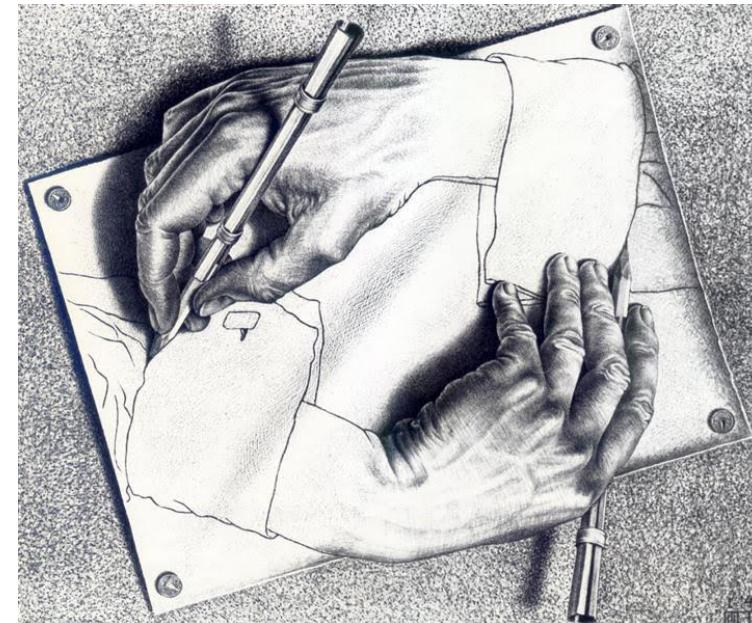


Why & how to make reproducible figures (in Python)

Thijs van der Plas,
lab meeting, 15 June 2022

vdplasthijs@gmail.com



Goal

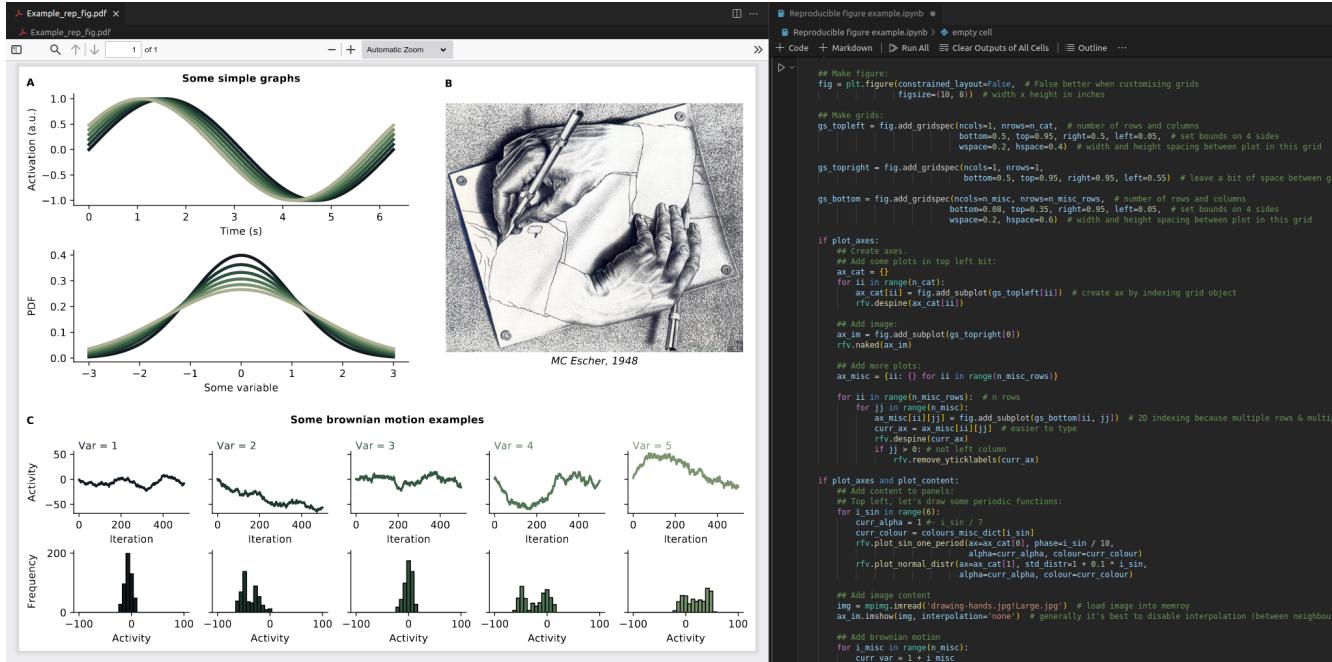
- Goal: provide ideas & resources, and make you excited to make reproducible figures!
- Not much code

Outline

- What do I mean by ‘reproducible figure’ (RF)?
- Why should you consider creating (only) RFs?
- How to make RFs in Python?
- Some more tips & tricks

What do I mean by ‘reproducible figure’?

- A (multi-panel) figure that is produced **completely** and **finally** by code, and can therefore be reproduced (by anyone).



What do I mean by ‘reproducible figure’?

- A (multi-panel) figure that is produced **completely** and **finally** by code, and can therefore be reproduced (by anyone).
- Alternative: make individual panels & compile in illustrator/inkscape + potentially add visual elements. => ‘manually-compiled figures (MCF)’
- *Why should you consider making RFs?*

Why RFs?

- **Time-efficient:** Avoid repetition

RFs avoid repetition.

Repetition across:

- Figure iterations
- Panels (supp figs)
- Projects/papers
- People

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

		HOW OFTEN YOU DO THE TASK					
		50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
1 SECOND		1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
5 SECONDS		5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
30 SECONDS		4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
HOW MUCH TIME YOU SHAVE OFF	1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
	5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
	30 MINUTES		6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS
	1 HOUR		10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS
	6 HOURS				2 MONTHS	2 WEEKS	1 DAY
1 DAY						8 WEEKS	5 DAYS

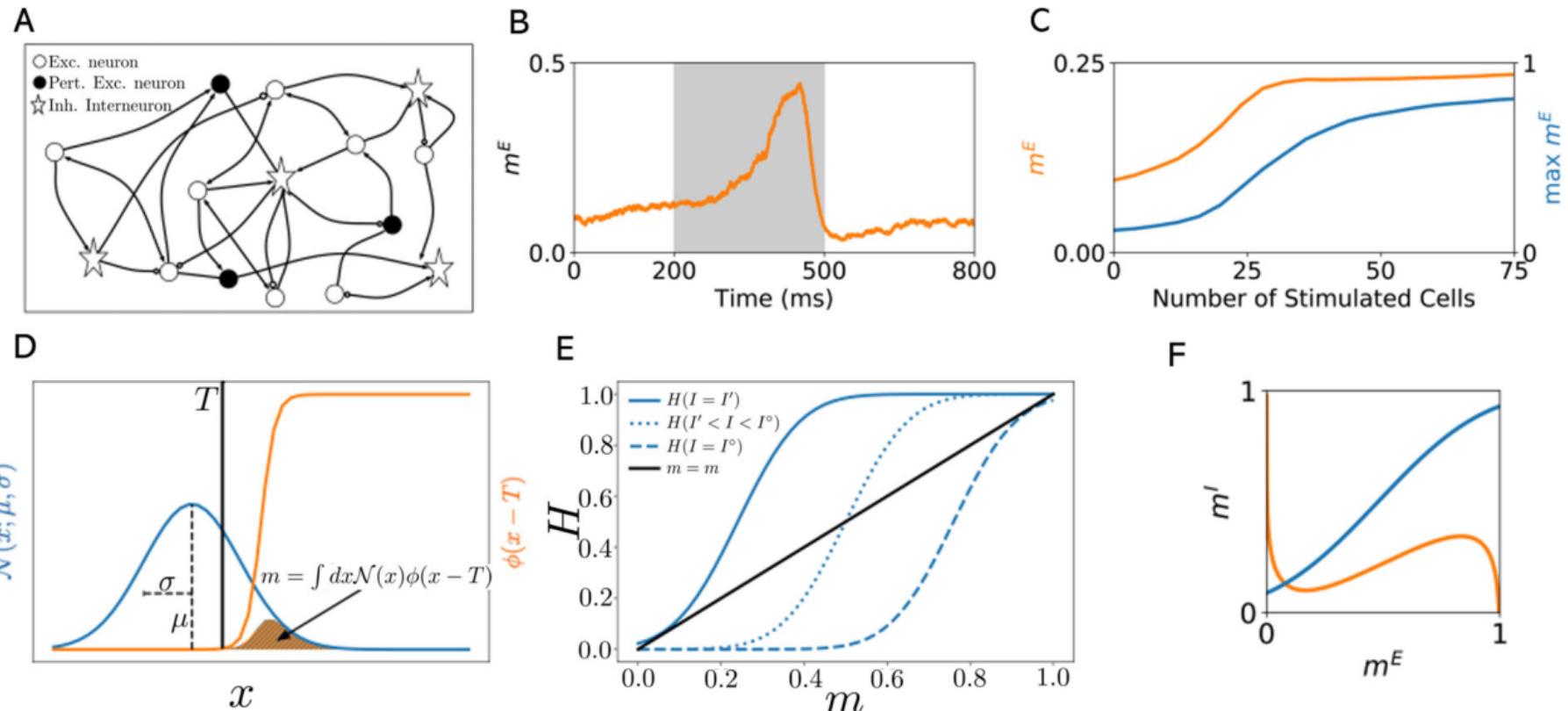
Why RFs?

- **Time-efficient:** Avoid repetition
- Figures are **reproducible** & traceable
 - Trace back exact methods from paper figures
 - Anyone can easily reproduce [with different data or different parameters]

Why RFs?

- **Time-efficient:** Avoid repetition
- Figures are **reproducible** & traceable
- '**Good coding**' makes 'bugs' less likely
 - 'All computer errors are caused by code, so less code is better'
 - Active (latent) workspaces like Jupyter are dangerous!

Bad figure-coding practices can lead to confusing figures & mistakes



Marshel ea, 2019, Science, Fig S17;
Colours are incorrect in panel F (see caption)

Why RFs?

- **Time-efficient:** Avoid repetition
- Figures are **reproducible** & traceable
- '**Good coding**' makes bugs less likely
- **Resource** for yourself, lab & community
 - Higher impact of publication if (good) code is included
 - Imagine a world where all papers would include reproducible figures (code)!

Why RFs?

- **Time-efficient:** Avoid repetition
- Figures are **reproducible** & traceable
- '**Good coding**' makes bugs less likely
- **Resource** for yourself, lab & community
- **Version control** of full figs (easy with for example pdf)
 - Go back through figure pdfs
 - Easier to collaborate

