Open in Colab

Plotting Neural Data Demo - the Allen Institute Visual Behavior 2P Dataset

Neural data can be complex and can contain many variables within one dataset. How do we make sense of this information in a meaningful way?

Today, we will plot the calcium signal from neurons recorded while mice performed a visual discrimination task.

This notebook will help us investigate data collected from the Visual Behavior 2P dataset from the Allen Brain Institute.

Learning Objectives 🙋

At the end of this notebook, you'll be able to:

- Plot calcium imaging data based on experimental conditions 💻 📈
- Examine differences in neuronal responses to familiar and novel images 🌄 🎆
- Understand differences in data format 💭 🧠
- Understand how to use common Python packages for data visualization 🐍
- Apply best practices for plotting data 📊 📓

IMPORTANT: This notebook is READ-ONLY. To edit and run this notebook, follow the steps in the next section.

💾 Before you start - Save this notebook!

When you open a new Colab notebook from a shared link (like you hopefully did for this one), you cannot save changes. So it's best to store the Colab notebook in your personal drive `"File > Save a copy in drive..."` **before** you do anything else.

The file will open in a new tab in your web browser, and it is automatically named something like: "**CopyofPlottingNeuralDataDemo.ipynb**". You can rename this to just the title of the assignment "**PlottingNeuralDataDemo.ipynb**". Make sure you do keep an informative name (like the name of the assignment) to help you be able to come back to this after you complete sections of the notebook.

Where does the notebook get saved in Google Drive?

By default, the notebook will be copied to a folder called “Colab Notebooks” at the root (home directory) of your Google Drive.

We are creating a folder for the datafiles, etc. needed for this notebook called **"NTC_materials_2024"**. You should keep your working notebook in a separate folder in case you want to restart, etc. This will make sure you don't overwrite the notebook each time you import all the files again (restart).

Quick Intro to Jupyter (Colab) Notebooks 📓

This section will introduce you using Jupyter Notebooks 📓 💻, a handy coding environment for learning as well as sharing code with others.

At the end of this notebook, you'll be able to:

- Recognize the main features of Jupyter Notebooks
- Use Jupyter Notebooks to run Python3 🐍 Code

About Jupyter Notebooks

Jupyter notebooks are a way to combine executable code, code outputs, and text into one connected file. They run in a web browser. 📶

The **'kernel'** is the thing that executes your code. It is what connects the notebook (as you see it) with the part of your computer that runs code.

Types of Cells

Jupyter Notebooks have two types of cells, a **Markdown** (like this one) and **Code**. Most of the time you won't need to run the Markdown cells, just read through them. However, when we get to a code cell, you need to tell Jupyter to run the lines of code that it contains.

Code cells will be read by the Python interpreter. In other words, the Python kernel will run whatever it recognizes as code within the cell.

In […

```
#@title Task: Run this cell
from IPython.display import HTML

alert_info = '''
<div style= "font-size: 20px"; class="alert alert-info" role="alert">
  <h4 class="alert-heading">Task</h4>
Run the cell below by clicking on the 'play arrow button' ▶ in the top left corner, or usin
</div>
'''

display(HTML('<link href="https://nbviewer.org/static/build/styles.css" rel="stylesheet">'))
display(HTML(alert_info))
```

Task

Run the cell below by clicking on the 'play arrow button' ▶ in the top left corner, or using the keys: shift + return (mac) or shift + enter (pc)

In […

```python
# In Python, anything with a "#" in front of it is code annotation,
# and is not read by the computer.
# You can run a cell (this box) by pressing shift-enter or shift-return.
# Click in this cell and then press shift and enter simultaneously.
# This print function below allows us to generate a message.
print('Nice work!')
```

Allen Institute Visual Behavior 2P dataset overview

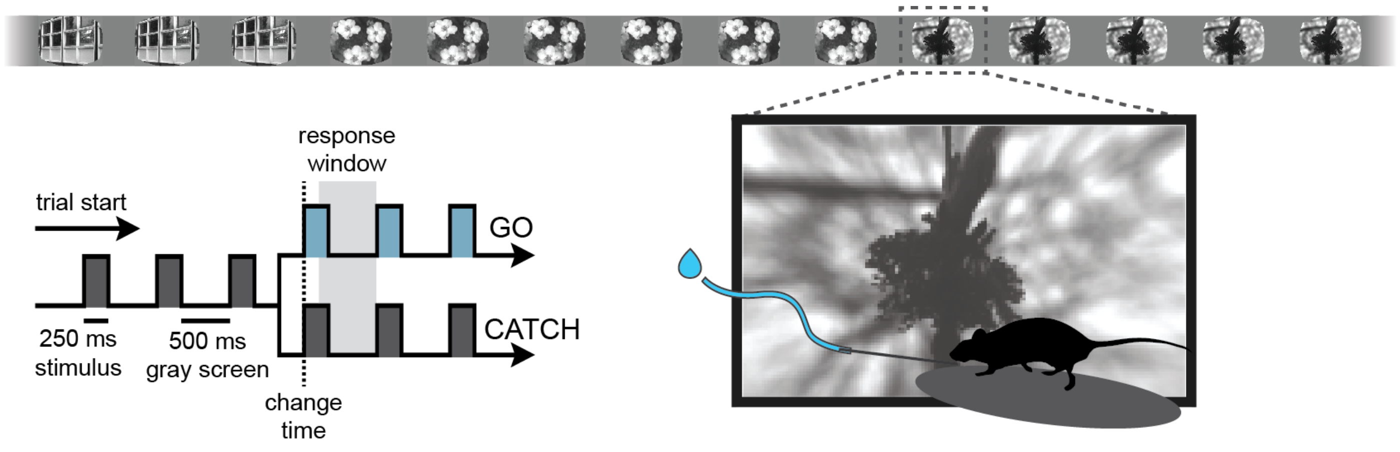
This dataset consists of neural activity measured with 2-photon calcium imaging in the visual cortex of mice performing an image change detection task. In this task, mice learn to report changes in stimulus identity by licking a spout to earn a water reward.

Dataset Notes

The entire dataset includes neural and behavioral measurements from:

- 107 mice
- 4787 behavior training sessions
- 704 *in vivo* imaging sessions
- 50,482 cortical cells

The data are openly accessible, and include information about all recorded timeseries, behavioral events, and experimental data in a standard data format: Neurodata Without Borders (NWB).

