordings_data_eeg = pd.DataFrame()
```

```
recordings_data_unity = pd.DataFrame()

for r in recordings:
    # current filename
    file = recordings[r]["file"]
    # load data
    streams, _ = pyxdf.load_xdf(f"data/{file}")
    # select stream channels
    u_ch, e_ch = select_streams(streams)
    # computer host name
    u_host, e_host = stream_host(streams, u_ch, e_ch)
    # select timestamps for each stream
    u_ts, e_ts = streams_ts(streams, u_ch, e_ch)
    # corrected timestamps to start at zero
    u_ts_corrected = corrected_ts(u_ts)
    e_ts_corrected = corrected_ts(e_ts)
    # latency of normalized timestamps
    u_latency = calculate_latency(u_ts_corrected)
    e_latency = calculate_latency(e_ts_corrected)
    # samples data from streams
    u_data, e_data = streams_data(streams, u_ch, e_ch)
    # calculate Unity - eeg time difference at start of recording
    # in milliseconds
    diff_ts = (u_ts[0] - e_ts[0]) * 1000
    # calculate recoding duration
    unity_duration =  test_duration(u_ts)
    eeg_duration =  test_duration(e_ts)
    # calculate sampling rate (FPS)
    unity_sr = sampling_fps(unity_duration, u_ts)
    eeg_sr = sampling_fps(eeg_duration, e_ts)
    # Store overview statistics for each test
    overview_df = overview_df.append({"file_name": file,
                          "u_duration": unity_duration,
                          "diff_ts": diff_ts,
                          "eeg_duration": eeg_duration,
                          "u_fps": unity_sr,
                          "eeg_fps": eeg_sr,
                          "u_host": u_host,
                          "eeg_host": e_host}, ignore_index=True)
    # Store key stream for use in analysis
    r_data_eeg = pd.concat([pd.DataFrame(np.resize(file, len(e_ts_corrected)),␣
 ↪columns=['filename']),
                            pd.DataFrame(e_ts, columns=['e_ts']),
                            pd.DataFrame(e_ts_corrected, columns=['e_ts_norm']),
                            pd.DataFrame(e_latency, columns=['e_latency']),
                            pd.DataFrame(e_data, columns=['sensor_data'])],␣
 ↪axis=1)
    r_data_unity = pd.concat([pd.DataFrame(np.resize(file, len(u_ts_corrected)),␣
 ↪columns=['filename']),
                            pd.DataFrame(u_ts, columns=['u_ts']),
                            pd.DataFrame(u_ts_corrected, columns=['u_ts_norm']),
                            pd.DataFrame(u_latency, columns=['u_latency']),
                            pd.DataFrame(u_data, columns=['switch_state'])],␣
 ↪axis=1)
    recordings_data_eeg = recordings_data_eeg.append(r_data_eeg, ignore_index=True)
    recordings_data_unity = recordings_data_unity.append(r_data_unity,␣
 ↪ignore_index=True)
```

[ ]:
```
overview_df
```

[ ]:
```
recordings_data_unity
```

[ ]:
```
recordings_data_eeg
```

Tests' descriptive statistic

Now we can calculate the average latency with which each EEG and Unity sample entered the system for each of the recorded tests.

Task 4:

- Calculate the average latency for each device (EEG, Unity)
- How constant are the frame rates?

[ ]:
```
# Unity's latency
recordings_data_unity.groupby('filename')['u_latency'].describe()
```

[ ]:
```
# EEG's latency
recordings_data_eeg.groupby('filename')['e_latency'].describe()
```

How constant are the framerates? - EEG (1024Hz) each sample ~ 0.98ms. - Unity (90Hz) each sample ~ 11.11 ms

Latencies visualization

Task 5:

- Visualize the latencies distributions for EEG and Unity

[ ]:
```
fig, ax = plt.subplots(2,1, figsize=(8, 5), sharex=True)
# set figure title
fig.suptitle("Latency distributions for EEG and Unity", fontsize=16,␣
 ↪fontweight='bold')
# plot eeg latencies
sns.boxplot(x="filename", y="e_latency",data=recordings_data_eeg,
            palette="muted", ax=ax[0])
ax[0].set_xlabel("")
ax[0].set_ylabel("EEG times [s]", fontsize=12)
ax[0].set_ylim(recordings_data_eeg["e_latency"].min(),
               recordings_data_eeg["e_latency"].max())
ax[0].set_ylim(0.00085, 0.0011)
# plot unity latencies
sns.boxplot(x="filename", y="u_latency",data=recordings_data_unity,
            palette="muted", ax=ax[1])
ax[1].set_xlabel("Test file name",fontsize=12)
ax[1].set_ylabel("Unity times [s]", fontsize=12)
sns.despine()
plt.tight_layout()
plt.savefig("logos/latency-distributions.pdf",
            format='pdf',bbox_inches='tight', dpi=1200)
plt.close()
```

Markers' visualizations

Now we will visualize the behavior between EEG and Unity for the first 5 seconds of recording in one of the tests. ### Task 6: - Select test number 3 from the EEG and Unity dataframe and visualize the first 5 seconds of recording

[ ]:
```
# Unity's first 5 seconds
u_start_t3 = recordings_data_unity[(recordings_data_unity['filename'] == 'lsl_test3.
↪xdf') &
                                  (recordings_data_unity['u_ts_norm'] <= 5)]
# EEG's first 5 seconds
e_start_t3 = recordings_data_eeg[(recordings_data_eeg['filename'] == 'lsl_test3.
↪xdf') &
                                  (recordings_data_eeg['e_ts_norm'] <= 5)]
u_start_t3
```

[ ]:
```
fig, ax = plt.subplots(2,1, figsize=(8,5), sharex=True)
fig.suptitle("EEG and Unity switch states", fontsize=16, fontweight='bold')
# plot Unity data
g1 = sns.scatterplot(x="u_ts_norm", y="switch_state",data=u_start_t3,
                     hue="switch_state",ax=ax[0])
ax[0].legend(bbox_to_anchor=(1.02, 1), loc='upper left',  fontsize=12)
ax[0].set_xlabel("")
ax[0].set_ylabel("Unity markers", fontsize=12)
# # plot EEG data
g2 = sns.lineplot(x="e_ts_norm", y="sensor_data", data=e_start_t3,
                  label='photodiode', linewidth=1, ax=ax[1])
ax[1].set_xlabel("Time [s]", fontsize=12)
ax[1].set_ylabel("EEG (sensor data)", fontsize=12)
ax[1].legend(bbox_to_anchor=(1.24, 1), loc='upper right',  fontsize=12)
ax[1].set_ylim(e_start_t3["sensor_data"].min(), e_start_t3["sensor_data"].max())
ax[1].set_xlim(0,None)
plt.yscale("symlog")
sns.despine()
plt.tight_layout()
plt.savefig("logos/eeg-unity-switch-states.pdf",
            format='pdf',bbox_inches='tight', dpi=1200)
plt.close()
```

Conclusions:

1. EEG sampling rate was extremely constant with each sample latency (mean = ~ 0.98 ms, std = 0.0)
2. Unity sampling rate was constant with each sample latency (mean = ~ 11.11 ms, std = 0.75 ms)
3. Latencies for both devices were constant independently of recording duration
4. LSL is a reliable method to collect brain and behavioral data when combining EEG and VR.

[ ]:
```

```