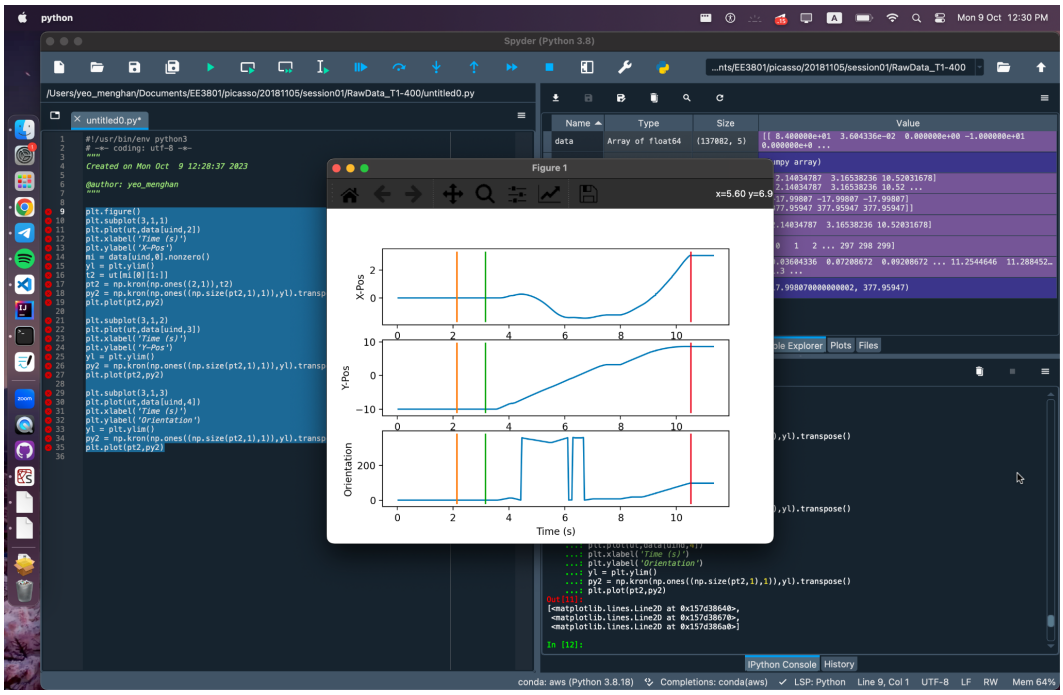
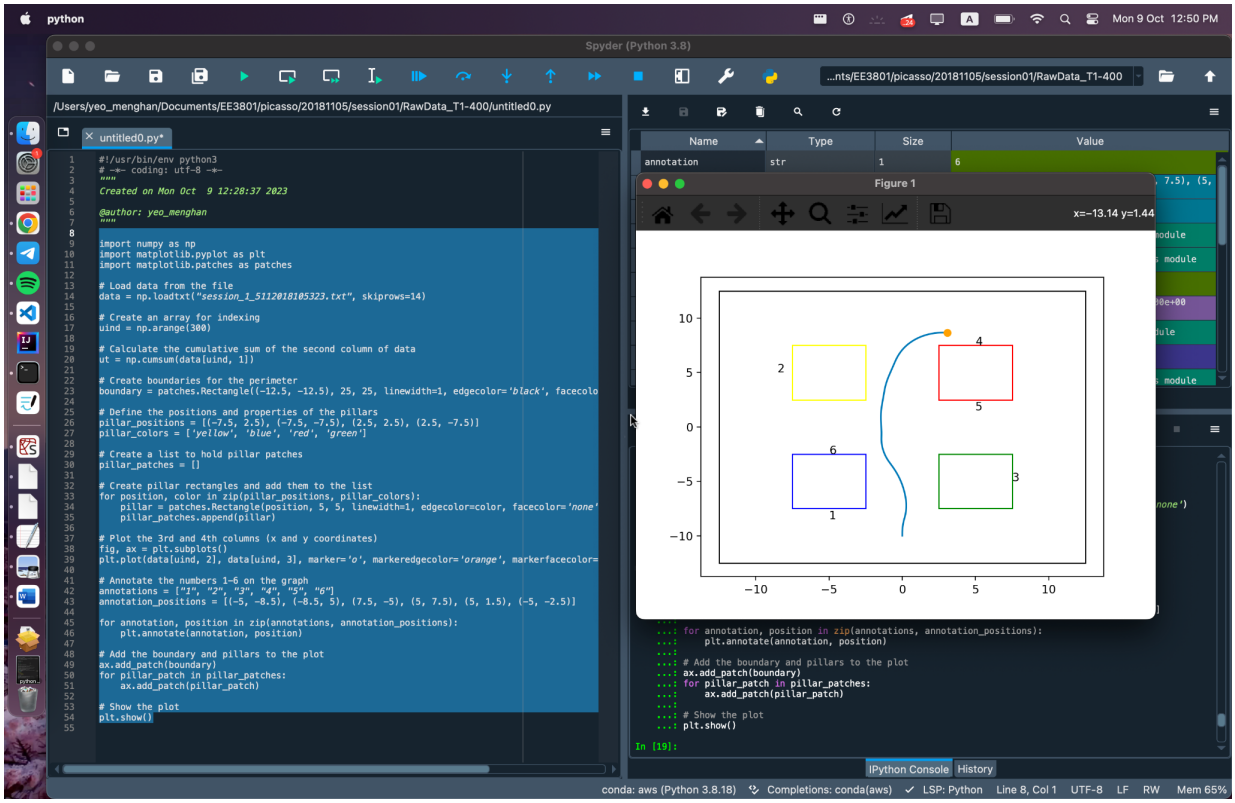
Yeo Meng Han (A0251772A)
Lab 4B - https://ee3801.github.io/Lab4/part-b/index.html

Task 26: *Include the figure produced by the code above in your lab report.*



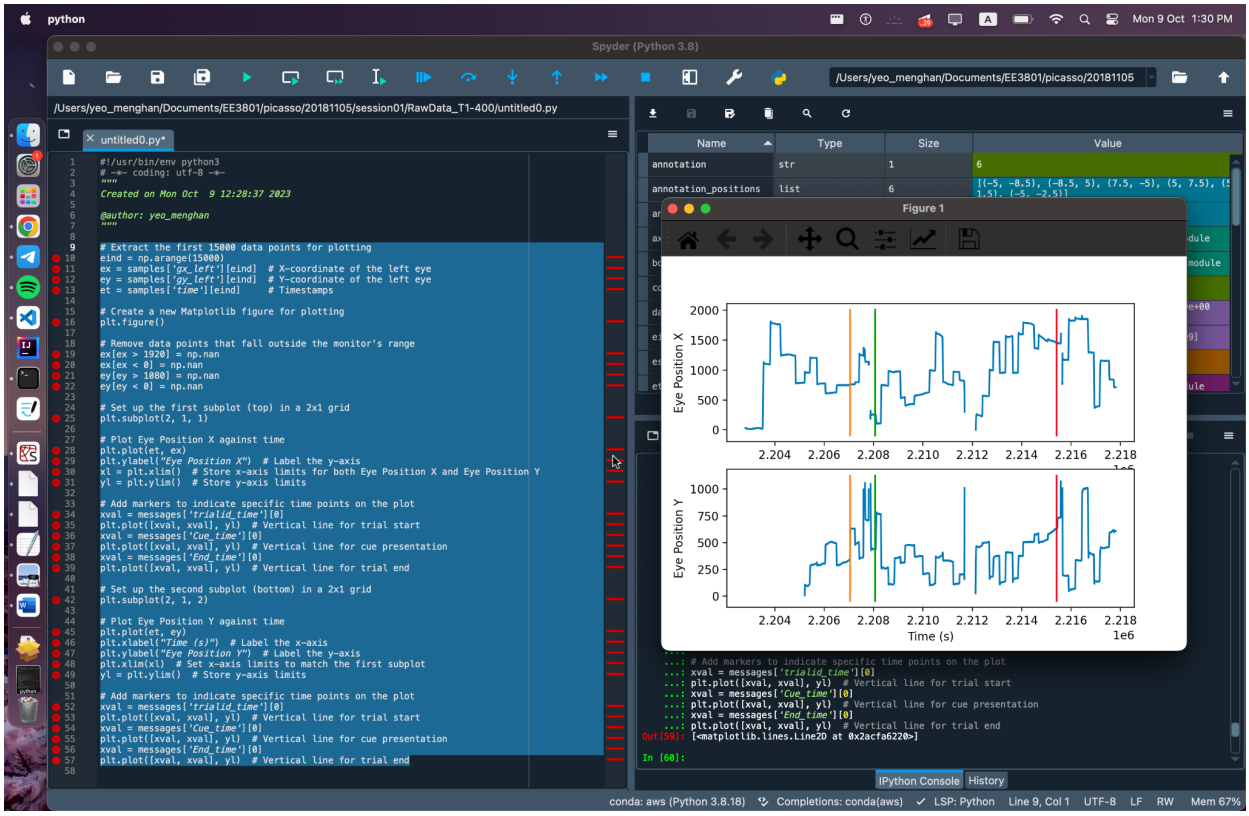
***Task 27**: A better way to view the position data is to create a 2D plot (like the one shown below), which you should do now. But before plotting the x,y positions of the animal, we should also show the boundaries of the maze (at -12.5 to 12.5 in both x- and y-directions), as well as the pillars (at ±2.5 and ±7.5) inside the maze. Add a different marker to the last point so it is clear which is the destination. Also add text labels to mark the locations of the posters (at ±2.5, ±5, and ±7.5).*




```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Load data from the file
data = np.loadtxt("session_1_5112018105323.txt", skiprows=14)

# Create an array for indexing
uind = np.arange(300)

# Calculate the cumulative sum of the second column of data
ut = np.cumsum(data[uind, 1])

# Create boundaries for the perimeter
```

```
boundary = patches.Rectangle((-12.5, -12.5), 25, 25, linewidth=1,
edgecolor='black', facecolor='none')

# Define the positions and properties of the pillars
pillar_positions = [(-7.5, 2.5), (-7.5, -7.5), (2.5, 2.5), (2.5, -7.5)]
pillar_colors = ['yellow', 'blue', 'red', 'green']

# Create a list to hold pillar patches
pillar_patches = []