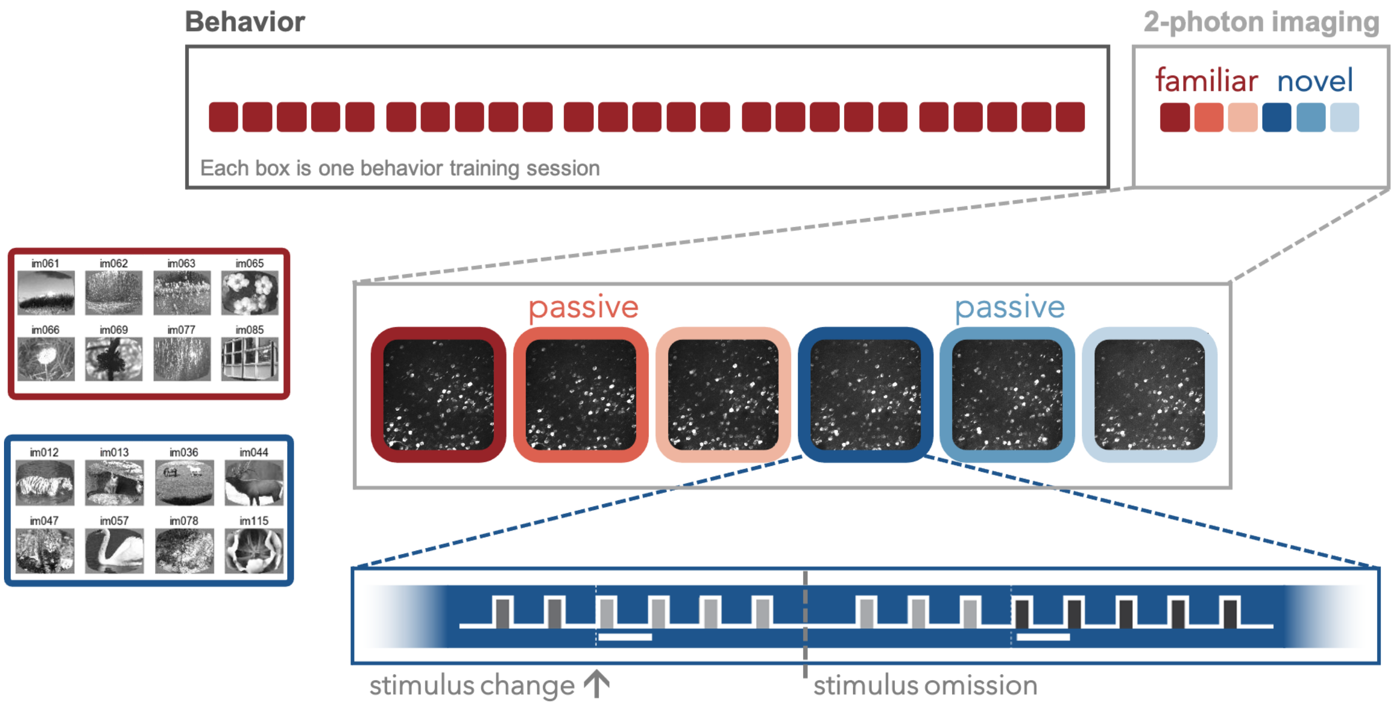


Behavioral Task

In some sessions, the mice perform the task with familiar images they have seen many times during training. In other sessions, mice perform the task with novel images.

During 2-photon imaging sessions, 5% of stimulus presentations are randomly omitted, allowing us to examine the effect of unexpected events on neural activity.

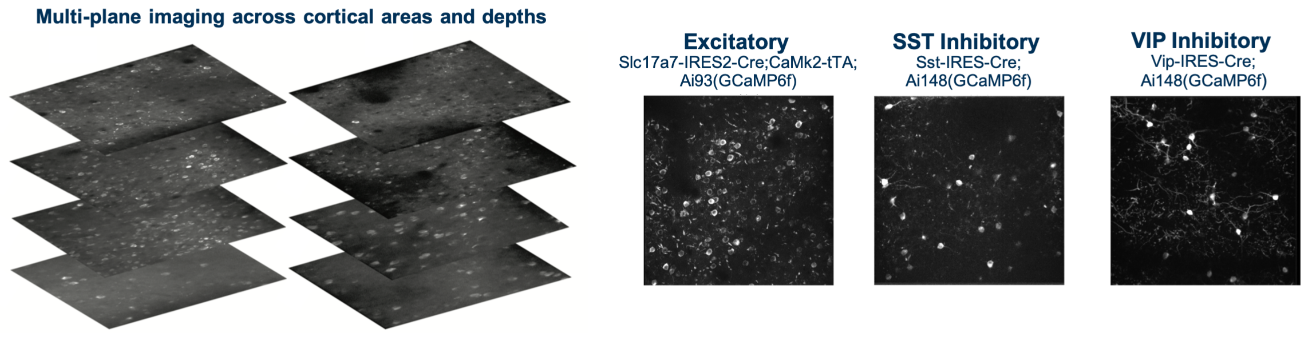
The same population of cells is imaged over multiple days with varying sensory and behavioral conditions.



Calcium Imaging

Multiple cortical areas and depths were measured concurently in each session, at a sample rate of 11Hz.

In the full dataset, data was collected from excitatory and inhibitory neural populations.


Multi-plane imaging across cortical areas and depths
Excitatory
Slc17a7-IRES2-Cre;CaMk2-tTA; Ai93(GCaMP6f)
SST Inhibitory
Sst-IRES-Cre; Ai148(GCaMP6f)
VIP Inhibitory
Vip-IRES-Cre; Ai148(GCaMP6f)

This example will focus on the activity of the two inhibitory types - VIP and SST neurons.

Check out a recent paper published in Neuron on this dataset: Behavioral strategy shapes activation of the Vip-Sst disinhibitory circuit in visual cortex

Setup and import files 🧰 🛠️

First time running this notebook (during workshop)

In […

```
#@title Task: Run these cells for setup
from IPython.display import HTML

alert_info = '''
<div style= "font-size: 20px"; class="alert alert-info" role="alert">
```

```
  <h4 class="alert-heading">Task</h4>
Run the cells step-by-step below to setup the notebook and retrieve the data files.
</div>
'''

display(HTML('<link href="https://nbviewer.org/static/build/styles.css" rel="stylesheet">'))
display(HTML(alert_info))
```

Task

Run the cells step-by-step below to setup the notebook and retrieve the data files.