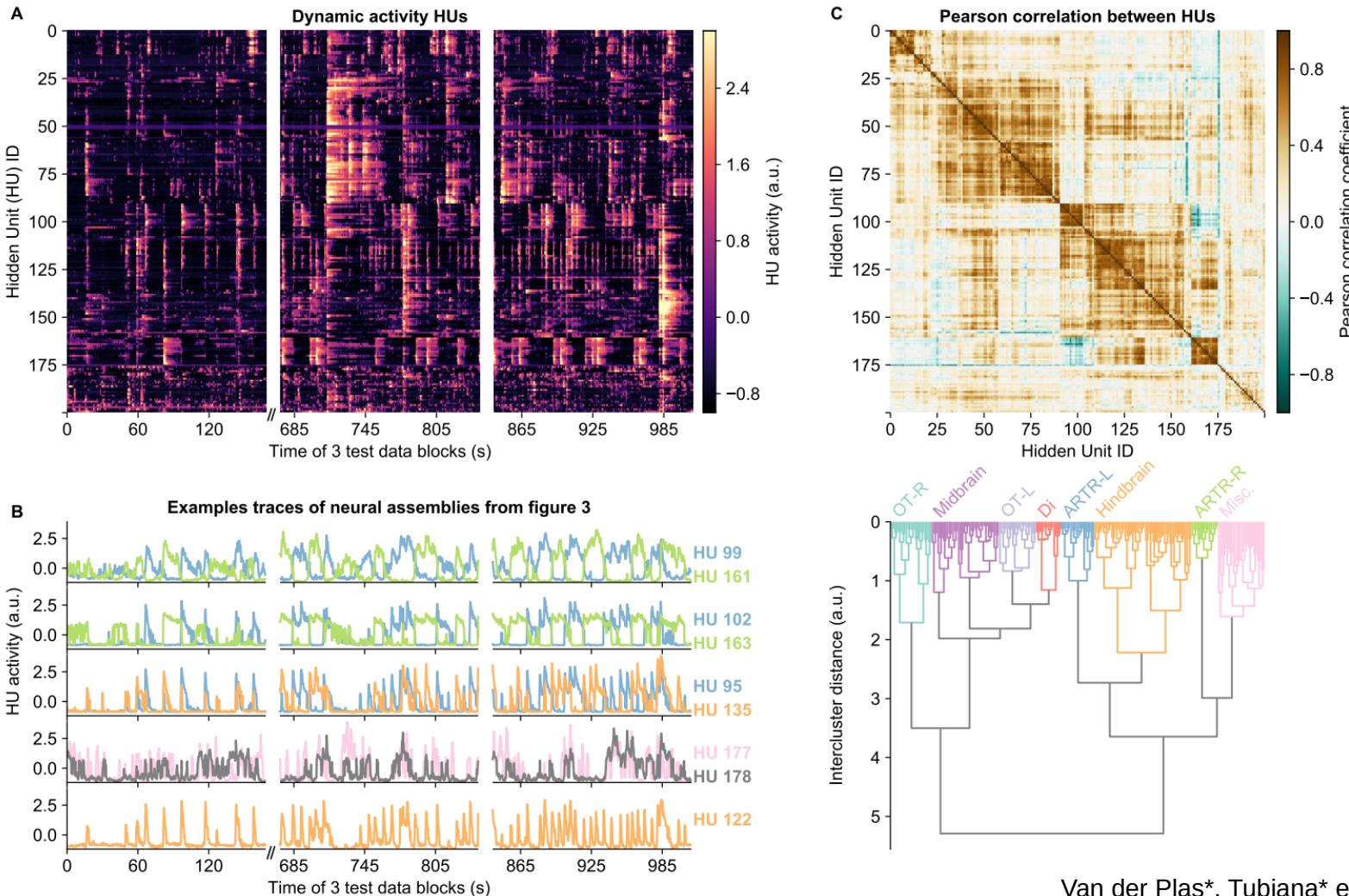
Why RFs?

- **Time-efficient**: Avoid repetition
- Figures are **reproducible** & traceable
- '**Good coding**' makes bugs less likely
- **Resource** for yourself, lab & community
- **Version control** of full figs (easy with for example pdf)
- Fun!! => great & **easy way to learn** better coding
 - Figure-coding = visual coding => direct visualisation of code output

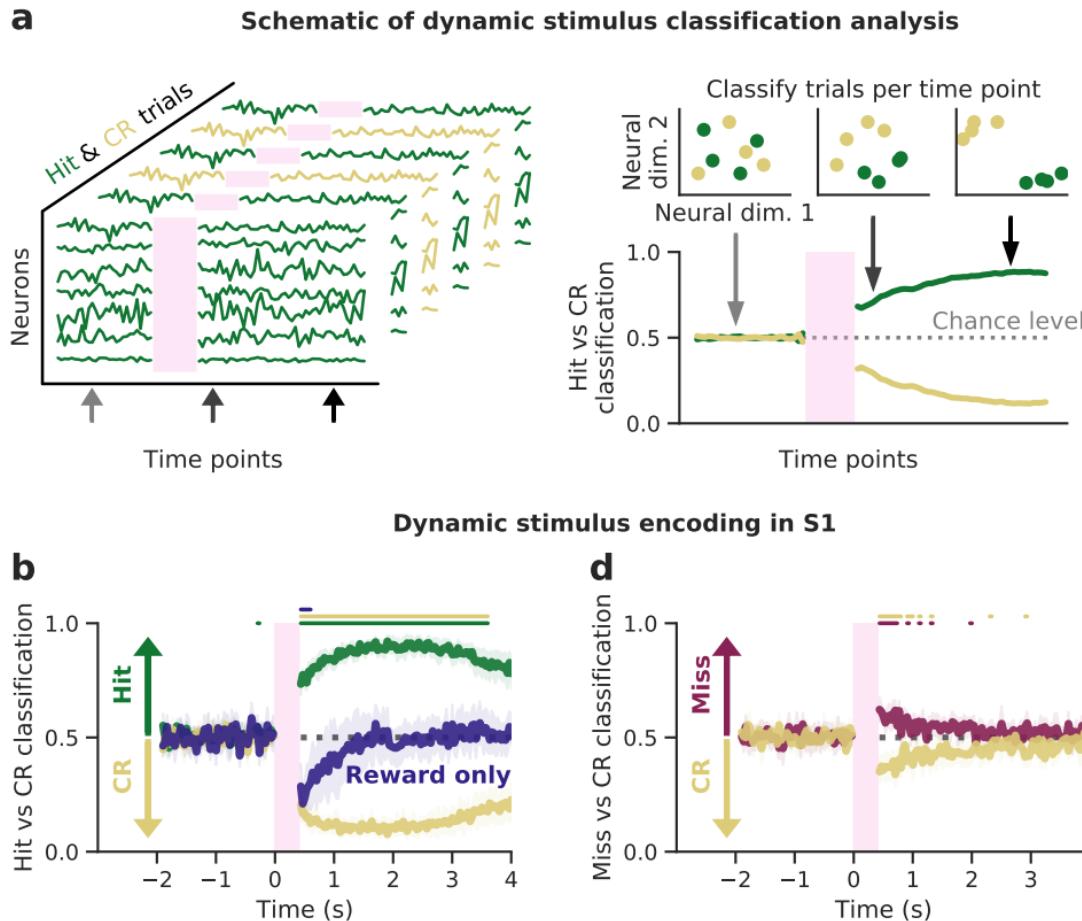
Why RFs?

- **Time-efficient**: Avoid repetition
- Figures are **reproducible** & traceable
- '**Good coding**' makes bugs less likely
- **Resource** for yourself, lab & community
- **Version control** of full figs (easy with for example pdf)
- Fun!! => great & **easy way to learn** better coding
- More **possibilities**

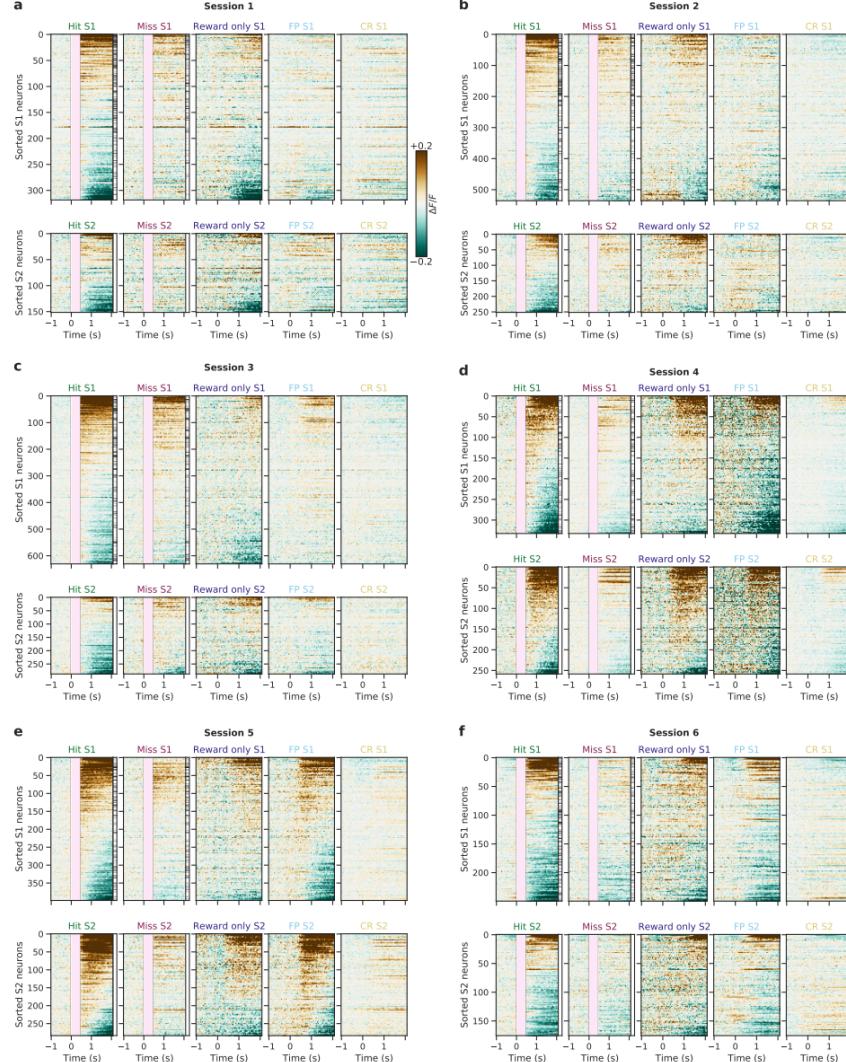
Example: across-panel alignment



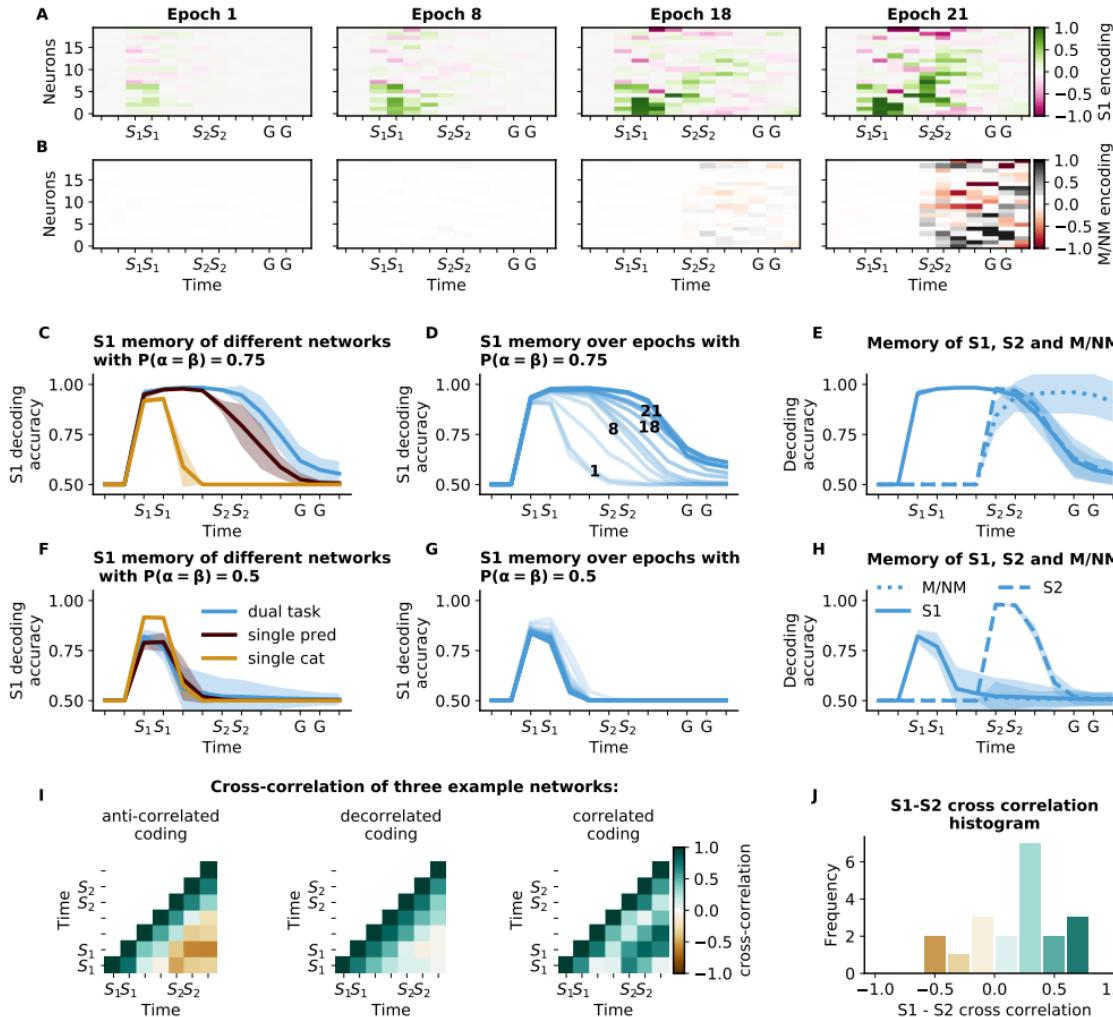
Example: consistent fig style with python-schematics