# Create pillar rectangles and add them to the list
for position, color in zip(pillar_positions, pillar_colors):
pillar = patches.Rectangle(position, 5, 5, linewidth=1, edgecolor=color,
facecolor='none')
pillar_patches.append(pillar)

# Plot the 3rd and 4th columns (x and y coordinates)
fig, ax = plt.subplots()
plt.plot(data[uind, 2], data[uind, 3], marker='o', markeredgecolor='orange',
markerfacecolor='orange', markevery=[-1])

# Annotate the numbers 1-6 on the graph
annotations = ["1", "2", "3", "4", "5", "6"]
annotation_positions = [(-5, -8.5), (-8.5, 5), (7.5, -5), (5, 7.5), (5, 1.5),
(-5, -2.5)]

for annotation, position in zip(annotations, annotation_positions):
plt.annotate(annotation, position)

# Add the boundary and pillars to the plot
ax.add_patch(boundary)
for pillar_patch in pillar_patches:
ax.add_patch(pillar_patch)

# Show the plot
plt.show()
```

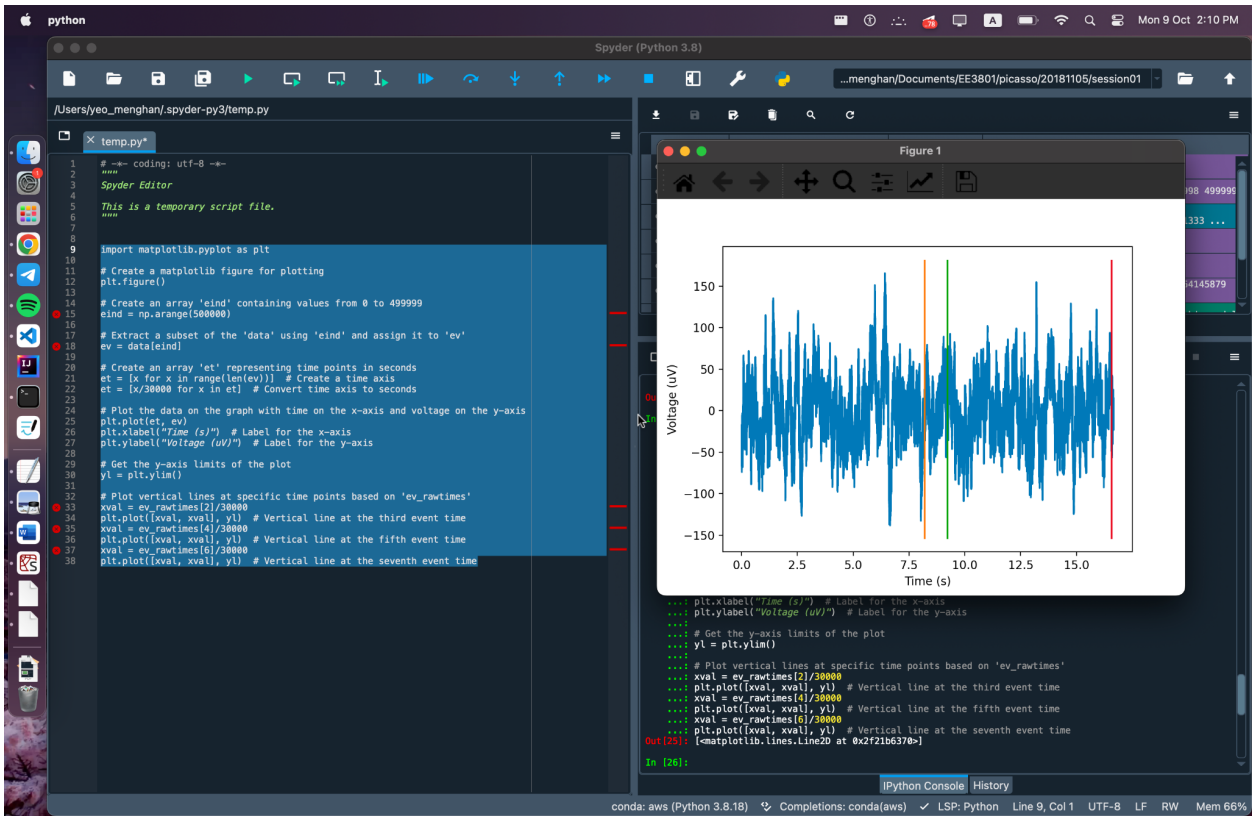
Task 41: *Using the steps described above, write a Python script to create a figure with 2 sub-plots with appropriate labels (shown below) that shows the x-positionj and y-position for the first 15000 time points. You can use the DataFrame.to_numpy() function to convert DataFrame values to a numpy array, and the xlim function in matplotlib to align the x-axes. Remember to remove both x- and y-values outside the range of the monitor (see Step 35). Please use vectorization techniques to draw vertical lines indicating the 3 marker times (trialid_time, Cue_time, and End_time) for the first trial on each subplot.*