In […

```
# Step 1
# Setup and add files needed to access gdrive
from google.colab import drive                                         # these lines mount your g
drive.mount('/content/gdrive', force_remount=True)
```

In […

```
# Step 2
# Make a folder to store the files
!mkdir -p '/content/gdrive/My Drive/Colab Notebooks/NTC_materials_2024/'
```

In […

```
# Step 3
# Change directory to the correct location in gdrive
import os
os.chdir('/content/gdrive/My Drive/Colab Notebooks/NTC_materials_2024/')
```

In […

```
# Step 4
# These lines clone (copy) all the files you will need from where I store the code+data for
# Second part of the code copies the files to this location and folder in your own gdrive
!git clone https://github.com/tmckim/NTC_2024 '/content/gdrive/My Drive/Colab Notebooks/NTC
```

Restarting: Run these steps (do not need to run all of the above after the first time) 🔄

In […

```
# Step 1
# Setup and add files needed to access gdrive
from google.colab import drive                                         # these lines mount your g
drive.mount('/content/gdrive', force_remount=True)
```

In […

```
# Step 2 (original step 3)
# Change directory to the correct location in gdrive
import os
os.chdir('/content/gdrive/My Drive/Colab Notebooks/')
```

In […

```
# Step 3
# remove existing dir so we can reimport the files we need from github
!rm -r NTC_materials_2024
```

In […

```
# Step 4
# make dir so we can reimport the files we need from github
!mkdir -p '/content/gdrive/My Drive/Colab Notebooks/NTC_materials_2024/'
```

In […

```
# Step 5 (original step 4)
# These lines clone (copy) all the files you will need from where I store the code+data for
# Second part of the code copies the files to this location and folder in your own gdrive
!git clone https://github.com/tmckim/NTC_2024 '/content/gdrive/My Drive/Colab Notebooks/NTC
```

Import packages needed 📫

Each time we start an analysis in Python, we must import the necessary code packages. If you're running this notebook in Colab, the cells below will install packages into your coding environment -- these are *not* installed on your computer.

Just like in many computer languages, the beauty of Python is that other people have spent much time developing useful code that they have tested robustly and wish to share with others. These are called libraries or packages. We can import popular packages into the Python/Colab environment and if the package has not already been installed on our version of Python, we can easily install the package and use it.

For this Colab notebook we will be working with four packages that typically come pre-installed on most versions of Python. They are:

- **numpy**: A library for working with numerical lists, called arrays
- **pandas**: A library for working with two dimenional lists, called dataframes
- **matplotlib**: A package for plotting, commonly used in tandem with numpy and pandas
- **seaborn**: A package for plotting, commonly used in tandem with pandas

</br> NumPy pandas </br> matplotlib

seaborn </br></br> We can import these libraries (or certain classes/functions from these libraries) into our local runtime using the following code. The nicknames I used (`pd` , `plt` , `np` , and `sns`) are pretty standard nicknames that most Python programmers use, though you could use anything.

Setup coding environment

Below, we'll `import` a common selection of packages that will help us analyze and plot our data. We'll also configure the plotting in our notebook.

- This will ensure that our coding environment has NumPy, Pandas, Matplotlib, and Seaborn installed.

In […

```
#@title Task: Import packages
from IPython.display import HTML

alert_info = '''
<div style= "font-size: 20px"; class="alert alert-info" role="alert">
  <h4 class="alert-heading">Task</h4>
 Make sure our needed packages are installed in our coding environment by running the code
 </div>
'''

display(HTML('<link href="https://nbviewer.org/static/build/styles.css" rel="stylesheet">'))
display(HTML(alert_info))
```

Task

Make sure our needed packages are installed in our coding environment by running the code below. For example, when you see `np` in our script, it's actually calling scripts from the `numpy` package.

In […

```
# If you restart colab, make sure you run this cell again!
# Import our plotting package from matplotlib
import matplotlib.pyplot as plt

# Specify that all plots will happen inline
%matplotlib inline

# Import pandas for working with databases
import pandas as pd

# Import seaborn for plotting adjustments with matplotlib
import seaborn as sns

# Import numpy
import numpy as np

# Print statement to confirm
print('Packages imported!')
```

Import the data file into a `pandas` dataframe ⏳

The data was directly downloaded from the Allen Institute website (parquet is just another file format). It was copied to your google drive folder upon import at the beginning. This way we can easily pull in the data we want to work with in our notebook.

In […

```
#@title Task: Import the data
from IPython.display import HTML

alert_info = '''
<div style= "font-size: 20px"; class="alert alert-info" role="alert">
  <h4 class="alert-heading">Task</h4>
Run the cell below to import the data using the <code>pandas</code> package (which we short
</div>
'''

display(HTML('<link href="https://nbviewer.org/static/build/styles.css" rel="stylesheet">'))
display(HTML(alert_info))
```

Task

Run the cell below to import the data using the `pandas` package (which we shortened to `pd` when we imported it).

In […
```
# Import data
filename = "allen_visual_behavior_2p_change_detection_familiar_novel_image_sets.parquet"
data = pd.read_parquet(filename)
```

Viewing the Data (`pandas` dataframe) 👀

Notice that when you write an assignment statement like the one above, python doesn't automatically print anything out or show you the dataset in this case.

Each row contains all data for a given cell on a given trial

In […
```
#@title Task: Show me the data
from IPython.display import HTML

alert_info = '''
<div style= "font-size: 20px"; class="alert alert-info" role="alert">
  <h4 class="alert-heading">Task</h4>
To show the data, run the cell below where we have typed out the name of the variable that
</div>
'''

display(HTML('<link href="https://nbviewer.org/static/build/styles.css" rel="stylesheet">'))
display(HTML(alert_info))
```

Task

To show the data, run the cell below where we have typed out the name of the variable that contains our dataset.

In […
```
# Select a random sample of 5 rows to display
data.sample(5)
```

In […
```
# Let's check out all of our column names
list(data)

# np.sort(list(data)) # if you want them alphabetical instead of the order they appear
```

In […
```
# Get info about our data - rows by col #s
data.shape
```

🧠 Available data includes:

- The cell `trace` aligned to stimulus (or omission) onset in a [-1.25, 1.5] second window around onset time
  - Cell traces are in units of delta F/F, the change in fluorescence relative to baseline
- The `trace_timestamps` for each trial, aligned to stimulus or omission onset
- The `mean_response` on a given trial in a 500ms window after stimulus onset
- The `baseline_response` on a given trial in a 500ms window before stimulus onset
- The `image_name` for each trial. Trials where the stimulus was omitted have `image_name` = `omitted`
- The `image_index` indicates which of the 8 eight images for the session was presented. Should also correspond with `image_name`.
- The `mean_running_speed` in a 500ms window after stimulus onset
- The `mean_pupil_area` in a 500ms window after stimulus onset
- The `response_latency` when the mouse licked after stimulus onset
- Whether or not the trial was `rewarded`
- Whether or not the trial `is_change`
- Whether or not the trial was `omitted`
- Info about `age_in_days` of the animal
- Biological `sex` of the animal

