Lab 6 - Data!

Collecting Data From the Red Light, Green Light Game

To follow along, please download this jupyter notebook file from BruinLearn.

Reminders

```
1) proposals due to me and the professor tonight, along with this week's lab and
lecture assignments
2) if you need a partner for the lab project, please send me a email letting me
know
3) I will be holding a lab section on Monday, 11/7. Part of this time will be to
let you work on programming your experiments. I will be available to provide hand-
on help.
4) I will hang around after lab today and Monday if you have any questions about
homework and the final project.
5) If you want to meet with me next week for additional help on your assignments,
send me an email and we can schedule a Zoom meeting.
```

Python Packages

When programming, we may often find ourseleves needing a function that is not available in base Python. In these cases, you can turn to Python **packages**, which are open-source software developed by the Python community.

Two commonly used packages for data manipulation are `pandas` and `numpy`. We will be using functions from these packages to:

```
1) record useful data from each trial to a data frame
2) write this data frame to a csv file
3) open up this csv file and wrangle the data
4) create a simple visualization of the data
```

Installing Packages

To install packages, we can use `pip`. More information about `pip` can be found here: https://packaging.python.org/tutorials/installing-packages/

We can use `pip` to install `pandas`. `pip` will also install *dependencies*, or other Python packages, that are needed to run `pandas`. When you run the following command, `numpy` will also be installed because it is a dependency of `pandas`:

In [ ]:

```
# Windows Users (command prompt):
py -m pip install pandas

# Mac Users (terminal):
python3 -m pip install pandas
```

Note that the Python environment that ships with PsychoPy already installed these packages. You only need to install this if you want to follow along in Jupyter Notebook. If you don't have Jupyter Notebook, but still want to follow along, you can use the PsychoPy Coder window.

Intro to Numpy and Pandas

`numpy` and `pandas` are both very popular packages that are frequently used when we need to manipulate data.

Importing packages

In order to have access to a function during a session, we need to first import the packages. Let's import the `pandas` and `numpy` modules with the following variable names `pd` and `np` as shorthand to access methods in those packages:

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

The numpy array

The `numpy` array is another way to store data. Unlike `lists`, however, `numpy` arrays can only contains data of the same type.

What happens here when we create an array vs a list? Print the variables to see:

In [ ]:
```
my_array = np.array(["asdf",1,2,3])

my_list = ["asdf",1,2,3]

#print(my_array)
#print(my_list)

# my_list[1]
# my_array[1]
```

The np.array *coerces* the integers to a string.

Unlike lists, however, `numpy` arrays can be formatted in both rows and columns. This is helpful when we want to record data in PsychoPy experiments, because we usually want to add a new row of data for each trial.

In [ ]:
```
# Create a 3x3 array
my_array = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(my_array)
```

The pandas data frame

So what if we want to mix data types and still have a matrix-like format? Well, we can use the `pandas` data frame.

With the data frame, we can label the columns. One way to build data frames is by using a dictionary. Let's fabricate a data frame that you might get from our Red Light, Green Light game:

In [ ]:
```
data_dict = {'trial_type':['go','nogo','go','go'],'rt': [1.1,0,0,0.8],'resp':['space','NA'
```

```
data_df = pd.DataFrame(data_dict)

data_df
```

Adding trial-by-trial data

In our PsychoPy experiment, we will be adding a row to a nested list one trial at a time. Then turn this list to a data frame. Let's simulate how we'll be adding data to our list with each trial:

In [ ]:
```
# Note on variables:
#   trial_list: the "inner list" that temporarily stores the data of a given trial
#   session_list: the "nested list" that trial_list is appended to at the end of each tria

# These are the dummy "trial" data
trial_types = ['go', 'nogo', 'go', 'go', 'go', 'nogo', 'go']
rts = [1.1, 0, 1.4, 0, 0.8, 0.7, 1.2, 1.5]
resps = ["space", "NA", "space", "NA", "space", "space", "space", "space"]

# Initialize an empty list
session_list = list()

# imagine this for loop is like each iteration of your "trial" routine
for i in range(len(trial_types)):
    # Put the data for this trial in a dictionary:
    trial_list = [trial_types[i], rts[i], resps[i]]
    # Add this trial's dictionary data to the data frame:
    session_list.append(trial_list)

print(session_list)
```

Now that we have a `session_list` (which is a nested list where each sub-list contains the data from each trial), we can put it in a `pandas` data frame. We would usually put this code somewhere near the end of our PsychoPy experiment.