```
# Extract the first 15000 data points for plotting
eind = np.arange(15000)
ex = samples['gx_left'][eind] # X-coordinate of the left eye
ey = samples['gy_left'][eind] # Y-coordinate of the left eye
et = samples['time'][eind] # Timestamps

# Create a new Matplotlib figure for plotting
plt.figure()

# Remove data points that fall outside the monitor's range
ex[ex > 1920] = np.nan
ex[ex < 0] = np.nan
ey[ey > 1080] = np.nan
```

```
ey[ey < 0] = np.nan

# Set up the first subplot (top) in a 2x1 grid
plt.subplot(2, 1, 1)

# Plot Eye Position X against time
plt.plot(et, ex)
plt.ylabel("Eye Position X") # Label the y-axis
xl = plt.xlim() # Store x-axis limits for both Eye Position X and Eye Position
Y
yl = plt.ylim() # Store y-axis limits

# Add markers to indicate specific time points on the plot
xval = messages['trialid_time'][0]
plt.plot([xval, xval], yl) # Vertical line for trial start
xval = messages['Cue_time'][0]
plt.plot([xval, xval], yl) # Vertical line for cue presentation
xval = messages['End_time'][0]
plt.plot([xval, xval], yl) # Vertical line for trial end

# Set up the second subplot (bottom) in a 2x1 grid
plt.subplot(2, 1, 2)

# Plot Eye Position Y against time
plt.plot(et, ey)
plt.xlabel("Time (s)") # Label the x-axis
plt.ylabel("Eye Position Y") # Label the y-axis
plt.xlim(xl) # Set x-axis limits to match the first subplot