Cell and session level metadata includes:

- The `stimulus_presentations_id` indicating the trial number within the session
- The `cell_specimen_id` which is the unique identifier for each cell (note that a cell can be imaged in multiple sessions; if that's the case, the same cell_specimen_id appears in multiple sessions)
- The `cre_line` indicating the cell type
  - `Sst-IRES-Cre` labels SST inhibitory cells
  - `Vip-IRES-Cre` labels VIP inhibitory cells
  - `Slc17a7-IRES-Cre` labels excitatory cells (these are excluded for now)
- The `full_genotype` indicates all information about genetic background of the animal
- The `reporter_line` indicates reporter genetic background of animal
- The `driver_line` indicates driver genetic background of animal
- The `indicator` details the calcium flourescence reporter
- The `imaging_depth` indicating the cortical depth where the cell was located
- The `targeted_structure` indicating the cortical area the cell was from
- The `session_type` indicating the session order and image set
- The `exposure_level` which tells you whether the image set was familiar or novel
- The `session_number` corresponds to which familiar and novel imaging context was used
- The `mouse_id` indicating which mouse the cell came from
- The `ophys_session_id` indicating the recording day for that trial
- The `ophys_experiment_id` indicating which imaging plane within the session that the cell came from
- The `ophys_container_id` which links the same imaging plane recorded across multiple sessions. Cells that are imaged across multiple sessions will have the same `cell_specimen_id`
- The `behavior_session_id` which indicates the behavior session code for each animal

Use `unique` to find information in our columns of interest 🔬

One way to start examining our data is to see what values appear in the columns. Luckily, we don't have to do this by eye 👀 and scroll the whole way through the table (although I do recommend doing this to get a feel for the data!).

We can use the function `unique` in `pandas` to help us out. It will return the unique values from the column we specify based on the order of appearance (it does not sort them in any way).

Helpful info can be found in the pandas documentation and elsewhere.

Below, we will first call the name of our dataframe (`data`) then reference the column name (`.<insert_column_name_here>`) and finally use `.unique()` to figure out the possible values in our column

---

In […

```
#@title Task: Find column values using <code>unique</code>
from IPython.display import HTML

alert_info = '''
<div style= "font-size: 20px"; class="alert alert-info" role="alert">
  <h4 class="alert-heading">Task</h4>
Which column do you think has the info about image type (familiar vs novel)? <br>
Fill this in the code below where it says <code> insert_column_name_here </code> <br><br>
<i>Hint</i>: Look back at the dataframe (table) and check out what values are in each colum
</div>
'''

display(HTML('<link href="https://nbviewer.org/static/build/styles.css" rel="stylesheet">'))
display(HTML(alert_info))
```

Task

Which column do you think has the info about image type (familiar vs novel)?
Fill this in the code below where it says `insert_column_name_here`

Hint: Look back at the dataframe (table) and check out what values are in each column.

In […

```
##--- Edit this

# Image types
print('exposure_levels:', data.<insert_column_name_here>.unique())
```

In […

```
#@title Example Solution
# Image types
```

```
print('exposure_levels:', data.exposure_level.unique())
```

In […
```
# Same or different image
print('stimulus presentations can be changes:', data.is_change.unique())
```

In […
```
# Omission trials
print('stimulus presentations can be omitted:', data.omitted.unique())
```

In […
```
# Cre lines
print('cre lines (cell types) included in this dataset are:', data.cre_line.unique())
```

In […
```
# Mouse count
print('there are', len(data.mouse_id.unique()), 'mice in this dataset')
```

In […
```
# Cell count
print('there are', len(data.cell_specimen_id.unique()), 'cells in this dataset')
```

In […
```
# Brain region(s) info
print('the brain region(s) in this dataset are:', data.targeted_structure.unique())
```

In […
```
#@title Additional Examples of Variable Info
# Session count
print('there are', len(data.ophys_session_id.unique()), 'sessions in this dataset')

# Female count
females = data[(data.sex == 'F')]
print('there are', len(females.mouse_id.unique()), 'females in this dataset')

# Male count
males = data[(data.sex == 'M')]
print('there are', len(males.mouse_id.unique()), 'males in this dataset')
```

1️⃣ How are VIP and SST cells affected by stimulus novelty?

Plot the population average change response for familiar and novel images for each cre line

1.1: Get trials where the image identity changed, for SST and VIP cells 🔎

In […
```
#@title Task: Subset the dataframe
from IPython.display import HTML

alert_info = '''
<div style= "font-size: 20px"; class="alert alert-info" role="alert">
  <h4 class="alert-heading">Task</h4>
Run the cell below. <br>
The code will find the data within the dataframe based on the following 2 conditions: <br>
1. rows of data for each cell type <br>
AND <br>
```

```
2. when the image changed
</div>
'''

display(HTML('<link href="https://nbviewer.org/static/build/styles.css" rel="stylesheet">'))
display(HTML(alert_info))
```

Task

Run the cell below.

The code will find the data within the dataframe based on the following 2 conditions:

1. rows of data for each cell type

AND

2. when the image changed

In […

```
# Run these cells
sst_data = data[(data.cre_line == 'Sst-IRES-Cre')&(data.is_change == True)]
vip_data = data[(data.cre_line == 'Vip-IRES-Cre')&(data.is_change == True)]
```

In […

```
sst_data.shape
```

In […

```
vip_data.shape
```

1.2: Plot the population average change response of **SST** cells for familiar and novel images 📈

First, let's think about what data we need from the table. In this case, we want to plot the change in calcium flourescence (y-axis) over time (x-axis).

The two columns with this info are:

`trace`
`trace_timestamps`

Let's take a look at this data before we plot.

In […

```
#@title Task: Review the data
from IPython.display import HTML

alert_info = '''
<div style= "font-size: 20px"; class="alert alert-info" role="alert">
  <h4 class="alert-heading">Task</h4>
Run the cells below to take a look at the data before we plot. <br>
What do you notice?
</div>
'''
```

```
display(HTML('<link href="https://nbviewer.org/static/build/styles.css" rel="stylesheet">'))
display(HTML(alert_info))
```

Task

Run the cells below to take a look at the data before we plot.

What do you notice?