In [ ]:
```
# Put the trial
session_df = pd.DataFrame(session_list,
                          columns = ["trial_type", "rt", "resp"])

session_df
```

Write data to a .csv file

Finally, we need to output our data frame `session_df` to a .csv file. We need to specify a file name for this method - to create a unique data file, we'll usually include the subject ID in the file name. Let's look at an example with a hypothetical subject #103

In [ ]:
```
sub_id = 103

# to create the full file path, we need to concatenate strings:
output_filepath = 'sub-' + str(sub_id) + '_redlight_greenlight.csv'
print(output_filepath)
# write the session_df data frame object to a csv file:
session_df.to_csv(output_filepath)
```

Creating Custom Data Outputs in Red Light, Green Light

After each session, PsychoPy by default outputs a .psydat and .csv data file to a sub-folder called "data". These file formats are not designed in a way that is easy to use for data analysis (open up one of the .csv files to see for yourself). Because these default outputs are not formatted in a nice way, we will use `numpy` and `pandas` to create data files that are easier to read and use for our later data analyses.

In our data output file, we each row represents one trial. Our goal is to output a data file with 3 columns: 1) trial_type ("go" or "nogo") 2) resp ("space" or "NA") 3) rt (reaction time as a float)

To follow along, please download **lab6_demo_redlight_greenlight_data.psyexp** from the Week 6 folder on BruinLearn.

Before we get started, let's run this version of the task just to see what we have so far!

Steps:

1. Let's create a new code component in the `trial` routine called `log_data`
2. In the Begin Experiment tab, we will 1) import modules that we need and 2) initialize empty lists:

In [ ]:
```
import pandas as pd

session_list = list()
```

1. In the End Routine tab, we need to retrieve the `trial_type` , `resp` , and `rt` . We then put it in a list (trial-level), and then put that trial-level list in the session-level list:

In [ ]:
```
# REMINDER ABOUT VARIABLES:
#
#    1. sequence - refers to our sequence of go and nogo trials (go == 0; nogo == 1); this
#    2. trials - refers to the loop object that encapsulates our trial routine; this is cre
#    3. trials.thisRepN - is an attribute of the trials object that counts the # of iterati
#    4. task_kb - refers to the keyboard object created in our trial routine
#    5. task_kb.rt/task_kb.keys - if a kb input is detected by our task_kb object during ar
#    6. trial_list - a temporary list that we create here that contains all the important
#    7. session_list - at the end of each trial, trial_list is appended to this list to cre

# is this a nogo or a go trial?
if sequence[trials.thisRepN] == 0:
    trial_type = "go"
elif sequence[trials.thisRepN] == 1:
    trial_type = "nogo"

# Get rt and resp
rt = task_kb.rt
resp = task_kb.keys

# Add this to trial_list, then append trial_list to session_list:
trial_list = [trial_type, resp, rt]

session_list.append(trial_list)
```

1. Create a new routine called `finished` and put it at the end of the Flow timeline. Add a code component called `write_data` . Then write that data frame to an output csv file. This csv file needs to contain the participant ID:

In [ ]:
```
# REMINDER ABOUT VARIABLES:
#
```

```
#   1. session_list - the nested list that we append trial_list to at the end of each tria
#   2. expInfo - a dictionary that stores the information that we input in the dialog box

# Put session_list in a data frame
session_df = pd.DataFrame(session_list,
                          columns = ["trial_type", "resp", "rt"])

# Concatenate output file path; Question: maybe we want to store the fa_outcome because ou
output_filepath = 'sub-' + expInfo['participant'] + '_redlight_greenlight.csv'

# Write data frame to csv file
session_df.to_csv(output_filepath)
```

We now have a .csv data output after we finish each session of our experiment!

Exploring the Data Output

Read the data

First, we need to read the .csv file as a `pandas` data frame:

In [ ]:
```
session_df = pd.read_csv('sub-101_redlight_greenlight.csv')

session_df
```

Let's play around with this data frame to see what we can do!

See the documentation here: https://pandas.pydata.org/docs/reference/frame.html

Count the number of "go" trials.

In [ ]:
```
# access a column:
session_df["trial_type"]

# logical array:
session_df["trial_type"] == "go"

# count number of trues in logical array:
sum(session_df["trial_type"] == "go")
```

Count number of hits

In [ ]:
```
# hits logical array is the intersection of these two conditional tests:
hits_la = (session_df["resp"] == "space") & (session_df["trial_type"] == "go")

sum(hits_la)
```

In [ ]:
```
session_df
```

Use logical indexing to label trials as "hits"

In [ ]:
```
# notice how we use hits_la (the logical index of hits):
session_df.loc[hits_la, "sdt_label"] = "hit"

session_df
```

Label the other trials as "hits", "false alarms", "misses", "correct rejections"

In [ ]:

```
hits_la = (session_df["resp"] == "space") & (session_df["trial_type"] == "go")
fa_la = (session_df["resp"] == "space") & (session_df["trial_type"] == "nogo")
cr_la = (session_df["resp"].isna()) & (session_df["trial_type"] == "nogo")
miss_la = (session_df["resp"].isna()) & (session_df["trial_type"] == "go")


sum(miss_la)
```

In [ ]:

```
session_df["resp"][0:1]
```