Example: easy to generate supplemental figures

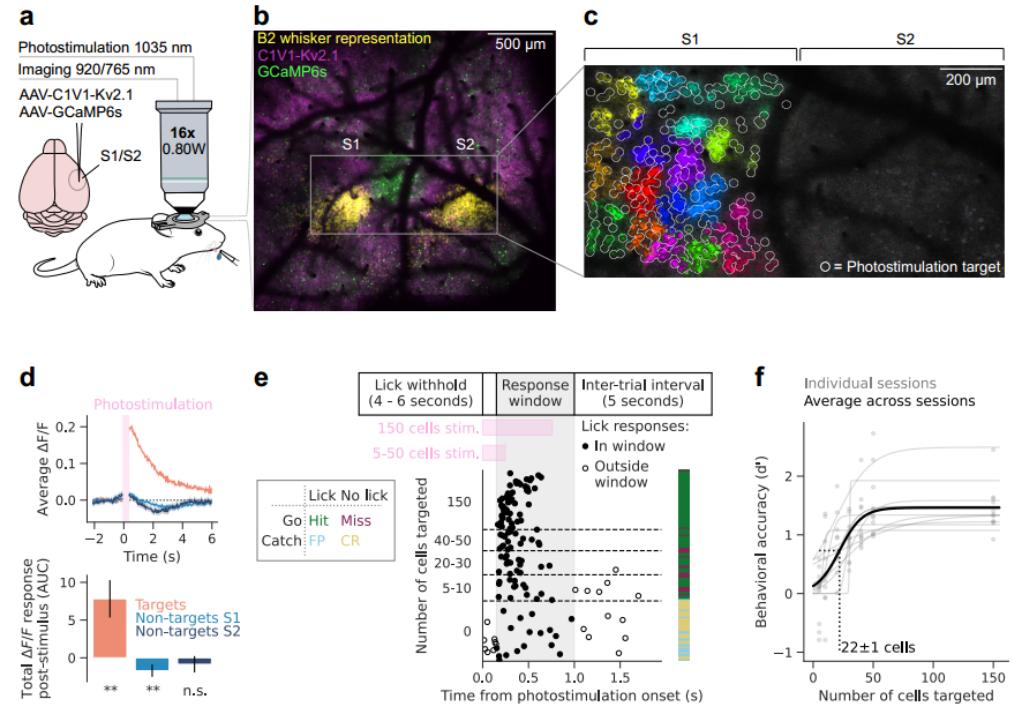


Example: easy to change highlighted examples



What about Illustrator?

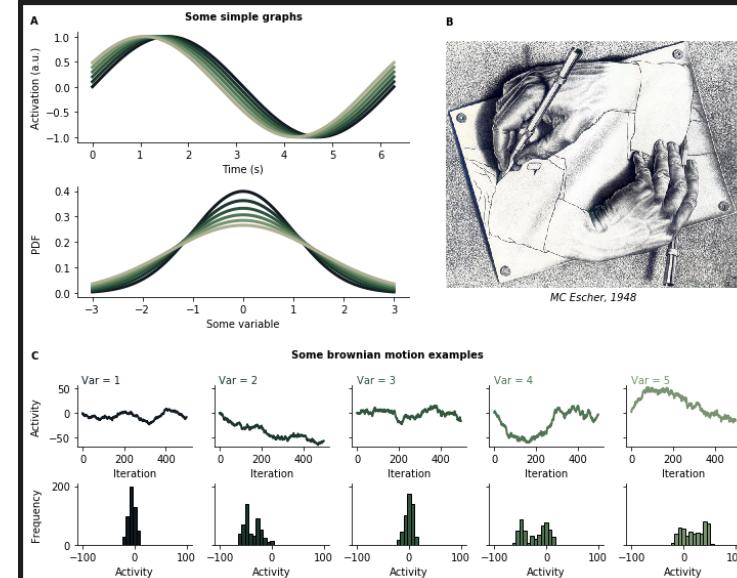
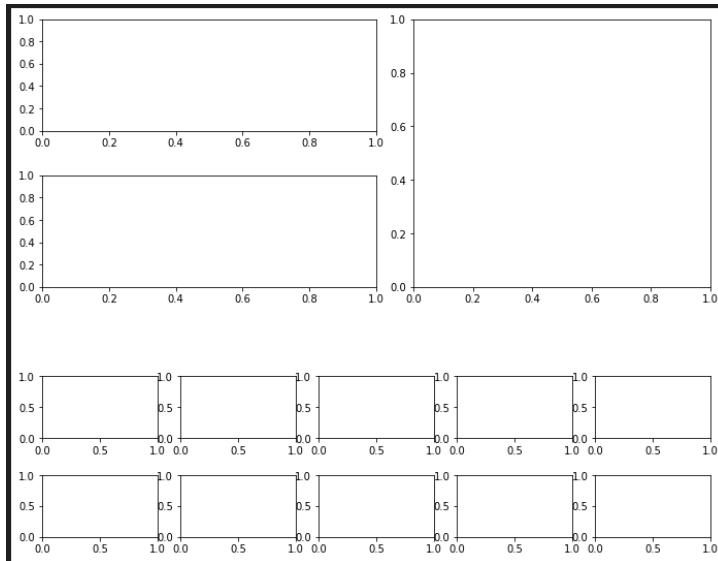
- Obvious use-cases exclusive to Illustrator
- These can be imported into code-generated RFs!
- Some elements seem easier in Illustrator (positioning text/visual aids). But think about avoid-repetition efficiency!



Rowland et al. Fig 1; top row contains illustrator-made elements that were imported into Python before generating full figure

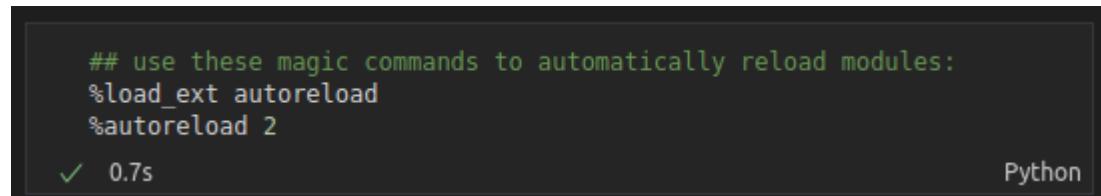
4 principles to make RFs in Python

- Separate panel *content* from panel *lay-out*
 - For content: functions that take 1) ax handle, 2) data, 3) parameters
 - For lay-out: use matplotlib's gridspec



4 principles to make RFs in Python

- Separate panel *content* from panel *lay-out*
- Work with modules (.py), use workspace (.ipynb) minimally
 - Avoid errors due to latent variables in workspace!!
 - Allows version control of modules
 - Easier to reuse code when in modules



```
## use these magic commands to automatically reload modules:  
%load_ext autoreload  
%autoreload 2
```

✓ 0.7s Python

A screenshot of a Jupyter Notebook cell. The cell contains three lines of code: '%load_ext autoreload', '%autoreload 2', and '# use these magic commands to automatically reload modules:'. The first two lines are highlighted in green, indicating they have been run. A green checkmark icon is on the left, and the execution time '0.7s' is at the bottom left. The word 'Python' is at the bottom right.

rep_fig_vis.py 3 × pop_off_plotting.py 9+

```
106
107 ##### SPECIFIC FUNCTIONS FOR THIS TUTORIAL
108 #####
109 #####
110
111 def plot_sin_one_period(ax=None, n_tp=500, phase=0, alpha=1, colour='k'):
112     '''Create sine over 1 period with offset phase'''
113     if ax is None:
114         ax = plt.subplot(111)
115
116     t_array = np.linspace(0, 2 * np.pi, n_tp)
117     sin_array = np.sin(t_array + phase)
118
119     ax.plot(t_array, sin_array, linewidth=3, alpha=alpha, c=colour)
120     ax.set_xlabel('Time (s)')
121     ax.set_ylabel('Activation (a.u.)')
122     ax.set_title('Some simple graphs', y=1.05, fontdict={'weight': 'bold'})
123
124
125 def plot_normal_distr(ax=None, n_tp=500, mean_distr=0, std_distr=1, alpha=1, c
126     '''Create sine over 1 period with offset phase'''
127     if ax is None:
128         ax = plt.subplot(111)
129
130     t_array = np.linspace(-3, 3, n_tp)
131     norm_array = scipy.stats.norm.pdf(t_array, loc=mean_distr, scale=std_distr
132
133     ax.plot(t_array, norm_array, linewidth=3, alpha=alpha, c=colour)
134     ax.set_xlabel('Some variable')
135     ax.set_ylabel('PDF')
136
137 def plot_brown_proc(ax_trace=None, ax_hist=None, var=1, n_steps=500,
138                     plot_ylabel=True, colour='k'):
139     '''Sample brownian motion & plot trace and histogram'''
140     gauss_samples = np.random.randn(n_steps) * np.sqrt(var)
141     brown_motion = np.cumsum(gauss_samples)
142     time_array = np.arange(n_steps)
143
144     if ax_trace is None or ax_hist is None:
145         fig, ax = plt.subplots(1, 2)
146         ax_trace, ax_hist = ax
147
148     ax_trace.plot(time_array, brown_motion, linewidth=2, c=colour)
149     ax_trace.set_xlabel('Iteration')
150     if plot_ylabel:
151         ax_trace.set_ylabel('Activity')
152
153     ax_hist.hist(brown_motion, bins=np.linspace(-100, 100, 30),
154                  facecolor=colour, edgecolor='k', linewidth=1)
155     ax_hist.set_xlabel('Activity')
156     if plot_ylabel:
157         ax_hist.set_ylabel('Frequency')
```



Reproducible figure example.ipynb ●

Reproducible figure example.ipynb > empty cell

```
+ Code + Markdown | ▶ Run All | Clear Outputs of All Cells | ▷ Outline ...
+---+---+
ax_m misc[0][1].misc).annotate(s=i_val - i_curr_val, xy=(0.04, 1), va='bottom',
                                xycoords='axes fraction', c=colours_misc_dict[i_misc])
if plot_ax_tidy:
    for i_row in range(n_misc_rows):
        rfv.equal_lims_n_axs(ax_list=list(ax_misc[i_row].values()))
fig.align_ylabels(axes=[ax_cat[0], ax_cat[1], ax_misc[0][0], ax_misc[1][0]])

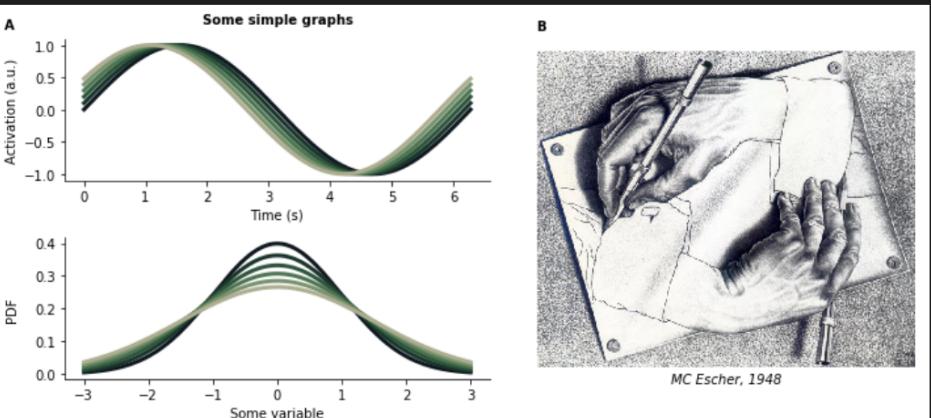
if plot_extra:
    ## First way of doing text, by plt.text()
    ax_cat[0].text(s='A', xc=-1.29, yc=1.25, # specify coords in data coord system of this ax
                   fontdict={'weight': 'bold'})
    ax_cat[0].text(s='C', xc=-1.29, yc=-5.4, # specify coords in data coord system of this ax
                   fontdict={'weight': 'bold'})

    ax_im.text(s='MC Escher, 1948', x=np.mean(list(ax_im.get_xlim())), y=ax_im.get_ylim()[0] + 10, # get
               fontdict={'ha': 'center', 'va': 'top', # change text alignment to make centering easier
                         'style': 'italic'})

    ## Alternatively, use annotate to specify coords in fraction of ax or fig
    ## (this is actually usually also easier to align panel labels)
    ax_im.annotate(s='B', xy=(0.578, 0.965), xycoords='figure fraction',
                   weight='bold')
    ax_im.annotate(s='Some brownian motion examples', xy=(0.5, 0.39), xycoords='figure fraction',
                   ha='center', weight='bold')

if save_fig:
    plt.savefig('Example_rep_fig.pdf', bbox_inches='tight')
```

