Introduction

These data come from a lab session done this semester in which students completed a baseline countermovement vertical jump (CMJ), a 15-second Wingate test againts 7.5% of their body mass, and then another CMJ. Goal of the lab was to introduce students to experimental designs by measuring something prior to and after (CMJ performance) some kind of intervention, in this case the intervention was the Wingate test. As well as introduce the energy systems associated with the Wingate test. jlk For their lab reports, students were given discrete variables all calculated via the Hawkin force plates such as:

- *Jump height (cm) via impulse-momentum.*
- *Modified reactive strength index [mRSI (AU)].*

For my own purposes of learning and improving my skills as a Python coder, I chose to use the spm1d package for Statistical Parameteric Mapping (SPM) of the CMJ force time curves prior to (PRE) and after (POST) the Wingate. This is a package I have been wanting to experiment with for a while. Since in exercise and sports science, we spend so much time on discrete (peaks, means, etc.) variables but collect much more data than we actually analyze. SPM allows for the analysis of the entire force-time curve.

```python
In [1]:  # import packages
         import pandas as pd
         import numpy as np
         import os
         import matplotlib.pyplot as plt
         from tkinter.filedialog import askdirectory
         import spm1d

         # defining plot theme.
         custom_theme = {"axes.spines.right": False, "axes.spines.top": False,
                         "axes.titlelocation": "center", "axes.titley": 1,
                         "font.weight":"bold", "axes.titlesize": "x-large", "axes.labelsize"
                         "axes.titleweight": "bold", "axes.labelweight": 'bold',
                         'font.family': 'Tahoma'}

         plt.rcParams.update({**custom_theme})
```

Data Loading

Select the directory where the files are stored (clone the git repo if necessary). Print the file names and check the number of files that will be analyzed.

```python
In [2]:  dir_to_read = askdirectory(title = 'Select Directory to Read CMJ Files')
         files_to_read = os.listdir(dir_to_read)
```

```
for file_to_print in files_to_read:
    print(file_to_print[:-4])
print(f'\nThere are {len(files_to_read)} files.')
```

```
001-POST
001-PRE
002-POST
002-PRE
003-POST
003-PRE
004-POST
004-PRE
005-POST
005-PRE
006-POST
006-PRE
007-POST
007-PRE
008-POST
008-PRE
009-POST
009-PRE

There are 18 files.
```

Custom Function for Analyzing CMJ Force Time Arrays

Next, I'm writing a custom function to read the force arrays, identify the indices we need, crop the arrays to those indices, and then interpolate the data to 101 data points (0-100% of the CMJ phases of interest).

Specifically, the CMJ force time data will be cropped from the point of movement initation ('start_move') to takeoff ('takeoff').

The index identification portion of the below code has been adapted from Merrigan et al..

In [3]:
```python
# This is a custom function to read, analyze (find the indices of interest), crop,
sf = 1000 # Hawkin sampling frequency = 1000 Hz

def read_and_crop_and_interpolate_cmj(file_path):
    full_data = pd.read_csv(file_path)
     # data are stored in the first column from how
     # I set it up to export from R's {HawkinR} package.
    fz_total = full_data.iloc[:,0]

    # calculate baseline fz ('weighing phase') - mean of first 1-second of data.
    bw_mean = fz_total[0:1000].mean()

    # calculate baseline standard deviation for determination of movement start.
    bw_sd = fz_total[0:1000].std()

    # convert body weight in Newtons to body mass in kilograms.
```

```python
    bodymass = bw_mean/9.81

    # this determines when movement begins, temp assign to index 20
    start_move = 20

    # while loop to identify the point at which the fz_total
    # array deviates from the mean by 5 standard deviations
    while fz_total[start_move] > (bw_mean - (bw_sd * 5)):
        start_move = start_move + 1

    # backtrack start_move to within 1 standard deviation of the baseline force
    while fz_total[start_move] < (bw_mean - (bw_sd)):
        start_move = start_move -1

    # identify takeoff index of original dataframe as when force drops below 30 N
    # first say that takeoff index = start_move index
    takeoff = start_move
    while fz_total[takeoff] > 30:
        takeoff = takeoff + 1

    # crop the array from start movement to takeoff
    cropped_fz_total_original = fz_total[start_move:takeoff]

    # normalize to bodymass in kilos so N/kg is comparable between subjects
    cropped_fz_normalized = cropped_fz_total_original/bodymass

    # interpolate to 101 datapoints
    original_data = np.linspace(0, len(cropped_fz_normalized)-1, len(cropped_fz_nor
    interpolated_indices = np.linspace(0, len(cropped_fz_normalized)-1, 101)
    force_interpolated = np.interp(interpolated_indices, original_data, cropped_fz_

    # return the cropped, normalized and interpolated array
    return(force_interpolated)
```