In […
```
# Run these cells
# trace data
trace_sst = sst_data.trace
trace_sst
```

In […
```
# timestamps- aligned to stimulus onset or omission
timestamps_sst = sst_data.trace_timestamps
timestamps_sst
```

^ Notice that all the timestamps are the same- they should be! Everything was aligned (in time) for us, so we only need this set of values once for our x-axis ⏰

Now that we know what these look like, we need to figure out how to incorporate them into a plot. We will first use `matplotlib`, which we imported as `plt`.
We reference the column from the dataset that we want (`trace_timestamps`), and use `.values` from `pandas` to show the values in the first one

In […
```
# Run these cells
timestamps = sst_data.trace_timestamps.values[0]  # show the first set of timestamps
timestamps
```

^Another new thing here! Why did we use 0 to reference the **first** one?
Python uses zero indexing - the first element is element 0, the second element is element 1, etc. This is something that takes getting used to. We will cover this soon in the next few weeks of class.

In […
```
# Now we want to plot the y-values for each image condition- familiar and novel
fam_sst = sst_data[sst_data.exposure_level=='familiar'].trace.values
nov_sst = sst_data[sst_data.exposure_level=='novel'].trace.values
```

In […
```
fam_sst
```

In […
```
#@title Task: Our first plot
from IPython.display import HTML

alert_info = '''
<div style= "font-size: 20px"; class="alert alert-info" role="alert">
  <h4 class="alert-heading">Task</h4>
Run the cell below and review the code while the plot appears. <br>
Once the plot appears, what do you see?
</div>
'''
```

```
display(HTML('<link href="https://nbviewer.org/static/build/styles.css" rel="stylesheet">'))
display(HTML(alert_info))
```

Task

Run the cell below and review the code while the plot appears.

Once the plot appears, what do you see?

In […

```
# Plotting population average (all cells) SST response for familiar and novel images
plt.plot(timestamps, np.mean(fam_sst), label='familiar')
plt.plot(timestamps, np.mean(nov_sst), label='novel')


# Plot aesthetics
plt.title('SST population average')
plt.xlabel('Time after change (sec)')
plt.ylabel('dF/F')
plt.legend()
plt.show()
```

In […

```
#@title Advanced: Using a for loop to create the same plot above
# Plotting code will generate the same plot as above, just more advanced using loop
timestamps = sst_data.trace_timestamps.values[0]

# Here we can loop through the unique levels of exposure (familiar and novel) to plot each
for exposure_level in sst_data.exposure_level.unique():
  traces = sst_data[sst_data.exposure_level==exposure_level].trace.values
  plt.plot(timestamps, np.mean(traces), label=exposure_level)

# Plot aesthetics
plt.title('SST population average')
plt.xlabel('Time after change (sec)')
plt.ylabel('dF/F')
plt.legend()
plt.show()
```

1.3: Plot the population average change response of **VIP** cells for familiar and novel images 📉

Similar to above, but now with VIP data instead of SST

In […

```
# Run these cells

# We reference the column from the dataset that we want (trace_timestamps), and use .values
timestamps_vip = vip_data.trace_timestamps.values[0]      # Notice that all the timestamps
timestamps_vip
```

In […

```
# Now we want to plot the y-values for each image condition- familiar and novel
fam_vip = vip_data[vip_data.exposure_level=='familiar'].trace.values
nov_vip = vip_data[vip_data.exposure_level=='novel'].trace.values
```

In […

```
# Plotting population average (all cells) VIP response for familiar and novel images
plt.plot(timestamps_vip, np.mean(fam_vip), label='familiar')
plt.plot(timestamps_vip, np.mean(nov_vip), label='novel')


# Plot aesthetics
plt.title('VIP population average')
plt.xlabel('Time after change (sec)')
plt.ylabel('dF/F')
plt.legend()
plt.show()
```

In […

```
#@title Advanced: Using a for loop to create the same plot above
# Advanced with a loop for the conditions (same plot as above though)

timestamps = vip_data.trace_timestamps.values[0]

# Here we can loop through the unique levels of exposure (familiar and novel) to plot each
for exposure_level in vip_data.exposure_level.unique():
  traces = vip_data[vip_data.exposure_level==exposure_level].trace.values
  plt.plot(timestamps, np.mean(traces), label=exposure_level)

# Plot aesthetics
plt.title('VIP population average')
plt.xlabel('time after change (sec)')
plt.ylabel('dF/F')
plt.legend()
plt.show()
```

2️⃣ How do we find cells that were imaged across multiple sessions? How do single cells change depending on the image set?

In […

```
#@title Task: Run the complete code
from IPython.display import HTML

alert_info = '''
<div style= "font-size: 20px"; class="alert alert-info" role="alert">
  <h4 class="alert-heading">Task</h4>
Run the cell below and review the output. <br>
In the following cells, we will breakdown the steps we went through individually.
</div>
'''

display(HTML('<link href="https://nbviewer.org/static/build/styles.css" rel="stylesheet">'))
display(HTML(alert_info))
```

Task

Run the cell below and review the output.
In the following cells, we will breakdown the steps we went through individually.

In […
```
# which cells are in more than one session? make a list
cells_in_multiple_sessions = []

for cell_specimen_id in vip_data.cell_specimen_id.unique():
  if len(vip_data[vip_data.cell_specimen_id == cell_specimen_id].ophys_session_id.unique())
    cells_in_multiple_sessions.append(cell_specimen_id)

print(cells_in_multiple_sessions)
```

In […
```
# 1 - we start by getting a list of the unique ids
vip_data.cell_specimen_id.unique()
```

In […
```
# 2 - but who wants to count- let's use len to tell us how many cells there are
len(cells_in_multiple_sessions)
```

We pick a test cell id from the list. We will just pick the first one (note it is not in the multiple session list for reference)

In […
```
# 3 - let's pick a test cell
cell_specimen_id = 1086492391
```

In […
```
# 4 - let's see what appears first before evaluating the length and >1
# Notice this is just part of the code in the line that starts with 'if'
vip_data[vip_data.cell_specimen_id == cell_specimen_id].ophys_session_id.unique()
```

In […
```
# 5 - Now put it all together - does the output make sense?
# Find the data for the single cell
# Find the unique entries of ophys_session_id
# If the length of this is great than 1, it was a repeat cell
len(vip_data[vip_data.cell_specimen_id == cell_specimen_id].ophys_session_id.unique()) > 1
```

We pick a test cell id from the multiple session list. We will just pick the first one

In […
```
# 6 - cell id from the multiple session list
cell_specimen_id = 1086495458
```