✓ 1.2s

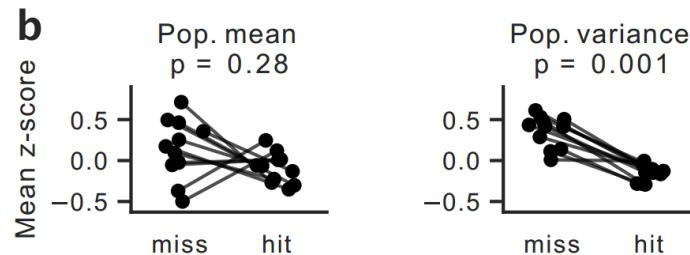


MC Escher, 1948

Python

4 principles to make RFs in Python

- Separate panel *content* from panel *lay-out*
- Work with modules (.py), use workspace (.ipynb) minimally
- Separate data analysis (functions) from plotting (functions)
 - Save time: analysis only needs to be done once, plotting pretty often
 - Computationally intensive data analysis can be done separately and stored (and loaded for visualisation). But DO NOT store into data file!!
 - Additionally: separate data selection & plotting => pure function



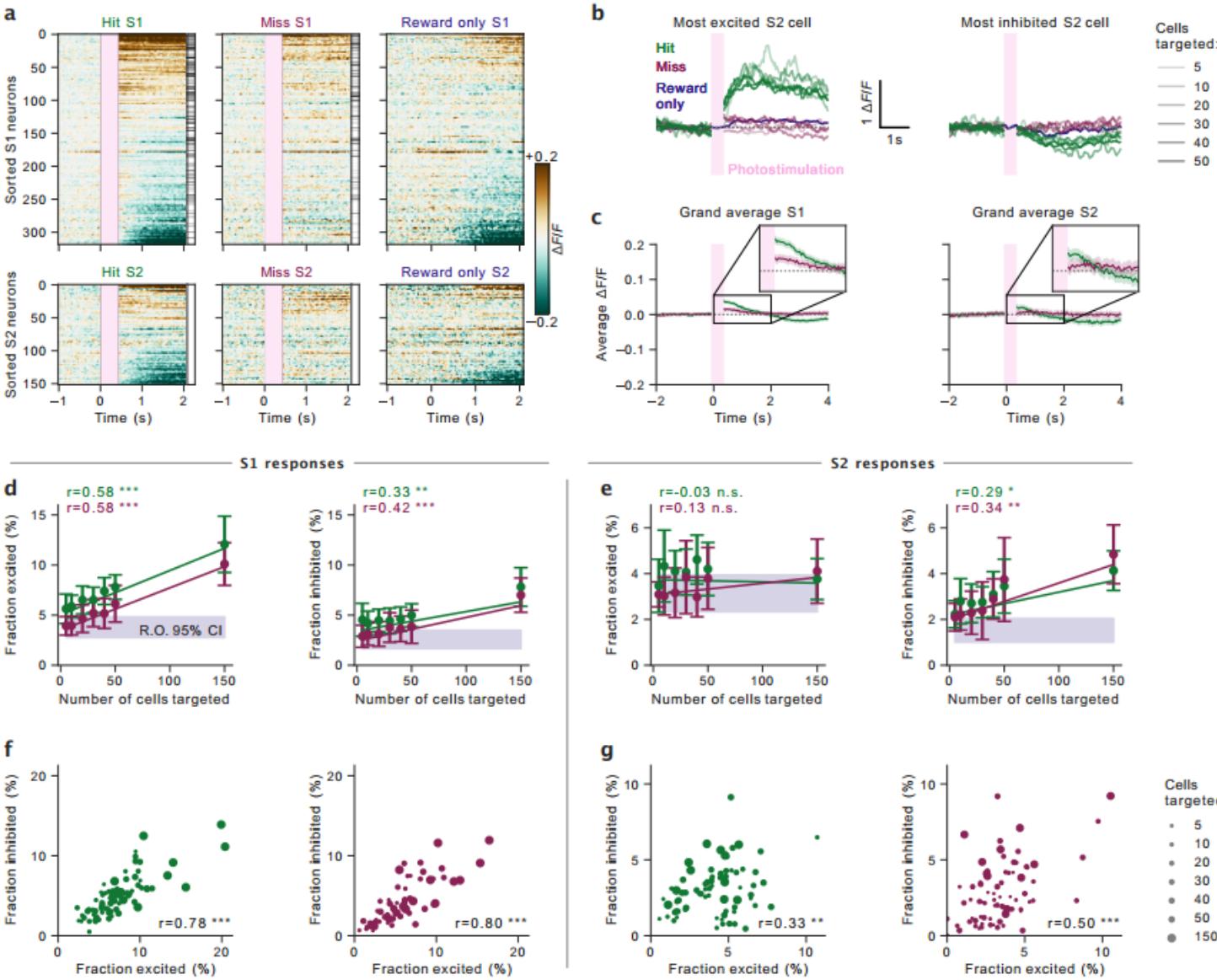
4 principles to make RFs in Python

- Separate panel *content* from panel *lay-out*
- Work with modules (.py), use workspace (.ipynb) minimally
- Separate data analysis (functions) from plotting (functions)
- “*Write functions, not scripts*”
 - Object-oriented programming:
one of Python’s strengths => use it!
 - Never copy-paste, but wrap code in
object and reuse (DRY)
 - OOP leads to faster code, less code,
bug-free code

```
92     def remove_xticklabels(ax):
93         '''remove x ticklabels but keep ticks'''
94         ax.set_xticklabels(['' for x in ax.get_xticklabels()])
95
96     def remove_yticklabels(ax):
97         '''remove y ticklabels but keep ticks'''
98         ax.set_yticklabels(['' for x in ax.get_yticklabels()])
99
100    def remove_both_ticklabels(ax):
101        '''both x and y'''
102        remove_xticklabels(ax)
103        remove_yticklabels(ax)
104
105
```