We just wrote that custom function, now apply it in a for loop over the files we need to read, crop, normalize, and interpolate.

For sanity's check as well, we will be creating a Spaghetti plot of all the arrays in the loop.

In [4]:
```python
# preallocate a panda's dataframe to store the data in
output_data = pd.DataFrame()

# initalize a plot
plt.figure(figsize = (10, 6))

for file_name in files_to_read:
    # get the full file path by joining the directory we selected with the filename
    file_path_of_choice = os.path.join(dir_to_read, file_name)

    # apply the function
    cleaned_and_interpolated_data = read_and_crop_and_interpolate_cmj(file_path=fil

    # remove the .csv from the filename
    base_file_name = file_name[:-4]
```
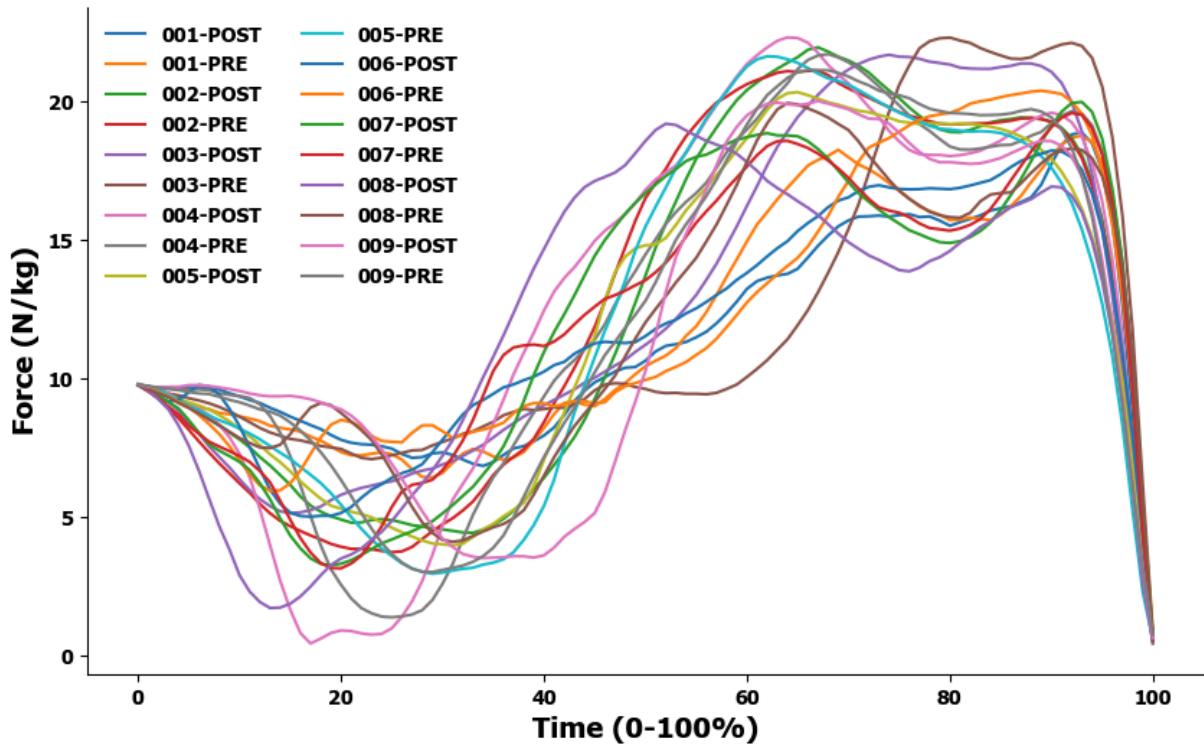
```
    # append the dataframe with the values we just generated...
    output_data[base_file_name] = cleaned_and_interpolated_data

    # plot the data for sanity checks. always.
    plt.plot(cleaned_and_interpolated_data, label = base_file_name)

# outside of the loop, figure theme settings
plt.legend(loc = 'upper left', frameon = False, ncol = 2)
plt.ylabel('Force (N/kg)')
plt.xlabel('Time (0-100%)');
```