In […
```
# 7 - let's see what appears first before evaluating the length and >1
# Notice this is just part of the code in the line that starts with 'if'
vip_data[vip_data.cell_specimen_id == cell_specimen_id].ophys_session_id.unique()
```

In […
```
# 8 - Now put it all together- does the output make sense?
len(vip_data[vip_data.cell_specimen_id == cell_specimen_id].ophys_session_id.unique()) > 1
```

2.1: Plot and review the data from example cells in the multiple sessions list

In […
```
# this one looks like the population average
example_cell_specimen_id = cells_in_multiple_sessions[15]
```

```
# Select the data for our cell of interest
cell_data = vip_data[vip_data.cell_specimen_id == example_cell_specimen_id]
timestamps = cell_data.trace_timestamps.values[0]

# Use a loop to plot for both exposure levels (familiar and novel)
for exposure_level in cell_data.exposure_level.unique():
  mean_trace = cell_data[cell_data.exposure_level == exposure_level].trace.mean()
  plt.plot(timestamps, mean_trace, label=exposure_level)

# Plot aesthetics
plt.title(f'cell_specimen_id:{example_cell_specimen_id}') # new - formatted string literal
plt.xlabel('time after change (sec)')
plt.ylabel('dF/F')
plt.legend()
plt.show()
```

In […
```
# this one does *not* look like the population average
example_cell_specimen_id = cells_in_multiple_sessions[0]

# Select the data for our cell of interest
cell_data = vip_data[vip_data.cell_specimen_id==example_cell_specimen_id]
timestamps = cell_data.trace_timestamps.values[0]

# Use a loop to plot for both exposure levels (familiar and novel)
for exposure_level in cell_data.exposure_level.unique():
  mean_trace = cell_data[cell_data.exposure_level == exposure_level].trace.mean()
  plt.plot(timestamps, mean_trace, label=exposure_level)

# Plot aesthetics
plt.title(f'cell_specimen_id:{example_cell_specimen_id}')
plt.xlabel('time after change (sec)')
plt.ylabel('dF/F')
plt.legend()
plt.show()
```

2.2 What about trial to trial variability? How does the response of a single cell vary across a session?

In […
```
# Pre-selected id - feel free to choose another
example_cell_specimen_id = vip_data[vip_data.exposure_level=='novel'].cell_specimen_id.uniq
example_cell_specimen_id
```

In […
```
# Get all the data from the larger table for this cell id we chose above
cell_data = vip_data[vip_data.cell_specimen_id == example_cell_specimen_id]
cell_data
```

In […
```
# What images were shown? Need to know how many in the list to choose one below
cell_data.image_name.unique()
```

In […
```
# pick a cell from a novel image session
example_cell_specimen_id = vip_data[vip_data.exposure_level=='novel'].cell_specimen_id.uniq

# Select the data for our cell of interest
cell_data = vip_data[vip_data.cell_specimen_id == example_cell_specimen_id]
cell_data = cell_data[(cell_data.image_name == cell_data.image_name.unique()[2])] # pick on

# This controls the color options
offset = 1 / len(cell_data.stimulus_presentations_id.unique())
color = [0, 0, 0]

# Loop for plotting all of the data for this cell
for i, stimulus_presentations_id in enumerate(cell_data.stimulus_presentations_id.unique())
  trial_data = cell_data[cell_data.stimulus_presentations_id == stimulus_presentations_id]
  timestamps = trial_data.trace_timestamps.values[0]
  trace = trial_data.trace.values[0]
  plt.plot(timestamps, trace, color = color)
  color = [color[0] + offset, color[1] + offset, color[2] + offset]

# Plot aesthetics
plt.title(f'cell_specimen_id:{example_cell_specimen_id}')
plt.xlabel('time after change (sec)')
plt.ylabel('dF/F')
plt.show()
```

3️⃣ How do SST and VIP cells respond when stimuli are omitted?

Plot the population average response to stimulus omission

3.1: Get trials where the stimulus was omitted, for SST and VIP cells

In […

```
#@title Task: Compare this code to the previous
from IPython.display import HTML

alert_info = '''
<div style= "font-size: 20px"; class="alert alert-info" role="alert">
  <h4 class="alert-heading">Task</h4>
How does this code compare to what we used previously to filter/subset our data? <br>
<i>Hint:</i> There is one change!
 </div>
'''

display(HTML('<link href="https://nbviewer.org/static/build/styles.css" rel="stylesheet">'))
display(HTML(alert_info))
```

Task

How does this code compare to what we used previously to filter/subset our data?
Hint: There is one change!

In […
```
sst_data_omit = data[(data.cre_line == 'Sst-IRES-Cre')&(data.omitted == True)]
vip_data_omit = data[(data.cre_line == 'Vip-IRES-Cre')&(data.omitted == True)]
```

3.2: Plot the population average omission response of **SST** cells for familiar and novel images

In […
```
#@title Task: Use the code above to write your own here
from IPython.display import HTML

alert_info = '''
<div style= "font-size: 20px"; class="alert alert-info" role="alert">
  <h4 class="alert-heading">Task</h4>
Can you use the previous example we visualized (for <b>change</b> responses) to write your
Feel free to copy and paste - just make sure to edit it accordingly.
 </div>
'''

display(HTML('<link href="https://nbviewer.org/static/build/styles.css" rel="stylesheet">')
display(HTML(alert_info))
```

Task

Can you use the previous example we visualized (for **change** responses) to write your own here?

Feel free to copy and paste - just make sure to edit it accordingly.

In […
```
# Your code here
```

In […
```
#@title Example Solution
# Get the x-axis values for the time
timestamps = sst_data_omit.trace_timestamps.values[0] # trace timestamps are relative to st

# Get the y-axis values of dF/F to plot for each exposure level (novel and familiar)
for exposure_level in sst_data_omit.exposure_level.unique():
  traces = sst_data_omit[sst_data_omit.exposure_level == exposure_level].trace.values
  plt.plot(timestamps, np.mean(traces), label=exposure_level)

# Plot aesthetics
plt.title('SST population average')
plt.xlabel('time after omission (sec)')
plt.ylabel('dF/F')
plt.legend()
plt.show()
```

3.3: Plot population average omission response of **VIP** cells for familiar and novel images