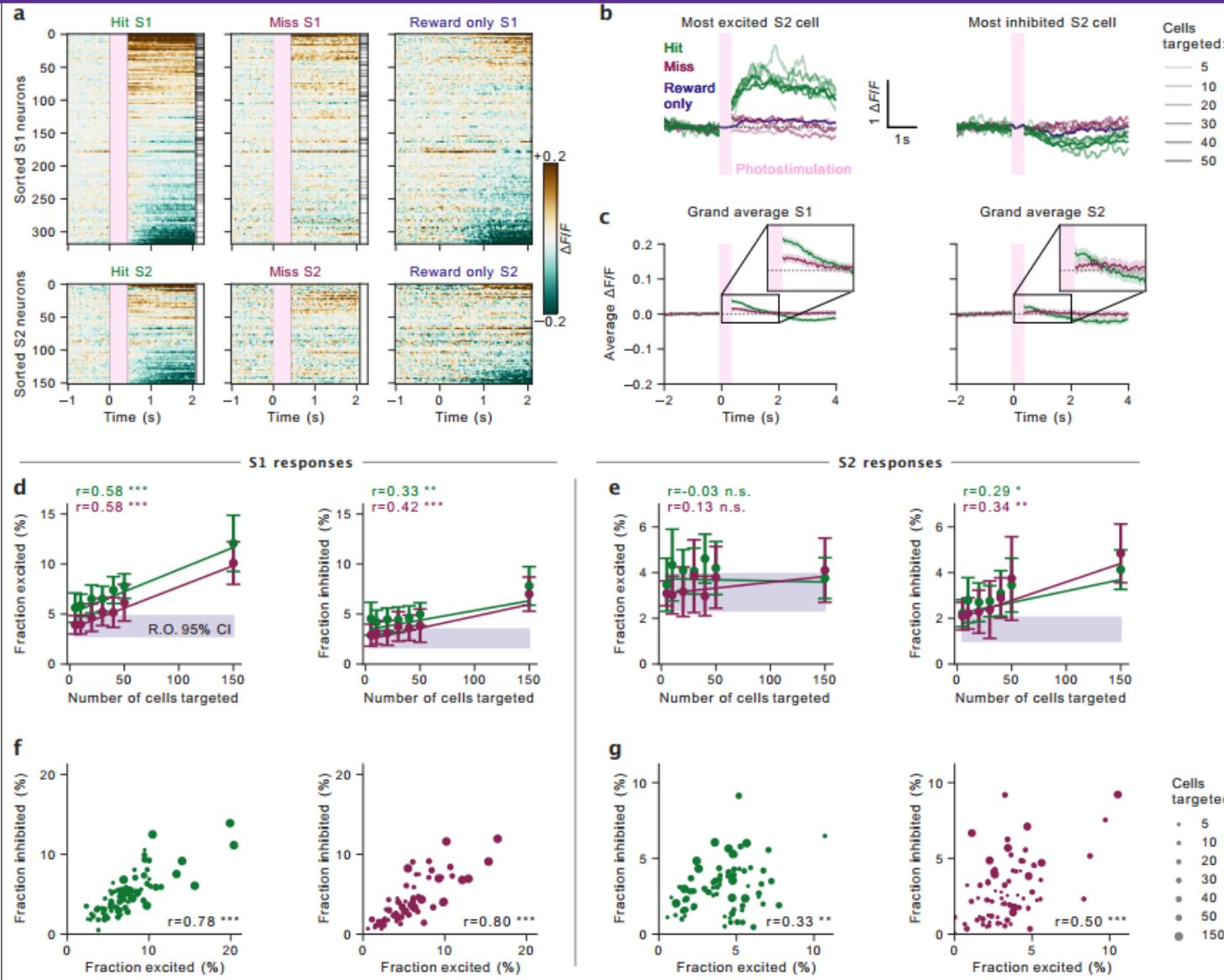
Example

- Rowland et al., Fig 2
- Let's deconstruct lay-out



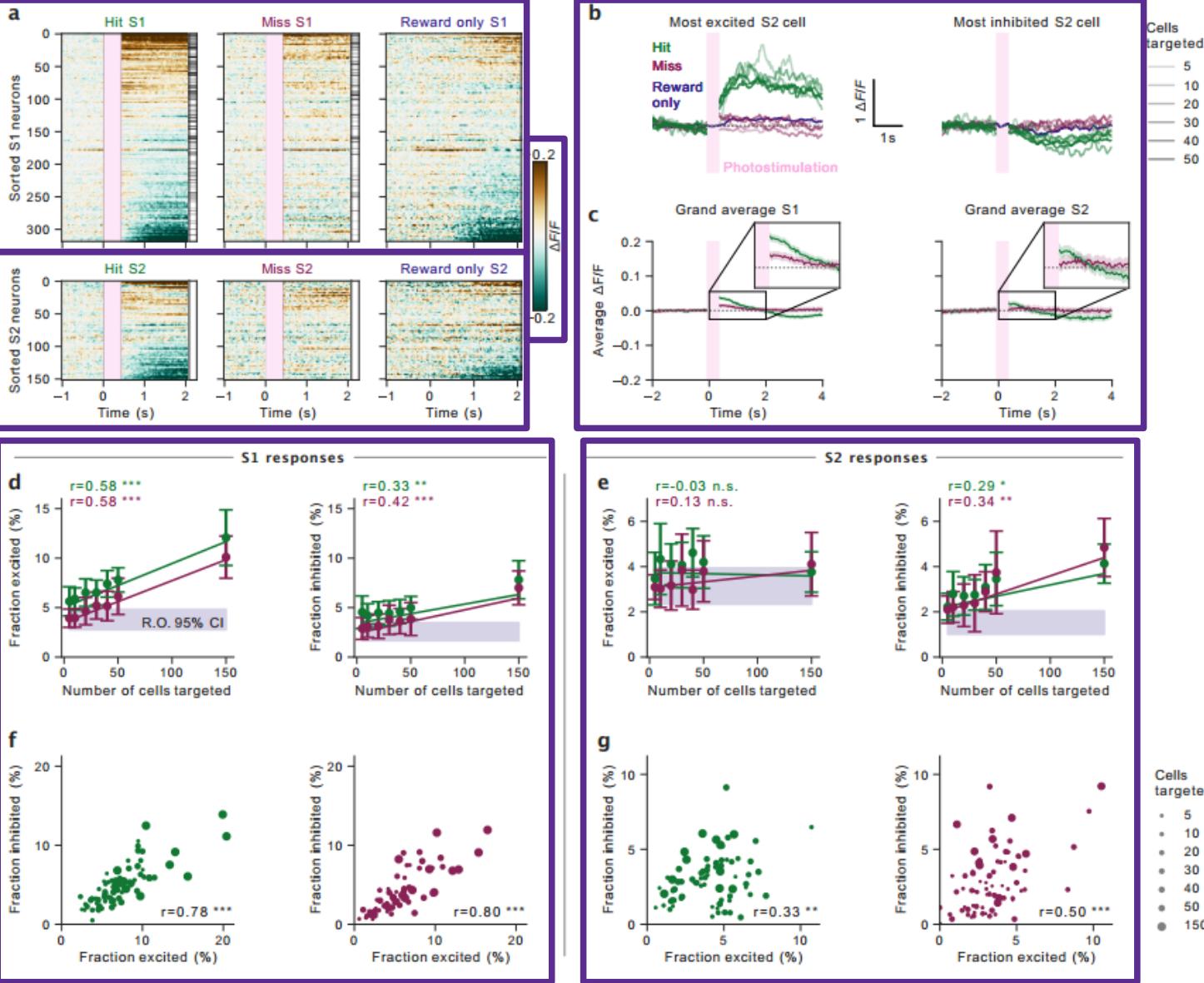
Example

- Figure



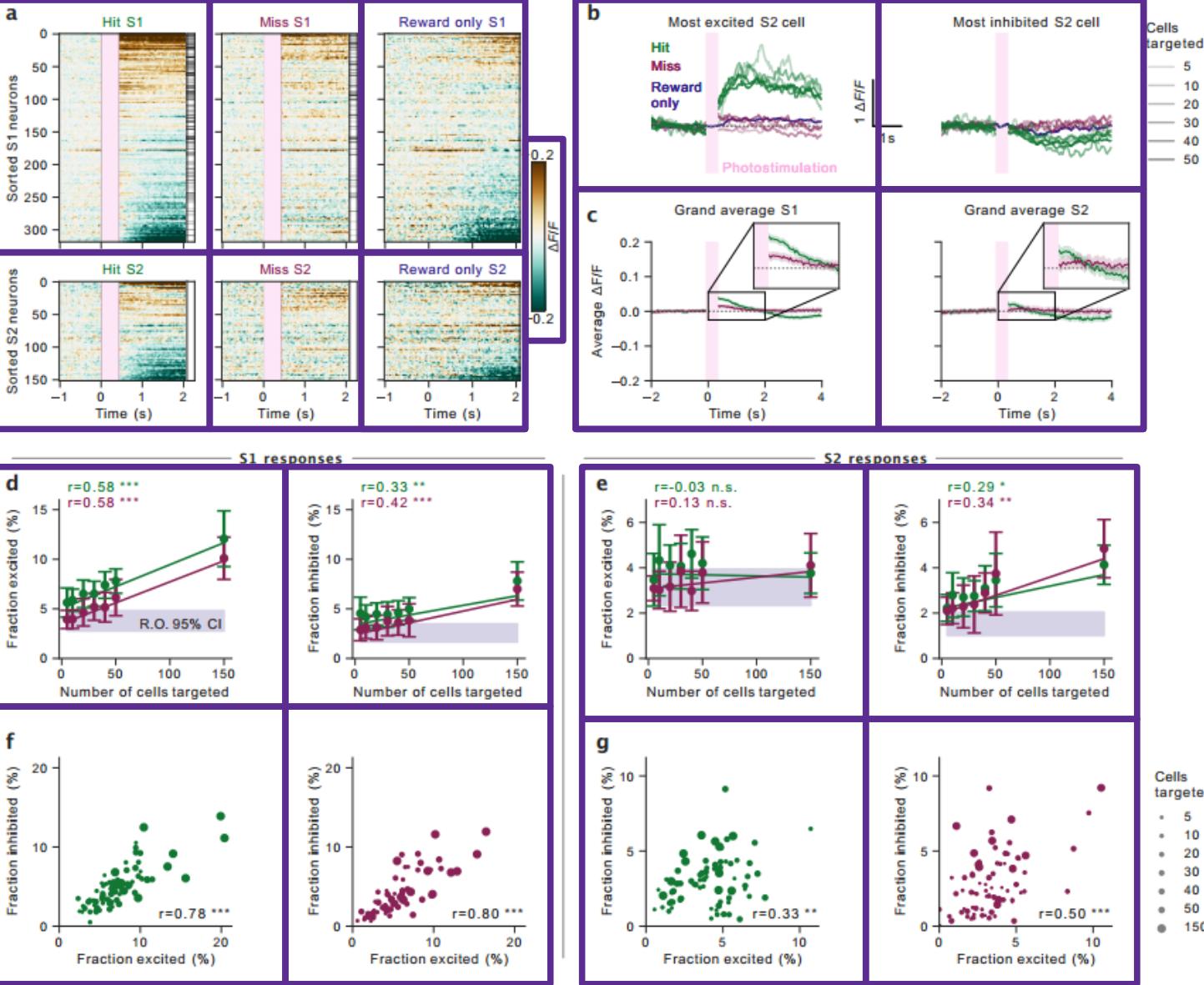
Example

- Figure
- Grids



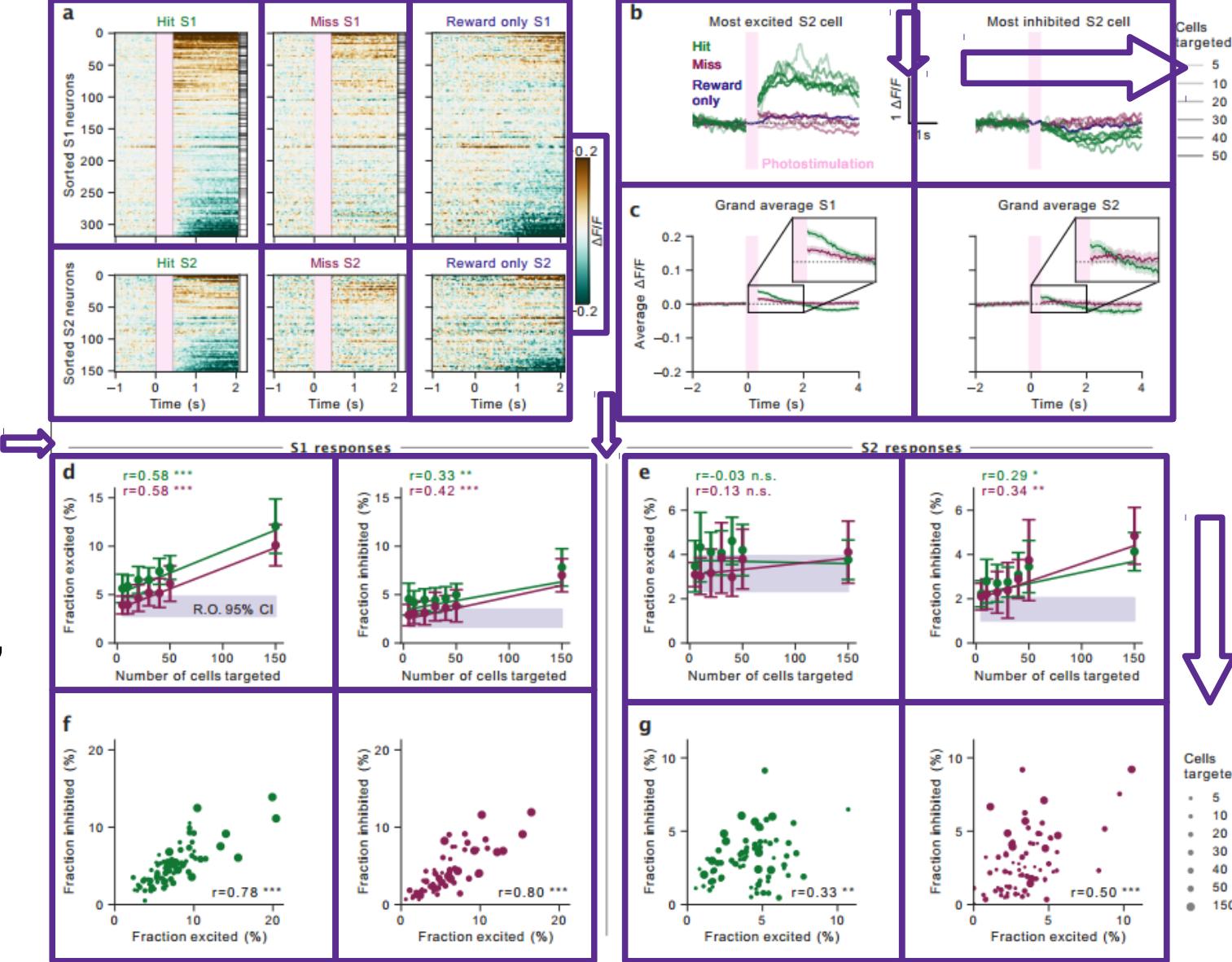
Example

- Figure
- Grids
- Axes



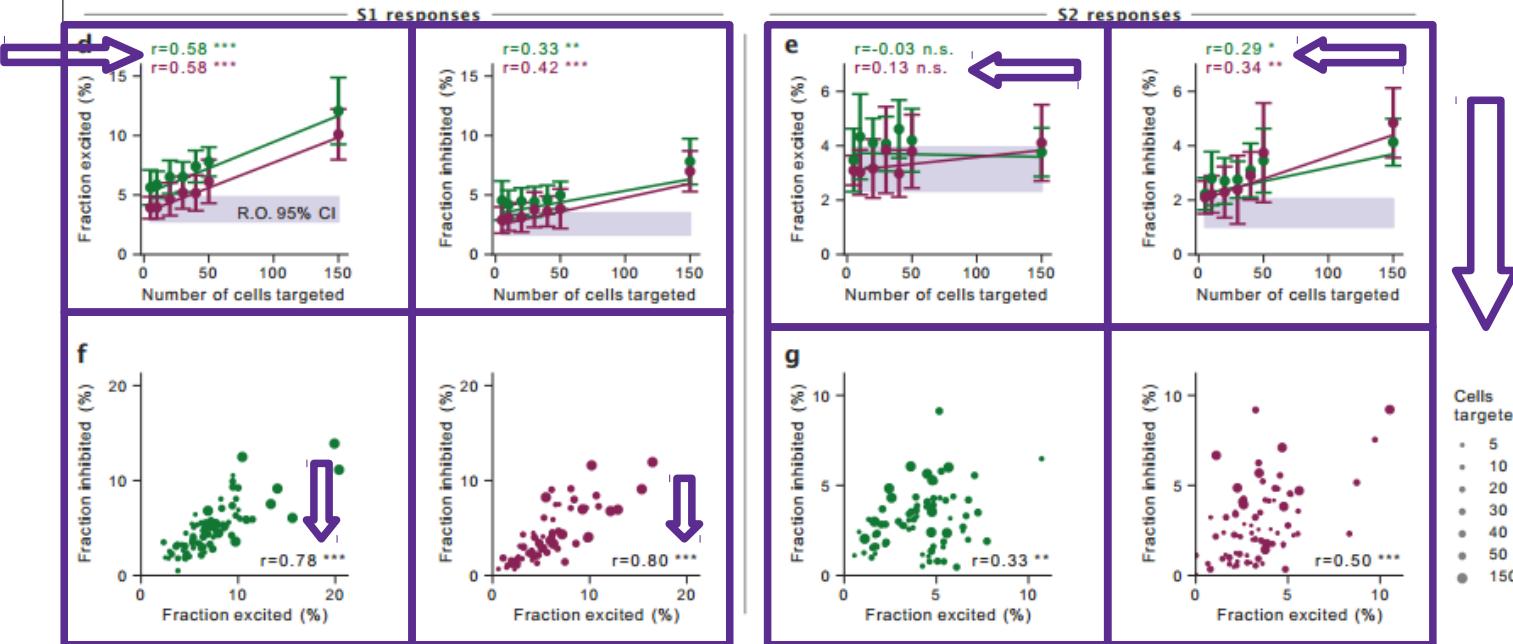
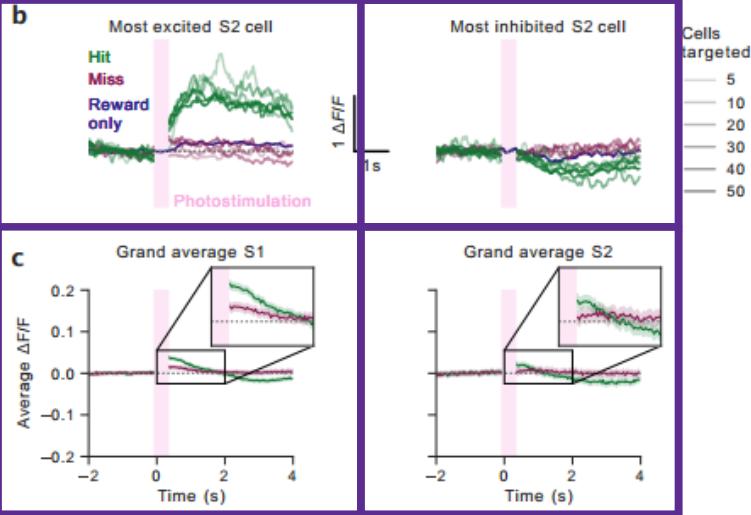
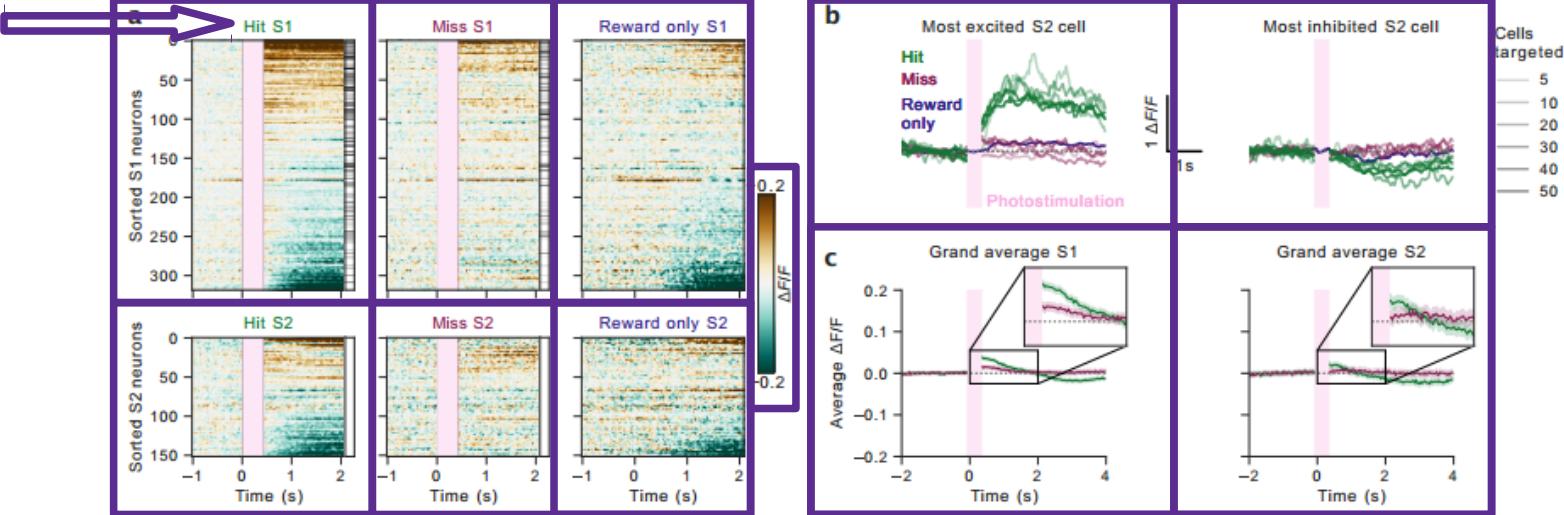
Example