Things look good! All CMJs have been cropped, interpolated, and normalized correctly. Let's check the dataframe now.

There is one (009-POST) that does not look like it had the best unweighing phase. For the purposes of this demonstration, however, I am not too concerned about that.

Data Analysis

The next chunk will create the two arrays we will be analyzing in spm1d at - PRE vs. POST.

1. First, filter the dataframes into post_df and pre_df based on column names
2. Then transpose the dataframes into a J x Q matrix (subjects [rows] X nodes/points per trial [columns])
3. Lastly, transfer the dataframe into an array for ease.

```
In [5]:  post_df = output_data.loc[:, output_data.columns.str.contains('POST')]
         pre_df = output_data.loc[:, output_data.columns.str.contains('PRE')]

         post_transpose = post_df.T
         pre_transpose = pre_df.T

         pre_ft = np.array(pre_transpose)
         post_ft = np.array(post_transpose)
```

Always Plot Your Data!

The next code chunk will plot mean±SD values for PRE and POST prior to analysis. I will,
similar to the plot above, overlay the individual responses as well.

Very simply, calculate the mean for each data point (point 0, point 1, point 2, ... point 100).
Calculate the SD for each data point. Plot the mean data, plot the individual data, have the
SD be as a shadow around the mean data. Label appropriately.

```
In [6]:  # calcualte mean/sd for pre
         pre_mean = pre_ft.mean(axis = 0)
         pre_sd = pre_ft.std(axis = 0)

         # calculate mean/sd for post
         post_mean = post_ft.mean(axis = 0)
         post_sd = post_ft.std(axis = 0)

         # make an x values array for plotting purpsoes
         x_vals = np.arange(101)

         # initialize plot
         fig, axes = plt.subplots(1, 2, figsize=(10, 6)) # call plot subplots

         # on axis 0 = PRE
         axes[0].plot(x_vals, pre_mean, color='blue', linewidth=4, label = 'Mean±SD')