In […
```
# Your code here
```

In […
```
#@title Example Solution
# Get the x-axis values for the time
```

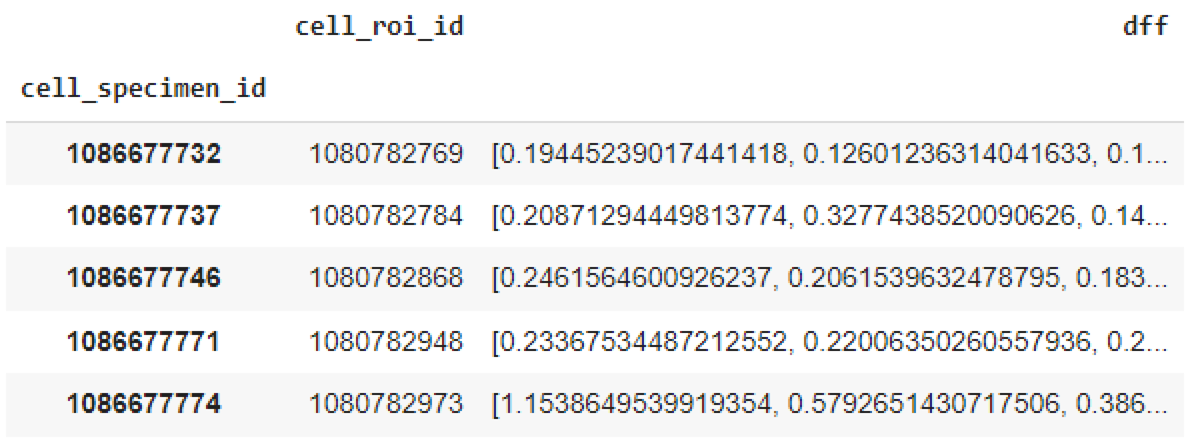
```python
timestamps = vip_data_omit.trace_timestamps.values[0]

# Get the y-axis values of dF/F to plot for each exposure level (novela and familiar)
for exposure_level in vip_data_omit.exposure_level.unique():
  traces = vip_data_omit[vip_data_omit.exposure_level == exposure_level].trace.values
  plt.plot(timestamps, np.mean(traces), label=exposure_level)

# Plot aesthetics
plt.title('VIP population average')
plt.xlabel('time after omission (sec)')
plt.ylabel('dF/F')
plt.legend()
plt.show()
```

❓ Additional questions you could ask about and plot from this dataset:

1. Does the omission response correlate with behavior?
2. How do the dynamics of image and omission evoked activity change over time during the novel image session?

4️⃣ Comparing data format: plotting with long format data 🔀

Note on data format: This dataset is in wide format, where each row would represent a different `cell_id` and all data for that cell would be contained in the columns of a single row. See a smaller image of the table we've been working with below, focusing on the `dff` column (which has multiple values in it) in the example image below for comparison:



	cell_roi_id	dff
cell_specimen_id		
1086677732	1080782769	[0.19445239017441418, 0.12601236314041633, 0.1...
1086677737	1080782784	[0.20871294449813774, 0.3277438520090626, 0.14...
1086677746	1080782868	[0.2461564600926237, 0.2061539632478795, 0.183...
1086677771	1080782948	[0.23367534487212552, 0.22006350260557936, 0.2...
1086677774	1080782973	[1.1538649539919354, 0.5792651430717506, 0.386...

Tidy Data 🧹

However, sometimes datasets can be easier to work with in long-format, or `tidy` data.
To demonstrate this, let's open another file for a **single mouse** 🐭 that has been reformatted already.

Here is another example of loading data - it can also easily be loaded from `.csv` files using `pandas`.

In […
```python
# Load the data file - SST_data.csv
single_data = pd.read_csv('SST_data.csv')
```

```
# Show the data for review
single_data
```

You can see that in this dataframe, `trial_id` is repeated across multiple rows of the dataset. So, a single `cell_id` is repeated across the rows. This is most similar to the data we will work with in this course.

Advantages to tidy data:

1. Consistency: enables learning tools to work with your data, such as `seaborn` (data viz) that is designed to work with tidy data
2. Efficiency: Placing variables in columns makes using pandas operations most efficient

---

Columns in the dataset

They are similar and a smaller subset from the ones above, with some names just slightly adjusted:

- `dF_F` calcium imaging signal (baseline corrected, normalized fluorescence)
- `time_from_stim` is the timepoint of each row of data, aligned to an image presentation. Onset = time zero, and the times here span a (-1.25, 1.5) sec window
- `cell_id` id number for the cell
- `exposure` whether the image for a trial was familiar or novel
- `trial_id` each image presentation is a separate trial
- `omitted` whether a trial had an omitted image
- `pupil_area` measured 500ms after stimulus presntation
- `mean_response` average dF/F over the 500ms following image presentation

Now, let's take a closer look at the data and do some preliminary exploration to understand what we are working with.

In […
```
#@title Task: How many cells are in the dataset?
from IPython.display import HTML

alert_info = '''
<div style= "font-size: 20px"; class="alert alert-info" role="alert">
  <h4 class="alert-heading">Task</h4>
Use <code>len</code> and <code>unique</code> to find the number of cells (<code>cell_id</cc
<i>Hint:</i> Look back where we previously used <code>unique</code>
</div>
'''

display(HTML('<link href="https://nbviewer.org/static/build/styles.css" rel="stylesheet">'))
display(HTML(alert_info))
```

Task

Use `len` and `unique` to find the number of cells (`cell_id` column) in the dataset.

Hint: Look back where we previously used `unique`

In […
```
##--- Edit this
# Find the unique entries from the cell_id column of the dataset
# Review: index the dataframe using .COLUMN_NAME to refer to the column that you want
```

In […
```
#@title Example Solution
# Find the unique entries from the cell_id column of the dataset
# Review: index the dataframe using .COLUMN_NAME to refer to the column that you want
len(single_data.cell_id.unique())
```

Let's look at a single `trial_id` and single `cell_id` combination.

In […
```
# select which trial id and cell id from the dataframe
singlecell_trial_data = single_data[(single_data.trial_id == 24) & (single_data.cell_id ==

# Display the data
singlecell_trial_data
```

4.1 Plotting with the Seaborn package

Part 1: Plotting separate conditions with Seaborn

Recall that we imported `seaborn` as `sns`. First, we will use a function called `sns.lineplot`. We will use this to plot our calcium signal based on familiar and novel trials. `Seaborn` allows us to do this by using `hue` in combination with the name of our variable of interest.

Read what this does and refer to the examples in the `sns.lineplot` documentation.

4.1.1 Lineplot

Let's plot the data like we did above, but remember this is now trials averaged for just a single mouse.

Note: This may take a few seconds, compared to how long it took for the previous plotting code to run above. There are some statistics being computed in the background, so it takes a bit longer (~10 secs) for the plot to be displayed.