yl = plt.ylim() # Store y-axis limits

# Add markers to indicate specific time points on the plot
xval = messages['trialid_time'][0]
plt.plot([xval, xval], yl) # Vertical line for trial start
xval = messages['Cue_time'][0]
plt.plot([xval, xval], yl) # Vertical line for cue presentation
xval = messages['End_time'][0]
plt.plot([xval, xval], yl) # Vertical line for trial end
```

Task 48: *Write a Python script to create the figure below with appropriate labels that shows the broadband data together with the markers for the first 500,000 samples. Please use vectorization techniques to draw vertical lines indicating the 3 marker times.*




```
# Create a matplotlib figure for plotting
plt.figure()

# Create an array 'eind' containing values from 0 to 499999
eind = np.arange(500000)

# Extract a subset of the 'data' using 'eind' and assign it to 'ev'
ev = data[eind]

# Create an array 'et' representing time points in seconds
et = [x for x in range(len(ev))] # Create a time axis
et = [x/30000 for x in et] # Convert time axis to seconds

# Plot the data on the graph with time on the x-axis and voltage on the
y-axis
plt.plot(et, ev)
```

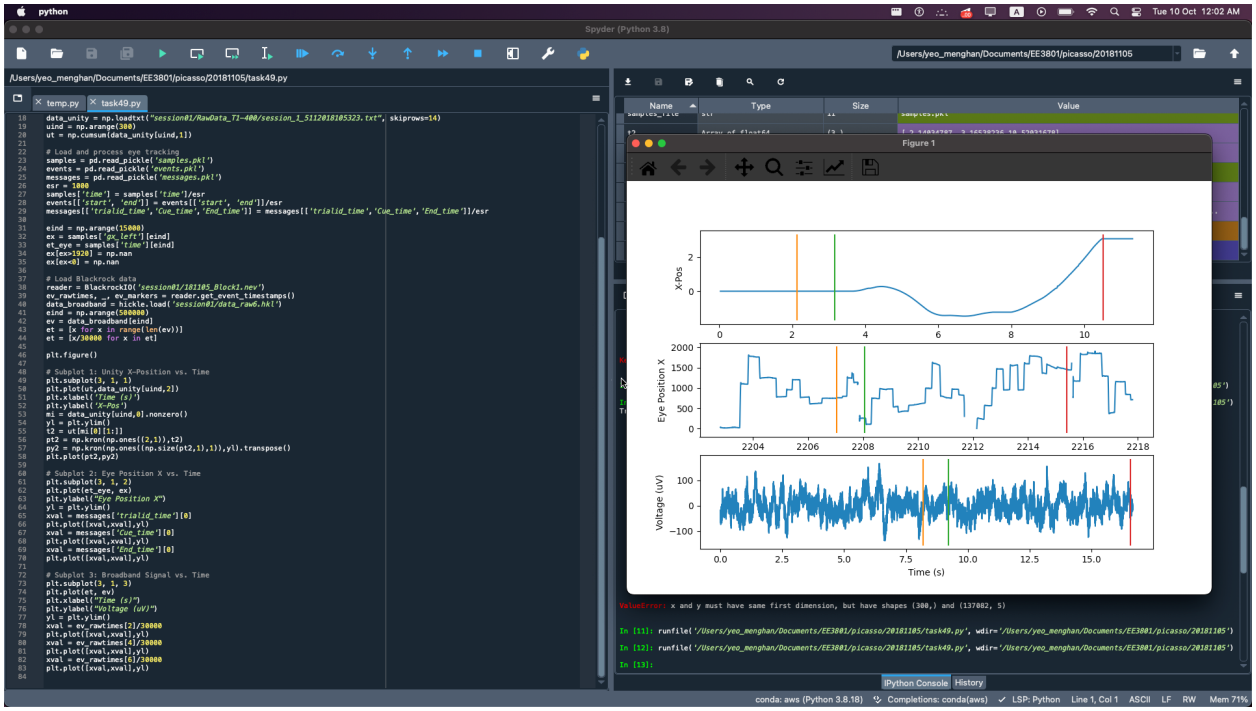
```
plt.xlabel("Time (s)") # Label for the x-axis
plt.ylabel("Voltage (uV)") # Label for the y-axis

# Get the y-axis limits of the plot
yl = plt.ylim()

# Plot vertical lines at specific time points based on 'ev_rawtimes'
xval = ev_rawtimes[2]/30000
