# Lab 7 - More Data!

Today, we will learn to process each subject's data, aggregate all the subject's data into one data frame, and then visualize the grouped data!

To follow along, please download all the materials (including the .csv data files) from the Week 7 Lab folder.

## Randomization Review

Most studies will involve multiple counterbalanced or experimental conditions. You would aim to have the number of subjects in each condition to be equal and randomly assign participants to one of your conditions.

For your purposes, because we know all the subjects, we will do the random assignment ahead of time. **For your final projects, you will need to randomly assign your classmates to one of the conditions in your experiment.**

We will go over an example of this process in a bit. But first, I want to review the `shuffle` function from the `numpy.random` library that we used when coding our PsychoPy experiment.

### Random Seed

With this approach, it's necessary to randomly shuffle the image sequence. You'll need to import the random module to do this.

**Ex 1: Using the `random.shuffle` method:**

In [ ]:
```
# import the shuffle function from numpy.random (this is automatically done inside Psychoi
import numpy as np
from numpy.random import shuffle


seq = list(range(1,51))
print(seq)
shuffle(seq)
print(seq)
```

In [ ]:
```
# define a sequence to randomize:
sequence = [1, 2, 3, 4, 5]

# shuffle this sequence a few times:
for i in range(4):
    shuffle(sequence)
    print(sequence)
```

**Ex 2: Let's do the same thing as Ex 1, but this time we will first set a randomization seed:**

In [ ]:
```
# define a sequence to randomize:
sequence = [1, 2, 3, 4, 5]

# set a randomization seed (np.random.seed expects an integer)
np.random.seed(123)

# shuffle this sequence a few times:
for i in range(4):
    shuffle(sequence)
    print(sequence)
```

Notice what happens to the printed sequences when I repeatedly run the code block in Ex 1 versus when I run the code block in Ex 2? What is going on and why?

**Ex 3: Let's now re-seed with the same seed as Ex 2, and run the loop of shuffles again. Compare this shuffled sequence with the first sequence printed in Ex 2:**

In [ ]:

```
# define a sequence to randomize:
sequence = [1, 2, 3, 4, 5]

# set a randomization seed (np.random.seed expects an integer)
np.random.seed(123)

# shuffle this sequence a few times:
for i in range(4):
    shuffle(sequence)
    print(sequence)
```

You can think of a seed as a password that unlocks the sequences of your psuedo-random number generator (PRNG). If you pass the same seed to the same PRNG algorithm, you will unlock the same (psuedo)randomizated sequences. This can be an issue if you're not careful! For example, the default randomization seed for Matlab's PTB Shuffle function is the same whenever you launch a new session of Matlab. If you forget to re-seed after you start a new Matlab session you'd be "randomizing" with the same seed, and all your subjects may be getting the same sequence even though you thought they were random!

The Python `np.random` module defaults to the OS time as the seed, and so it's not as much of a problem for PsychoPy experiments; however, it is in some cases important to know what the unique PRNG seed is for a subject in case you need to replicate the randomizations in a later session. In your final projects, we expect you to seed your experiments with the participant's ID (e.g., `np.random.seed(int(expInfo['participant']))` ).

## Random Assignment

For your final project, it is expected that you randomly assign your classmates to your experimental conditions before data collection. This is probably the easiest approach when you're running a study with a relatively small sample.

### Steps:

1. Create a spreadsheet that matches your participant's name to their experiment ID (e.g., "id_name_condition.csv" file in the lab materials). Leave the condition column empty
2. Randomly assign each participant id to one of your experiment conditions.
3. Fill in the randomly assigned conditions columns according to your randomization code's output.

In my example redlight-greenlight experiment, I want a "high" penalty condition where false alarms are penalized with 5 steps back, and a "low" penalty condition where false alarms are penalized with 1 step back.