         # for loop to overlay the individaul responses
         for i, data_to_plot in enumerate(pre_ft):
             label = 'Individual Subject' if i == 0 else None
             # brute force way to add
             # a legend depicting what the lines mean, just assigning the label for only the
             axes[0].plot(x_vals, data_to_plot, color = 'black', alpha = 0.21, label = label
         axes[0].fill_between(x_vals, pre_mean - pre_sd, pre_mean + pre_sd, color='blue', al
         axes[0].set_xlim(0, 100)
         axes[0].set_ylim(0, 30)
         axes[0].set_ylabel('Force (N/kg)')
         axes[0].set_title('PRE')
         axes[0].legend(frameon = False, loc = 'upper left')

         # on axis 1 = POST
         axes[1].plot(x_vals, post_mean, color='red', linewidth=4, label = 'Mean±SD')
         # for loop again
         for i, data_to_plot in enumerate(post_ft):
             label = 'Individual Subject' if i == 0 else None
             axes[1].plot(x_vals, data_to_plot, color = 'black', alpha = 0.25, label = label
```
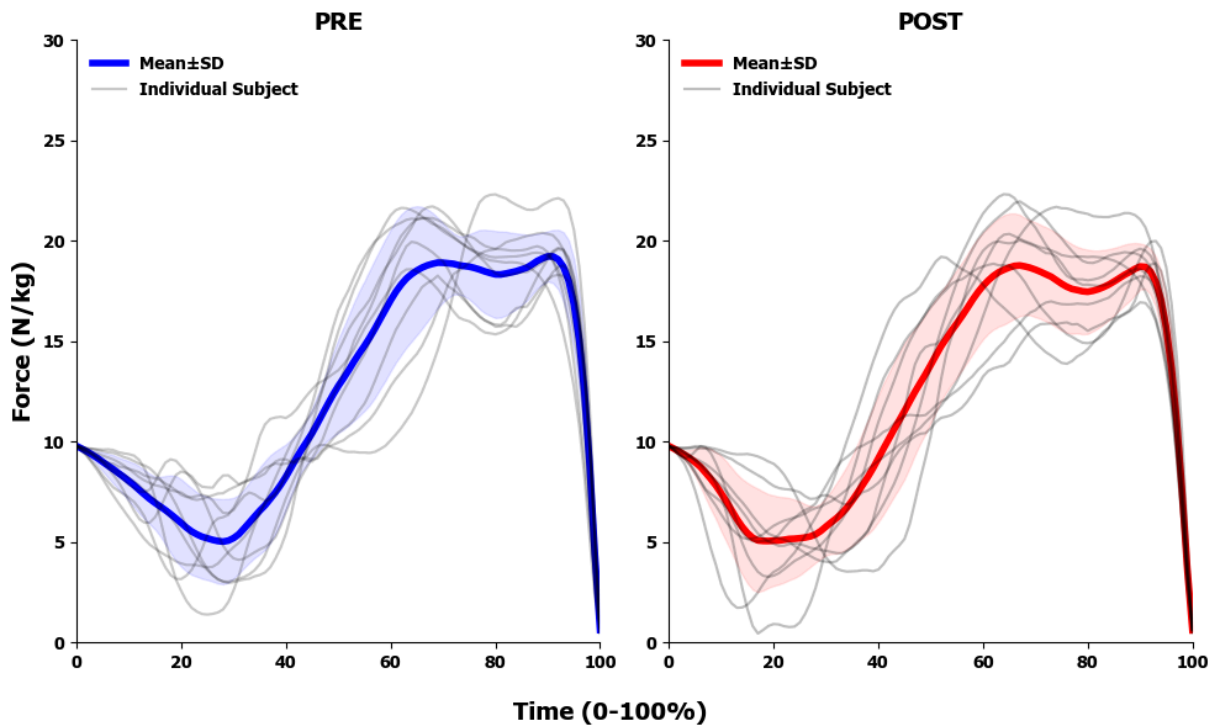
```
axes[1].fill_between(x_vals, post_mean - post_sd, post_mean + post_sd, color='red',
axes[1].set_xlim(0, 100)
axes[1].set_ylim(0, 30)
axes[1].set_title('POST')

axes[1].legend(frameon = False, loc = 'upper left')

# add a shared x-axis title
fig.supxlabel('Time (0-100%)', fontsize='x-large', fontweight='bold')

plt.tight_layout();
```


Running the SPM1D Analysis

Now, this next code block will run the spm1d analyses. Since we have a within-subjects design, spm1d's paired t-test function will be used. The second block of this chunk will return the inferential test.

```
In [7]:  # assign the output to t and then view it
         t = spm1d.stats.ttest_paired(pre_ft, post_ft)

         # assign alpha
         alpha = 0.05
         ti = t.inference(alpha = alpha)
         ti
```