plt.plot([xval, xval], yl) # Vertical line at the third event time
xval = ev_rawtimes[4]/30000
plt.plot([xval, xval], yl) # Vertical line at the fifth event time
xval = ev_rawtimes[6]/30000
plt.plot([xval, xval], yl) # Vertical line at the seventh event time
```

Task 49:

Now that we have looked at all three data files, we will look at how to align the three time-series to each other. Write a Python script that will create a figure with three subplots that contain

- *the x-position in Unity as a function of time;*
- *the eye position x as a function of time, and*
- *the broadband signal as a function of time.*

Remember to add the markers to each of the subplots like you did previously.

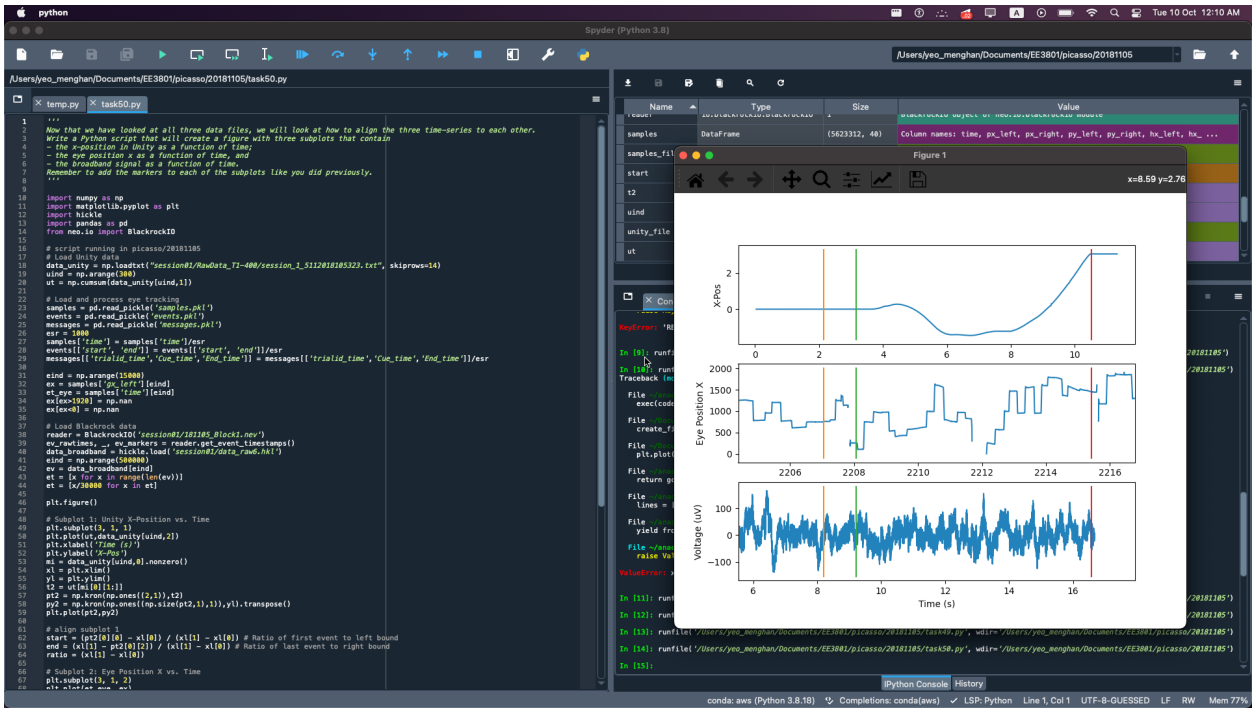



```
import numpy as np
import matplotlib.pyplot as plt
import hickle
import pandas as pd
from neo.io import BlackrockIO

# script running in picasso/20181105
# Load Unity data
data_unity = np.loadtxt("session01/RawData_T1-400/session_1_5112018105323.txt",
skiprows=14)
uind = np.arange(300)
ut = np.cumsum(data_unity[uind,1])

# Load and process eye tracking
samples = pd.read_pickle('samples.pkl')
```

```
events = pd.read_pickle('events.pkl')
messages = pd.read_pickle('messages.pkl')
esr = 1000
samples['time'] = samples['time']/esr
events[['start', 'end']] = events[['start', 'end']]/esr
messages[['trialid_time','Cue_time','End_time']] =
messages[['trialid_time','Cue_time','End_time']]/esr