In [ ]:

```
participant_ids = list(range(101,112))

# ids before shuffling:
print(participant_ids)

# shuffle ids
shuffle(participant_ids)
```

```
# ids after shuffling:
print(participant_ids)

# assign first 4
half_subs_n = round(len(participant_ids)/2)
all_subs_n = len(participant_ids)
low_penalty_ids = participant_ids[0:half_subs_n]
high_penalty_ids = participant_ids[half_subs_n:all_subs_n]

print(low_penalty_ids)
print(high_penalty_ids)
```

From here, you can label the each row of the conditions column with the condition label (i.e., 5 or 1) depending on which list the subject id falls in (high or low, respectively). This is the simplest way to do it (i.e., manually), and is maneageable because the class size is small. However, a better way to do it is programatically - and at this point, you have the necessary knowledge to implement this!

### Important note!

In a real study, you would need to de-identify your data, meaning that the name of the participant should not be traced back to the data. Thus, it's typical to have a file matching the participant id to participant's name, and a separate file matching the participant's id to their condition/other data. For ease of providing your classmates instructions, we will just use one sheet.

### For data collection:

Please provide your classmates with instructions on how to fill in their participant id and condition in the experiment info dialog that appears when you start running the experiment.

## Data Processing

The first 3 steps recaps what we learned already about processing your raw data files and labeling each trial with SDT terms (i.e., hits, fa, cr, misses).

I've generated several data files using the experiment that we created last class.

### Steps:

1. **Import modules**

In [ ]:
```
import pandas as pd
```

1. **Read in data as a data frame**

In [ ]:
```
filename = 'sub-101_fa-1_redlight_greenlight.csv'

subject_df = pd.read_csv(filename)

subject_df
```

1. **Test whether each row (i.e., trial) is a "HIT", "FA", "MISS", or "CR"**. We will do this by creating logical arrays. To test if a value is NaN, use `.isna`

In [ ]:
```
# First, let's generate logical arrays for each signal detection label:
hit_la = (subject_df["resp"] == "space") & (subject_df["trial_type"] == "go")
fa_la = (subject_df["resp"] == "space") & (subject_df["trial_type"] == "nogo")
cr_la = (subject_df["resp"] == '[]') & (subject_df["trial_type"] == "nogo")
miss_la = (subject_df["resp"] == '[]') & (subject_df["trial_type"] == "go")

# Optionally, add a column called sdt_label, which labels each trial
subject_df.loc[hit_la, "sdt_label"] = "hit"
subject_df.loc[fa_la, "sdt_label"] = "fa"
subject_df.loc[cr_la, "sdt_label"] = "cr"
subject_df.loc[miss_la, "sdt_label"] = "miss"

subject_df
```

1. **Calculate Hit Rate and FA Rate**

In [ ]:
```
# Count the number of hits and FAs using the logical arrays
hit_count = sum(hit_la)
fa_count = sum(fa_la)

# Option 2, count the number of hits and FAs using the sdt_label column:


# Count the number of nogo trials or go trials
go_count = sum(subject_df["trial_type"] == 'go')
nogo_count = sum(subject_df["trial_type"] == 'nogo')

# Calculate Hit Rate as hit_count/go_count; do same for fa_rate (fa_count/nogo_count)
hit_rate = hit_count/go_count
fa_rate = fa_count/nogo_count

print(hit_rate)
print(fa_rate)
```

1. **Calculate $d_{prime}$ and $\lambda$ assuming equal variance Gaussian model**. For this, you need to install and import the `scipy` library. We can use some of the stats methods to easily get the Z values from probabilities. Documentations for the stats methods here: https://scipy.github.io/devdocs/tutorial/stats.html

In [ ]:
```
import scipy.stats as st

# Convert the hit and fa rates to Z_hit and Z_fa
Z_hit = st.norm.ppf(hit_rate)
Z_fa = st.norm.ppf(fa_rate)
```

In [ ]:
```
# Calculate lambda (Book Eq'n 2.3)
lam = -Z_fa

# Calculate d-prime (Book Eq'n 2.4)
d_prime = Z_hit - Z_fa

print(lam)
print(d_prime)
```

1. **Calculate bias metrics $log\beta$ and $\lambda_{center}$**

In [ ]:
```
# Calculate lam_center (Book Eq'n 2.5)
lam_center = lam - .5*d_prime
```

```
# What's an alternative what to calculate lam_center from Z_hit and Z_fa? (Book Eq'n 2.6)
lam_center = -.5*(Z_fa + Z_hit)

# Calculate log_beta (Book Eq'n 2.10)
log_beta = d_prime*(lam - .5*d_prime)

log_beta
```

Based on these bias metrics, is the subject more biased towards the signal or the noise distribution?