- Figure
- Grids
- Axes
- Outside-ax-elements
 - In mpl functions, use arg: `clip_on=False`



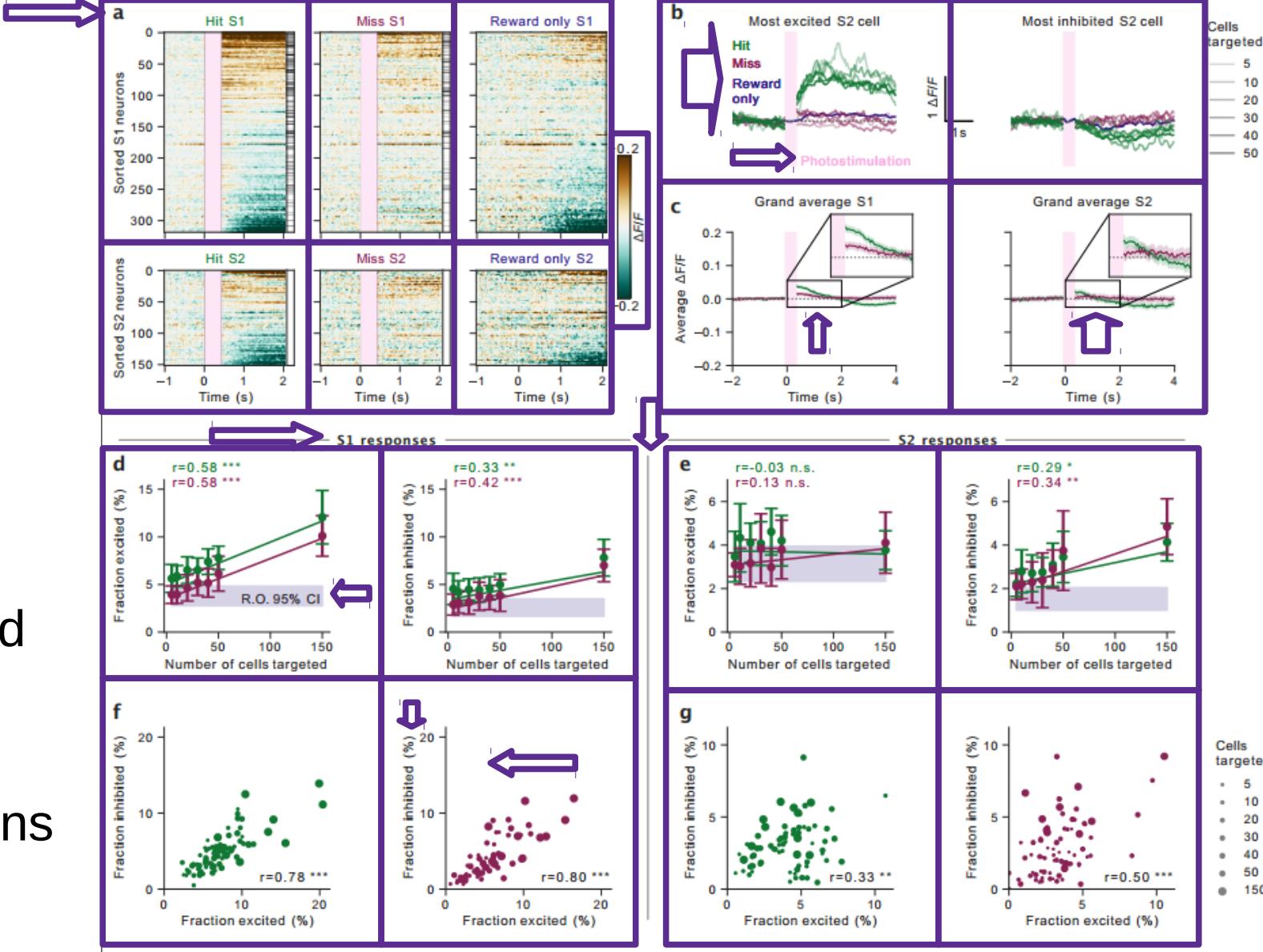
Example

- Figure
- Grids
- Axes
- Outside-ax-elements
- Auto-generated text, colour & numbers



Example

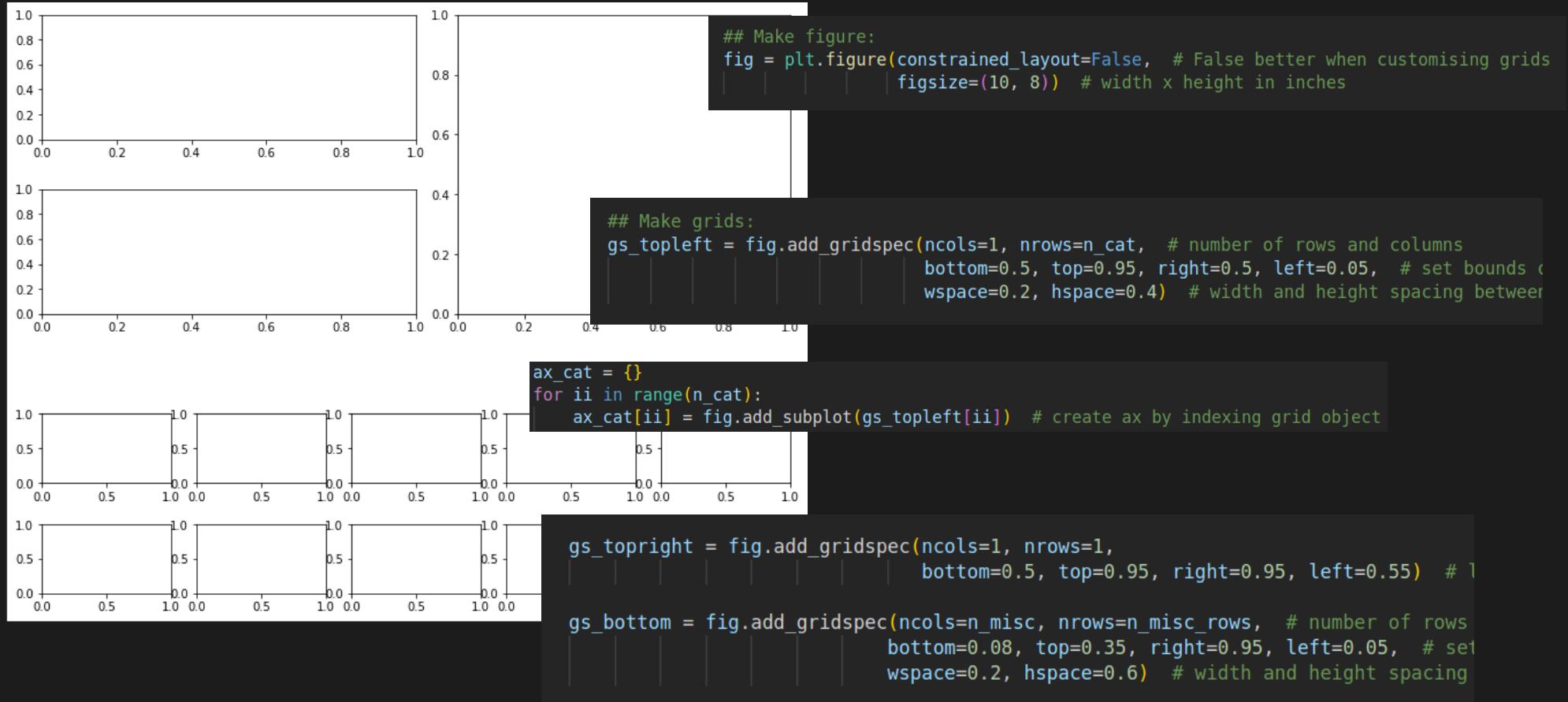
- Figure
- Grids
- Axes
- Outside-ax-elements
- Auto-generated text, colour & numbers
- Unique additions