eind = np.arange(15000)
ex = samples['gx_left'][eind]
et_eye = samples['time'][eind]
ex[ex>1920] = np.nan
ex[ex<0] = np.nan

# Load Blackrock data
reader = BlackrockIO('session01/181105_Block1.nev')
ev_rawtimes, _, ev_markers = reader.get_event_timestamps()
data_broadband = hickle.load('session01/data_raw6.hkl')
eind = np.arange(500000)
ev = data_broadband[eind]
et = [x for x in range(len(ev))]
et = [x/30000 for x in et]

plt.figure()

# Subplot 1: Unity X-Position vs. Time
plt.subplot(3, 1, 1)
plt.plot(ut,data_unity[uind,2])
plt.xlabel('Time (s)')
plt.ylabel('X-Pos')
mi = data_unity[uind,0].nonzero()
yl = plt.ylim()
t2 = ut[mi[0][1:]]
pt2 = np.kron(np.ones((2,1)),t2)
py2 = np.kron(np.ones((np.size(pt2,1),1)),yl).transpose()
plt.plot(pt2,py2)

# Subplot 2: Eye Position X vs. Time
```

```
plt.subplot(3, 1, 2)
plt.plot(et_eye, ex)
plt.ylabel("Eye Position X")
yl = plt.ylim()
xval = messages['trialid_time'][0]
plt.plot([xval,xval],yl)
xval = messages['Cue_time'][0]
plt.plot([xval,xval],yl)
xval = messages['End_time'][0]
plt.plot([xval,xval],yl)

# Subplot 3: Broadband Signal vs. Time
plt.subplot(3, 1, 3)
plt.plot(et, ev)
plt.xlabel("Time (s)")
plt.ylabel("Voltage (uV)")
yl = plt.ylim()
xval = ev_rawtimes[2]/30000
plt.plot([xval,xval],yl)
xval = ev_rawtimes[4]/30000
plt.plot([xval,xval],yl)
xval = ev_rawtimes[6]/30000
plt.plot([xval,xval],yl)
```