In […
```
# Lineplot code
sns.lineplot(data = single_data,
             x = 'time_from_stim',
             y = 'dF_F',
             hue = 'exposure');
```

In […
```
#@title Task: Compare this plot to output from <code>matplotlib</code>
from IPython.display import HTML

alert_info = '''
<div style= "font-size: 20px"; class="alert alert-info" role="alert">
  <h4 class="alert-heading">Task</h4>
Which specific aspects of the plot were added automatically?
</div>
```

```
'''

display(HTML('<link href="https://nbviewer.org/static/build/styles.css" rel="stylesheet">'))
display(HTML(alert_info))
```

Task

Which specific aspects of the plot were added automatically?

4.1.2 Relplot

Now, let's plot again, but using a different function within `seaborn` to compare.
Review the documentation on `relplot`.
Read the first section **Visualizaing statistical relationships**,
then scroll below (you can skip the sections leading up to the next one ->) and read **Emphasizing continuity with line plots**.

Let's use the code below to plot again.

In […

```
# Use relplot this time
sns.relplot(data = single_data,
            x='time_from_stim',
            y='dF_F',
            kind = 'line',
            hue = 'exposure');
```

Does this plot look similar to the first one? It should be exactly the same (note: the axes might appear visually stretched, but the data hasn't changed!).

This demonstrates that there may be multiple ways to use functions within packages to produce the same plots and data visualizations!

4.1.3 Changing Labels

We can also change the names of the labels on our plots. There are a few ways to do this, but the easiest is to interface with `matplotlib` like we did previously.

In […

```
#@title Task: Review and edit plot code
from IPython.display import HTML

alert_info = '''
<div style= "font-size: 20px"; class="alert alert-info" role="alert">
  <h4 class="alert-heading">Task</h4>
Add labels by replacing <code>insert_text</code>. Feel free to edit other settings and prac
We are again plotting calcium signal (<code>dF_F</code>) over time (<code>time_from_stim</c
</div>
'''

display(HTML('<link href="https://nbviewer.org/static/build/styles.css" rel="stylesheet">'))
display(HTML(alert_info))
```

Task

Add labels by replacing `insert_text`. Feel free to edit other settings and practice options for using `set_style` and `palette`.
We are again plotting calcium signal (`dF_F`) over time (`time_from_stim`) by familiar/novel (`exposure`) conditions.

In […

```
##--- Edit this
# Plot with adjusted labels
# Another style option
sns.set_style('ticks')

# Plot with palette and adjust labels/title
sns.lineplot(data = single_data,
             x = 'time_from_stim',
             y = 'dF_F',
             hue = 'exposure',
             palette = 'flare')

plt.xlabel(<insert_text>)   # edit
plt.ylabel(<insert_text>)   # edit
plt.title(<insert_text>)    # edit
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0);   # you don't need
```

In […

```
#@title Example Solution
# Plot with adjusted labels
# Another style option
sns.set_style('ticks')

# Plot with palette and adjust labels/title
sns.lineplot(data = single_data, x = 'time_from_stim', y = 'dF_F', hue = 'exposure', palett

plt.xlabel('Time from image presentation')
plt.ylabel('Calcium Activity')
plt.title('Calcium imaging over time')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0);   # you don't need
```

4.1.4 Adding multiple condition combinations

We can add another condition (`omitted`) to the plot using the style option.

In […

```
# Multiple conditions
# Set the size
plt.figure(figsize=(8,4))

# Plot the data
sns.lineplot(data = single_data,
             x = 'time_from_stim',
             y = 'dF_F',
             hue = 'exposure',
             style = 'omitted',
             palette = 'mako')
```

```
plt.xlabel('Time from stim')
plt.ylabel('dF/F')
plt.title('All the combinations')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0);  # you don't need
```

Part 2: Figure aesthetics 🌈 💄

With `seaborn`, there are multiple options for controlling aesthetics. Review the link to see how we will start use `sns.set_style` and `sns.set_context`.

Let's also look at the data in a different way, using another variable from the dataset: `mean_response`.

In […

```
#@title Task: Looking at the data another way
from IPython.display import HTML

alert_info = '''
<div style= "font-size: 20px"; class="alert alert-info" role="alert">
  <h4 class="alert-heading">Task</h4>
What does <code>mean_response</code> measure, and how does it relate to the previous variab
'''

display(HTML('<link href="https://nbviewer.org/static/build/styles.css" rel="stylesheet">'))
display(HTML(alert_info))
```

Task

What does `mean_response` measure, and how does it relate to the previous variable `dF_F` that we were using?

We will also introduce another plot type `catplot` to display individual data points. Review the documentation here.

In […

```
# Setup and plot data
# Set style and context
sns.set_style("whitegrid")
sns.set_context('talk')

# Subset of the data (faster display, but entire dataset can be used)
data_sample = single_data.sample(1000)

# Categorical plot
sns.catplot(data = data_sample,
            x = 'exposure',
            y = 'mean_response',
            hue = 'exposure',
            palette = 'mako');
```

Extra Practice

1) Can you figure out how to make a histogram of pupil area in `Seaborn`? Use the subset of data below.

2) Create a second plot, but split the histogram by `exposure` to plot

In […
```
# Subset of data to use for this exercise
data_sample = single_data.sample(1000)
```

In […
```
## Edit this--- Plot 1
# Setup and plot data
# Set style and context
sns.set_style("dark")
sns.set_context('paper')


# Categorical plot
sns.histplot(data = data_sample,
             x = ...);
```

In […
```
## Edit this--- Plot 2

# Setup and plot data
# Set style and context
sns.set_style("dark")
sns.set_context('notebook')


# Categorical plot
sns.histplot(data = data_sample,
             x = ...,
             hue = ...,
             palette = 'mako');
```

❓❗ Errors & Troubleshooting

If the data import doesn't work for you, another option is to download the files and upload:

1. datafile1
2. datafile2

to download the data file locally to your computer. You can then navigate to the 📁 icon on the left and drag and drop the files there for access.

🌟 Just for fun 🌀

In […
```
# Run this for a fun message + image
from IPython.display import HTML
print('Great work today!')
HTML('<img src="https://media.giphy.com/media/jkvmzOg3LtpF6/giphy.gif">')
```

Saving 💾

Remember to save your notebook before closing. Choose **Save** (and make sure you've already saved a copy in your drive) from the **File** menu.

Technical notes & credits 🔩👏👦

Much more information can be found in the Allen Brain Institute whitepaper as well as in their documentation.

This notebook was developed from the Allen Institute Notebooks, Neuromatch Academy, and the Columbia-Neuropythonistas repository.