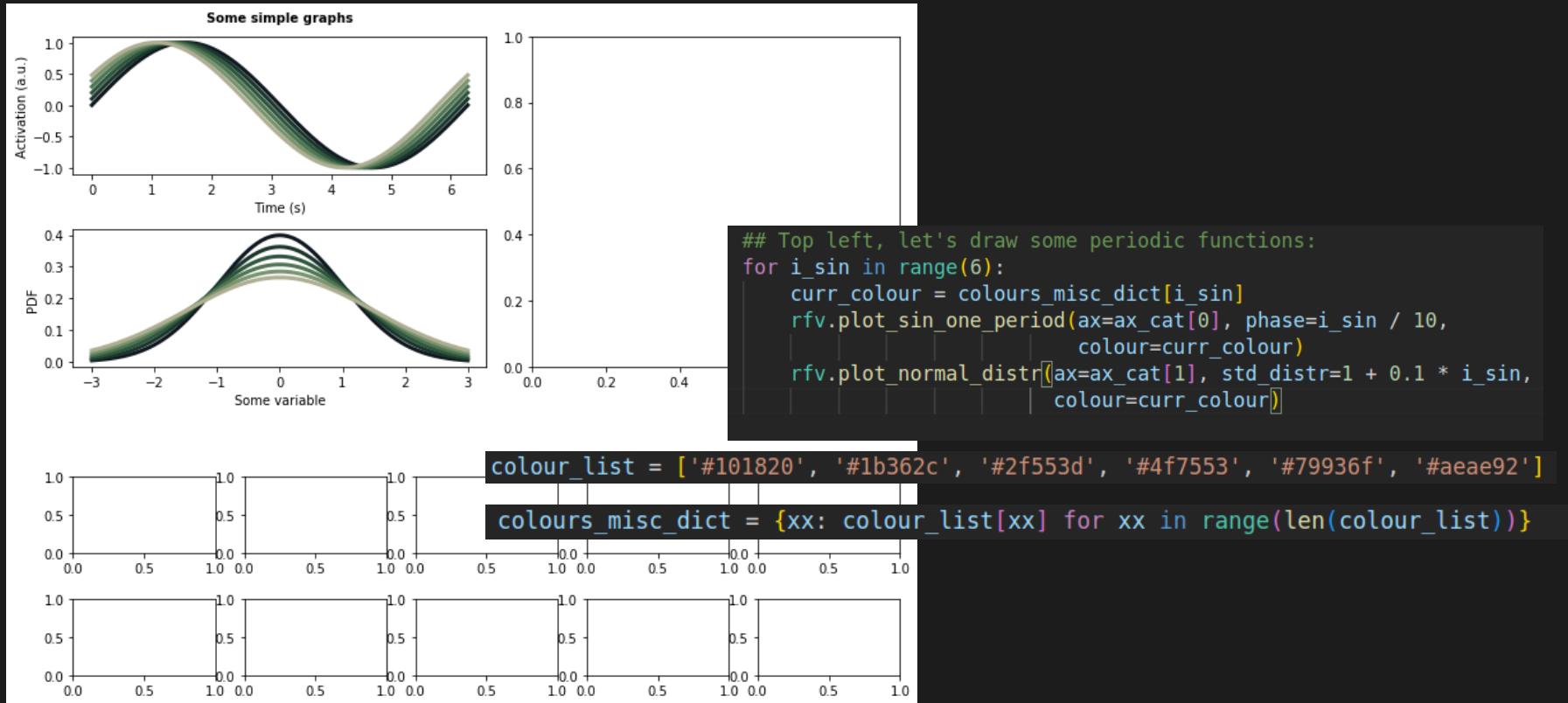
Example code

- Let's construct a RF
- Will show code, but logic of steps is the important bit
https://github.com/vdplasthijs/reproducible_figures

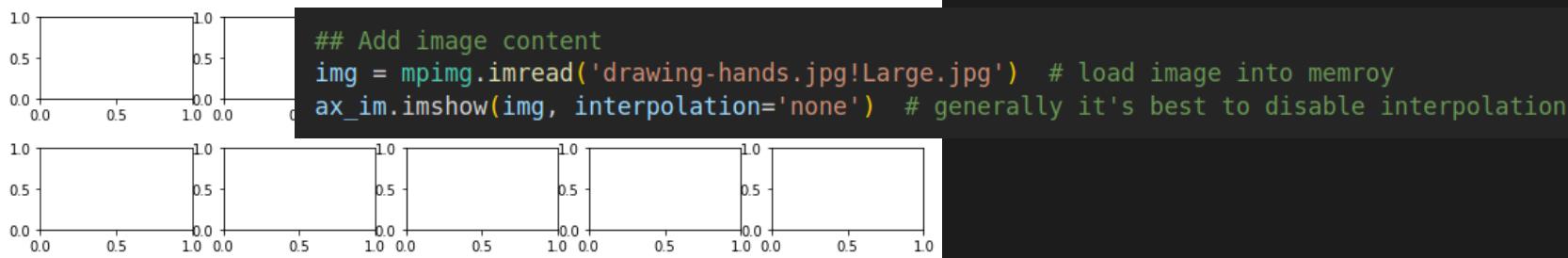
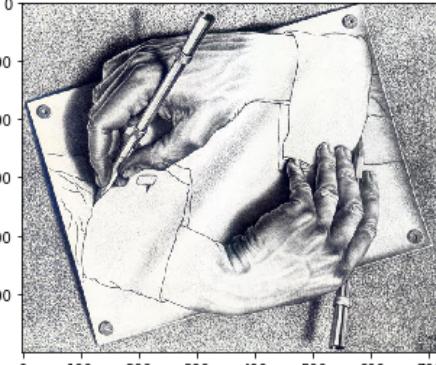
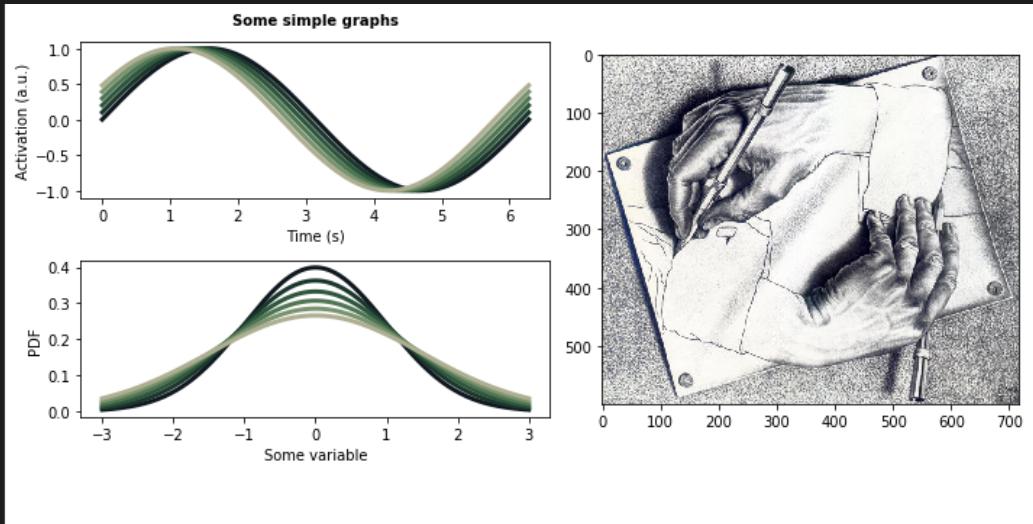
Example code – make grid & axes



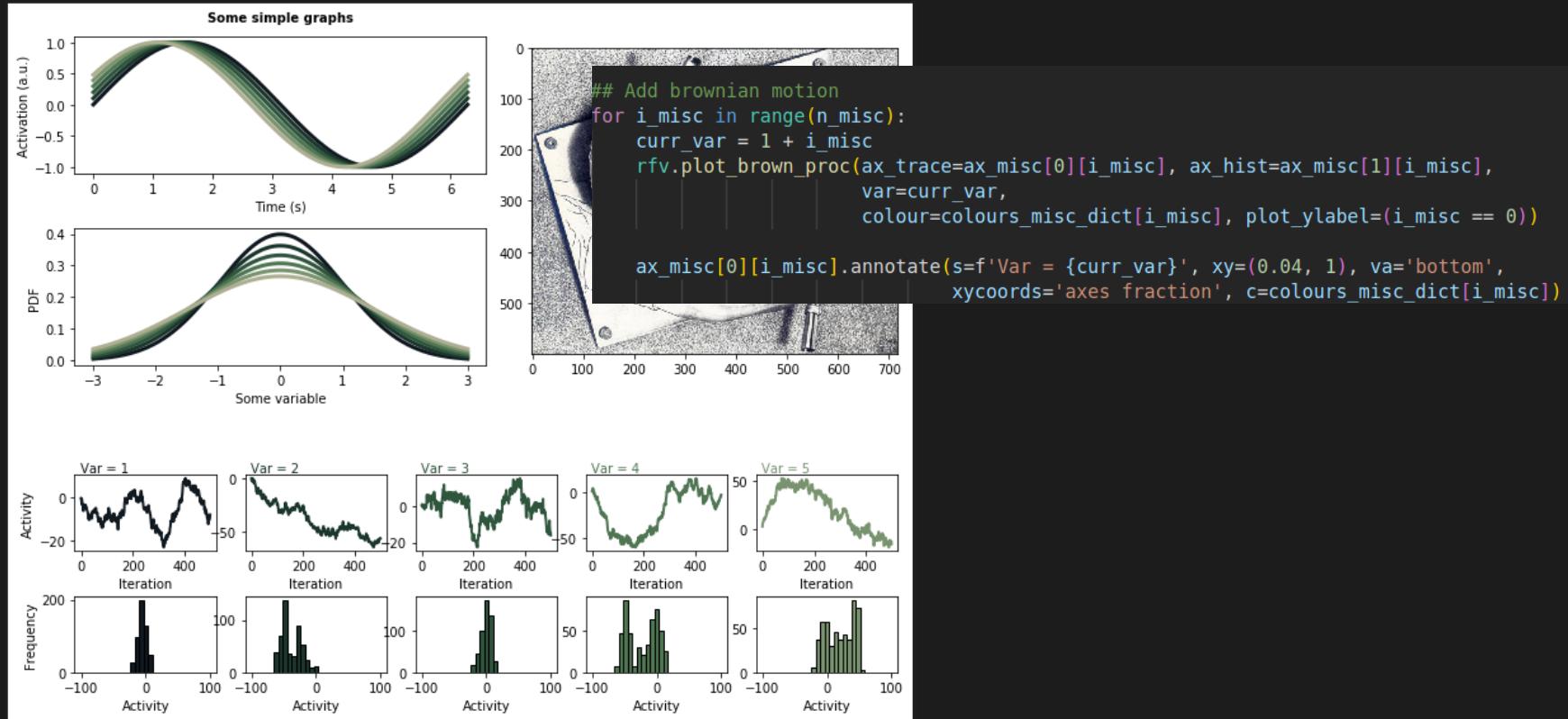
Example code – loop for soft-coded colours



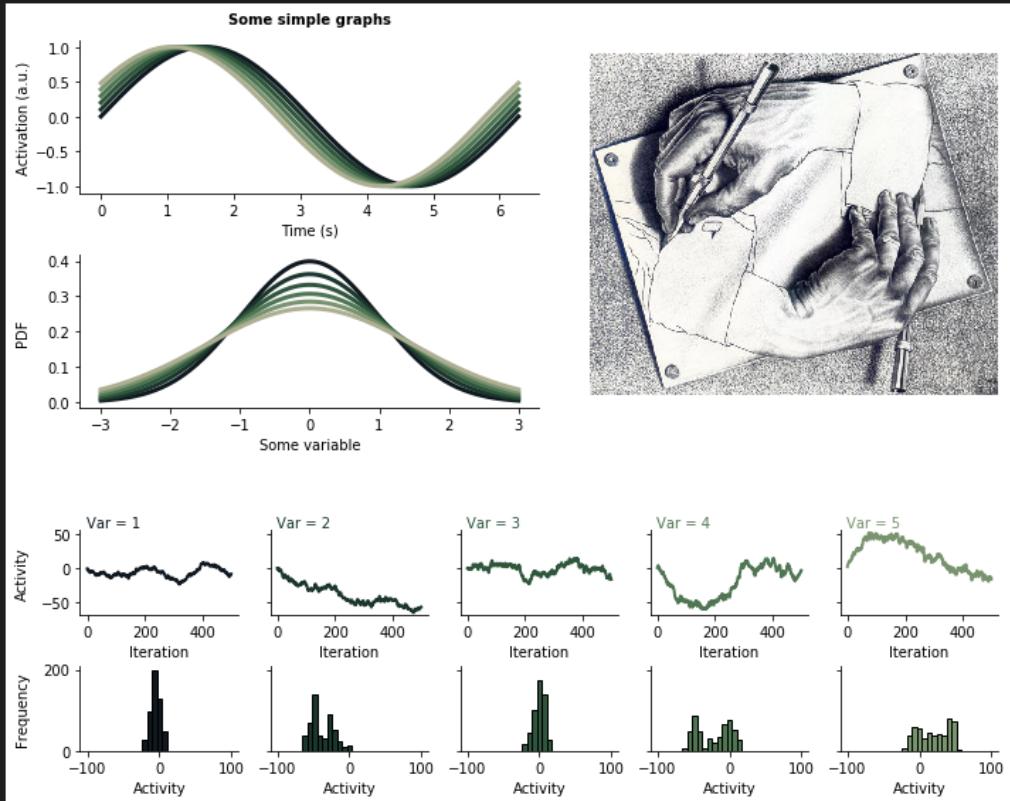
Example code – insert jpg/png images



Example code – more loop use cases



Example code – tidy up ax spines & labels

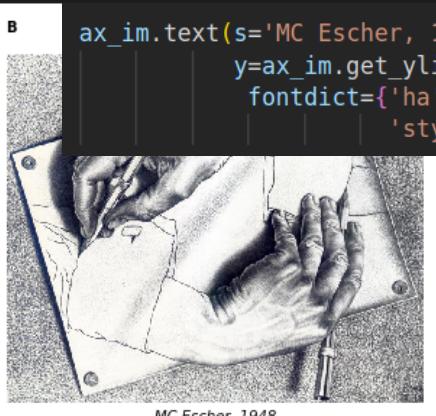
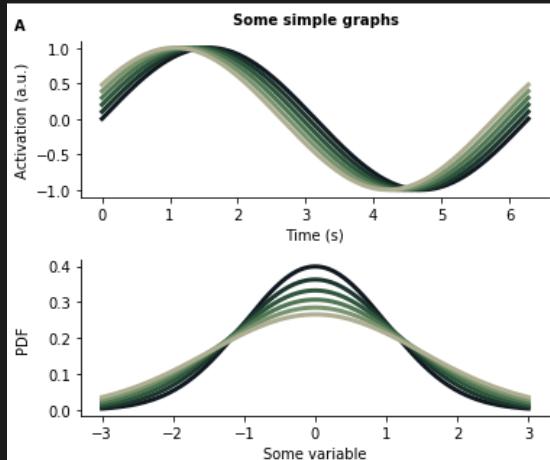


```
rfv.despine(ax_cat[ii])
```