Task 50: *Now write a Python script that will align the three time series to each other, and create a new figure with three subplots that shows the aligned data. In order to perform the alignment, think about what is common in all three time series. Add code that will set the x-limits for all the subplots to be the same.*




```
import numpy as np
import matplotlib.pyplot as plt
import hickle
import pandas as pd
from neo.io import BlackrockIO

# script running in picasso/20181105
# Load Unity data
data_unity = np.loadtxt("session01/RawData_T1-400/session_1_5112018105323.txt",
skiprows=14)
uind = np.arange(300)
ut = np.cumsum(data_unity[uind,1])

# Load and process eye tracking
samples = pd.read_pickle('samples.pkl')
events = pd.read_pickle('events.pkl')
messages = pd.read_pickle('messages.pkl')
esr = 1000
samples['time'] = samples['time']/esr
```

```
events[['start', 'end']] = events[['start', 'end']]/esr
messages[['trialid_time','Cue_time','End_time']] =
messages[['trialid_time','Cue_time','End_time']]/esr

eind = np.arange(15000)
ex = samples['gx_left'][eind]
et_eye = samples['time'][eind]
ex[ex>1920] = np.nan
ex[ex<0] = np.nan

# Load Blackrock data
reader = BlackrockIO('session01/181105_Block1.nev')
ev_rawtimes, _, ev_markers = reader.get_event_timestamps()
data_broadband = hickle.load('session01/data_raw6.hkl')
eind = np.arange(500000)
ev = data_broadband[eind]
et = [x for x in range(len(ev))]
et = [x/30000 for x in et]

plt.figure()

# Subplot 1: Unity X-Position vs. Time
plt.subplot(3, 1, 1)