1. **Record the all the metrics you just calculated in a list.** Also include Participant ID and the Condition.

When dealing with strings, you'll often need to parse and pattern match. This falls into the world of **regular expressions** or **RegEx**. Python has a base regex library called `re` (https://docs.python.org/3/library/re.html). We will use this to grab participant id and condition from `filename` .

In [ ]:

```
# This is a base python library for regular expressions.
import re

print(filename)
re.split('-|_',filename)
```

In [ ]:

```
# Extract Participant ID and Condition from the filename
_, sub, _, cond, _, _ = re.split('-|_',filename)

print(sub)
print(cond)
```

In [ ]:

```
# Include sub, cond, and metrics into a list
subject_list = [sub, cond, hit_rate, fa_rate, lam, d_prime, lam_center, log_beta]
subject_list
```

1. **Now, put all the previous steps in a loop!** Loop through all the data files that you collected.

Use glob to find all the _redlight_greenlight.csv files in this directory. This is a simple pattern matching way of retrieving only files that match a certain format.

In [ ]:

```
import glob

data_filenames = glob.glob("*_redlight_greenlight.csv")

data_filenames
```

Simply iterate through each `filename` in `data_filenames` and copy the steps above into the code chunk:

In [ ]:

```
aggregated_list = []
for filename in data_filenames:

    # Read filename into a data frame
    subject_df = pd.read_csv(filename)

    # First, let's generate logical arrays for each signal detection label:
    hit_la = (subject_df["resp"] == "space") & (subject_df["trial_type"] == "go")
    fa_la = (subject_df["resp"] == "space") & (subject_df["trial_type"] == "nogo")
    cr_la = (subject_df["resp"] == "[]") & (subject_df["trial_type"] == "nogo")
    miss_la = (subject_df["resp"] == "[]") & (subject_df["trial_type"] == "go")

    # Count the number of hits and FAs using the logical arrays
    hit_count = sum(hit_la)
    fa_count = sum(fa_la)

    # Count the number of nogo trials or go trials
    go_count = sum(subject_df.trial_type == 'go')
    nogo_count = sum(subject_df.trial_type == 'nogo')

    # Calculate Hit Rate as hit_count/go_count; do same for fa_rate (fa_count/nogo_count)
    hit_rate = hit_count/go_count
    fa_rate = fa_count/nogo_count

    # adjust "extreme" hit rates to avoid Inf z values (see section 2.3 of your book for 
    if hit_rate == 1:
        hit_rate = 1 - 1/go_count
    if fa_rate == 0:
        fa_rate = 0 + 1/nogo_count

    # Convert the hit and fa rates to Z_hit and Z_fa
    Z_hit = st.norm.ppf(hit_rate)
    Z_fa = st.norm.ppf(fa_rate)

    print(Z_hit)
    print(Z_fa)

    # Calculate lambda (Book Eq'n 2.3)
    lam = -Z_fa

    # Calculate d-prime (Book Eq'n 2.4)
    d_prime = Z_hit - Z_fa

    # Calculate lam_center (Book Eq'n 2.6)
    lam_center = lam - .5*d_prime

    # Calculate log_beta (Book Eq'n 2.10)
    log_beta = d_prime*(lam - .5*d_prime)

    # Extract Participant ID and Condition from the filename
    _, sub, _, cond, _, _ = re.split('-|_',filename)

    # Include sub, cond, and metrics into a list
    subject_list = [sub, cond, hit_rate, fa_rate, lam, d_prime, lam_center, log_beta]

    # Append subject_list to aggregated_list
    aggregated_list.append(subject_list)

aggregated_list
```

We now have an `aggregated_list` which is a nested list that contains all of the signal detection metrics for all our participants. Let's put it into a data frame.