`SPM{t} inference field`
```
    SPM.z          :  (1x101) raw test stat field
    SPM.df         :  (1, 8)
    SPM.fwhm       :  4.15726
    SPM.resels     :  (1, 24.05429)
  Inference:
    SPM.alpha      :  0.050
    SPM.zstar      :  5.56250
    SPM.h0reject   :  False
    SPM.p_set      :  1.000
    SPM.p_cluster  :  ()
```
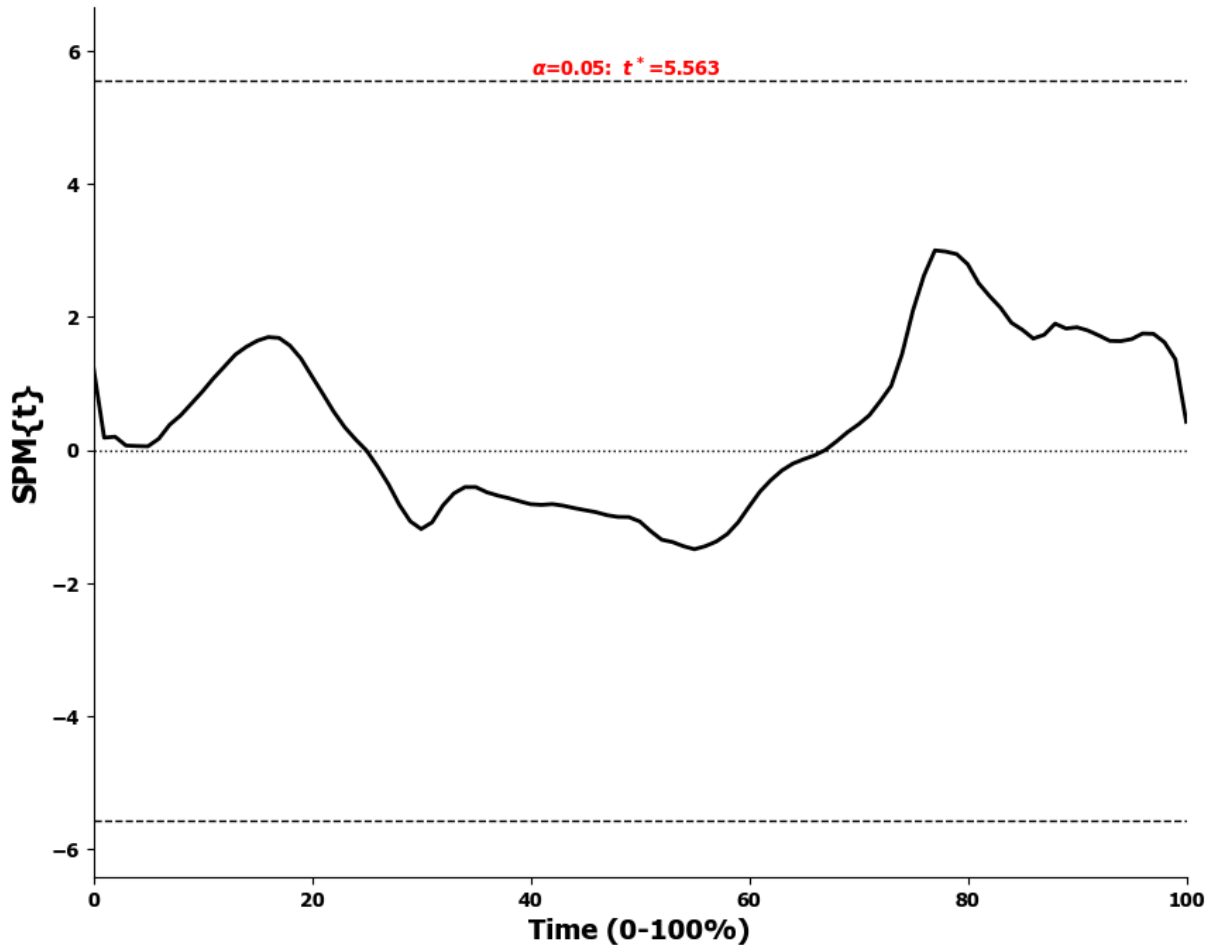
Plotting the SPM results.

```python
plt.figure(figsize = (10,8))
ti.plot()
ti.plot_threshold_label()
plt.xlabel('Time (0-100%)');
```


Conclusion

From the results of the SPM analysis, there was no effect of the 15-second Wingate test on CMJ performance. Why? The population was solely student-athletes and 15-seconds of a

Wingate may not have resulted in enough fatigue to interrupt their CMJ strategy.