plt.plot(ut,data_unity[uind,2])
plt.xlabel('Time (s)')
plt.ylabel('X-Pos')
mi = data_unity[uind,0].nonzero()
xl = plt.xlim()
yl = plt.ylim()
t2 = ut[mi[0][1:]]
pt2 = np.kron(np.ones((2,1)),t2)
py2 = np.kron(np.ones((np.size(pt2,1),1)),yl).transpose()
plt.plot(pt2,py2)

# align subplot 1
start = (pt2[0][0] - xl[0]) / (xl[1] - xl[0]) # Ratio of first event to left
bound
```

```
end = (xl[1] - pt2[0][2]) / (xl[1] - xl[0]) # Ratio of last event to right
bound
ratio = (xl[1] - xl[0])

# Subplot 2: Eye Position X vs. Time
plt.subplot(3, 1, 2)
plt.plot(et_eye, ex)
plt.ylabel("Eye Position X")
xl = plt.xlim()
yl = plt.ylim()
xval1 = messages['trialid_time'][0]
plt.plot([xval1,xval1],yl)
xval = messages['Cue_time'][0]
plt.plot([xval,xval],yl)
xval2 = messages['End_time'][0]
plt.plot([xval2,xval2],yl)
xleft = (xval1 - (ratio / (xl[1] - xl[0])) * start * (xl[1] - xl[0]))
xright = ((xval2 + (ratio / (xl[1] - xl[0])) * end * (xl[1] - xl[0])))
plt.xlim(xleft, xright)

# Subplot 3: Broadband Signal vs. Time
plt.subplot(3, 1, 3)
plt.plot(et, ev)
plt.xlabel("Time (s)")
plt.ylabel("Voltage (uV)")
yl = plt.ylim()
xval1 = ev_rawtimes[2]/30000
plt.plot([xval1,xval1],yl)
xval = ev_rawtimes[4]/30000
plt.plot([xval,xval],yl)
xval2 = ev_rawtimes[6]/30000
plt.plot([xval2,xval2],yl)
xleft = (xval1 - (ratio / (xl[1] - xl[0])) * start * (xl[1] - xl[0]))
xright = ((xval2 + (ratio / (xl[1] - xl[0])) * end * (xl[1] - xl[0])))
plt.xlim(xleft, xright)
```