In [ ]:
```
# Put aggregated_list in a data frame, specifying column headers
aggregated_df = pd.DataFrame(aggregated_list,
                             columns = ["subject_id", "fa_penalty", "hit_rate", "fa_rate", "la

aggregated_df
```

In [ ]:
```
# Let's save this  data frame as a csv
aggregated_df.to_csv('aggregated_data.csv')
```

## Visualization

When it comes to visualizing your data in python, you have *many* options (matplotlib, seaborn, plotly, plotnine). We will *briefly* go over some `matplotlib` and `plotnine` implementation for your experiment.

### Creating the noise and signal Gaussian distributions

Going into matplotlib is beyond the scope of the class, but here's a quick demo of how to plot the signal and noise distributions from your aggregated dataset.

Note that in your study, you would want to create separate distribution plots for each of your experimental conditions.

Once you have this SDT distributions, you can label the important characteristics of the plot (e.g., add a d' label, and a criterion vertical line and label it, etc.). This can be done programmatically (beyond the scope of today), or probably easier is to just export these plots and edit it in powerpoint or illustrator.

In [ ]:
```
import matplotlib.pyplot as plt
import scipy.stats as st

# find d' (aka mean of the signal when you're in the noise Z-space)
signal_mean = sum(aggregated_df['d_prime'])/len(aggregated_df['d_prime'])

# define the x and y variables of your two plots (x array is shared by the two plots)
x = np.arange(-4, 6, 0.01)     # the first two arguments can effectively be how you contro
noise_y = st.norm.pdf(x, 0, 1)  # by convention, noise mean is 0 and noise variance is 1
signal_y = st.norm.pdf(x, signal_mean, 1)    # assuming eq variance gaussian model, signal

# display the guassian distributions
f = plt.figure()
plt.plot(x, signal_y)
plt.plot(x, noise_y)

# save figure (note that I recommend figures in a vector format like svg)
plt.savefig('sdt_distributions.svg')
```

In [ ]:
```
from plotnine import ggplot, aes, geom_point, geom_col, geom_boxplot

ggplot(aggregated_df) + aes(x="fa_penalty", y="hit_rate") + geom_boxplot() + geom_point()
```

In [ ]:
```
ggplot(aggregated_df) + aes(x="fa_penalty", y="fa_rate") + geom_boxplot() + geom_point()
```

### Plot the bias metrics

How might we predict participants to respond when there's a greater FA penalty? Would they press the spacebar more often or less often?

Which of our two conditions (5-penalty vs 10-penalty) would have a more positive log_beta?

In [ ]:
```
ggplot(aggregated_df) + aes(x="fa_penalty", y="log_beta") + geom_boxplot() + geom_point()
```

## Further resources for curating stimuli!

I have access to some libraries of images and some audio stimuli. If any of these seem relevant to your experiment, please let me know and I can send you the files:

1. Animals
2. Tools
3. Produce
4. Line drawings (Snodgrass and Vanderwart; Nishimoto)
5. Emotional images (IAPS)
6. Bank of Standardized Stimuli (BOSS)
7. Open Affective Standardized Image Set (OASIS)
8. Lots of random objects
9. Emotional auditory stimuli (IADS)

There are also fun new tools that implement AI that generates stimuli that you can download. DALL-E2 from OpenAI has blown up in popularity recently since they launched a web app that allows you to generate wild and crazy images using any prompt (https://labs.openai.com/Links to an external site.). There are also many face-generating tools (e.g., https://this-person-does-not-exist.com/enLinks to an external site.) if your study involves faces and you want to use those images. You can also use google images, Pixabay, or Wikimedia Commons to search for free and open access images.

In terms of editing the stimuli, I highly recommend using a free software called XnConvert to batch process your images (e.g., you can turn all images to grayscale, resize, crop, desaturate, recolor, etc.). As UCLA students, you also have access to Adobe tools like Photoshop if you need to do more detailed photo editing. If you're working with audio files, Audacity is a very useful free software with a lot of features, but I mainly use it to trim audio files or match volume levels between clips.

If you have any questions about stimuli curation, I can address your questions during lab on Monday and hopefully direct you to the right resource.