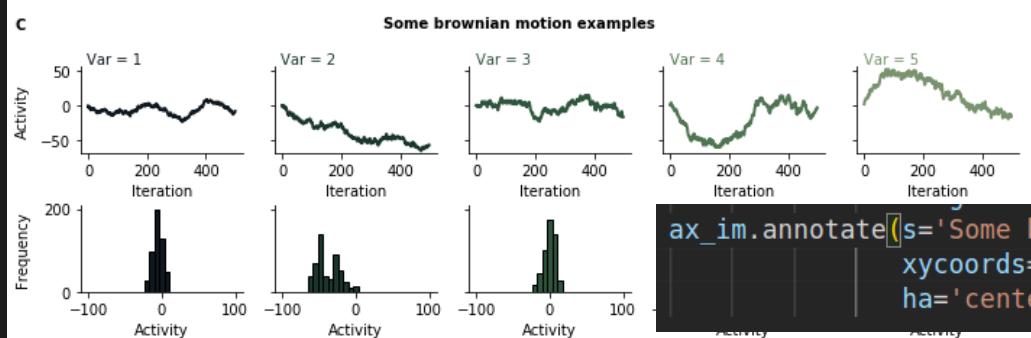
```
rfv.naked(ax_im)
```

```
rfv.remove_yticklabels(curr_ax)
```

Example code – add text elements

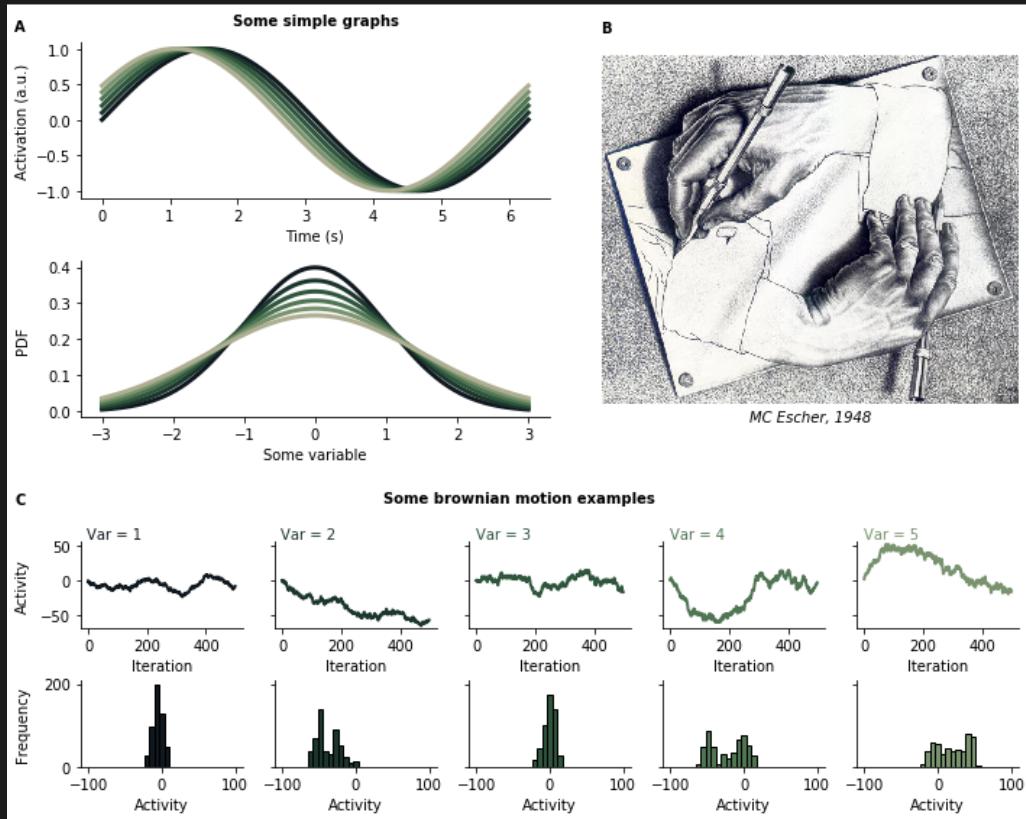


```
ax_im.text(s='MC Escher, 1948', x=np.mean(list(ax_im.get_xlim())),  
          y=ax_im.get_ylimits()[0] + 10, # get ax limits to define coords  
          fontdict={'ha': 'center', 'va': 'top', # change text alignment  
                    'style': 'italic'})
```



```
ax_im.annotate(s='Some brownian motion examples', xy=(0.5, 0.39),  
               xycoords='figure fraction',  
               ha='center', weight='bold')
```

Example code – done!



Tips & tricks

- Set figsize in cm/inches, set font size (all the same)

```
## Make figure:  
fig = plt.figure(constrained_layout=False, # False better when customising grids  
| | | | figsize=(10, 8)) # width x height in inches  
  
## Make grids:
```

Tips & tricks

- Set figsize in cm/inches, set font size (all the same)
- Set plt default parameters with plt.rcParams

```
def set_fontsize(font_size=12):
    plt.rcParams['font.size'] = font_size
    plt.rcParams['axes.autolimit_mode'] = 'data' # default: 'data'
```

Tips & tricks

- Set figsize in cm/inches, set font size (all the same)
- Set plt default parameters with plt.rcParams
- Save as pdf/svg for quality & file size
 - But rasterize large matrices/scatter plots..!

```
plt.savefig('Example_rep_fig.pdf', bbox_inches='tight')
```

Tips & tricks

- Set figsize in cm/inches, set font size (all the same)
- Set plt default parameters with plt.rcParams
- Save as pdf/svg for quality & file size
 - But rasterize large matrices/scatter plots..!
- Import png/jpg with plt.imshow() (watch out for anti-aliasing!) & import svg with svgutils

Tips & tricks

- Set figsize in cm/inches, set font size (all the same)
- Set plt default parameters with plt.rcParams
- Save as pdf/svg for quality & file size
 - But rasterize large matrices/scatter plots..!
- Import png/jpg with plt.imshow() (watch out for anti-aliasing!) & import svg with svgutils
- Most mpl functions maintain same syntax.

Tips & tricks

- Set figsize in cm/inches, set font size (all the same)
- Set plt default parameters with plt.rcParams
- Save as pdf/svg for quality & file size
 - But rasterize large matrices/scatter plots..!
- Import png/jpg with plt.imshow() (watch out for anti-aliasing!) & import svg with svgutils
- Most mpl functions maintain same syntax.
- Seaborn can complicate lay-out & perform additional analyses (KDE). Other packages exist too, but mpl has all the basic elements

Tips & tricks

- Set figsize in cm/inches, set font size (all the same)
- Set plt default parameters with plt.rcParams
- Save as pdf/svg for quality & file size
 - But rasterize large matrices/scatter plots..!
- Import png/jpg with plt.imshow() (watch out for anti-aliasing!) & import svg with svgutils
- Most mpl functions maintain same syntax.
- Seaborn can complicate lay-out & perform additional analyses (KDE). Other packages exist too, but mpl has all the basic elements
- Use numpy convention

numpydoc.readthedocs.io/en/latest/format.html

Colours



Colours

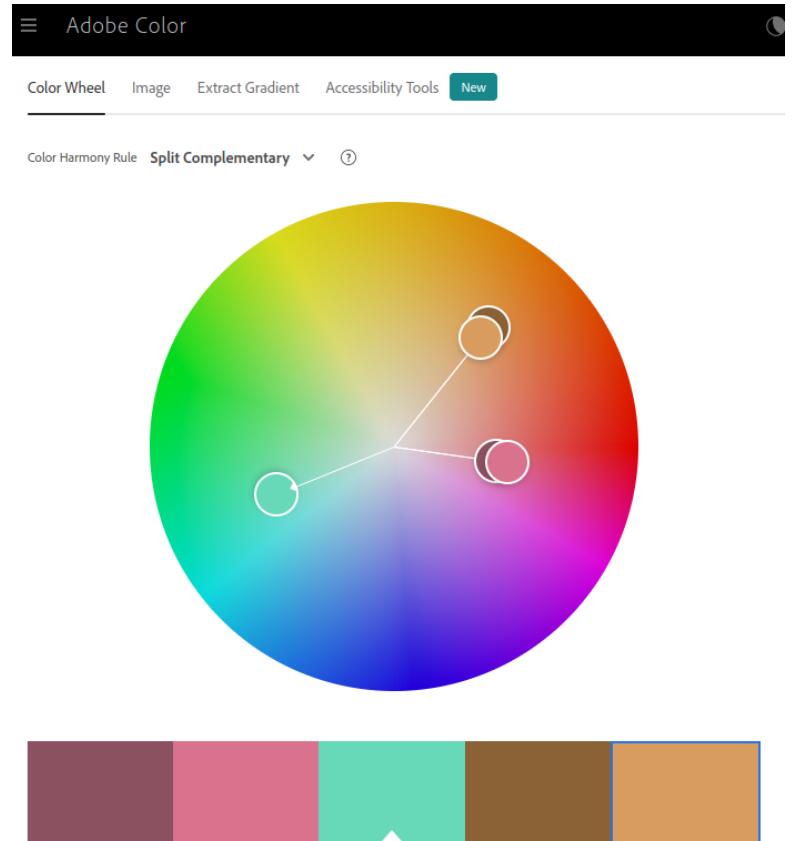
- Colours are encoded by hex-codes
 - #RRGGBB or #RRGGBBAA
 - 0123456789abcdef
- Specify colours to avoid spurious colour matching (!!)
- Soft-code colour values throughout
- <https://gka.github.io/palettes>

The screenshot shows a four-step process for creating a color palette:

- 1** What kind of palette do you want to create?
Palette type: sequential diverging Number of colors: 9
- 2** Select and arrange input colors
00429d 96ffea fffffe0
- 3** Check and configure the resulting palette ✓ This palette is colorblind-safe.
checkboxes: correct lightness, bezier interpolation
simulate: normal deut. prot. trit.
A color bar showing a gradient from dark blue to light yellow-green.
- 4** Export the color codes in various formats
You can also save your palette for later by bookmarking this page using **ctrl + d**.
#00429d, #2e59a8, #4771b2, #5d8abd, #73a2c6, #8abccf, #a5d5d8, #c5eddf, #ffffe0

Colours

- Colours are encoded by hex-codes
 - #RRGGBB or #RRGGBBAA
 - 0123456789abcdef
- Specify colours to avoid spurious colour matching (!!)
- Soft-code colour values throughout
- <https://gka.github.io/palettes>
- <https://color.adobe.com/create/color-wheel>

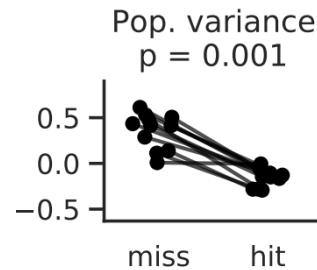


Separate topic(?): Designing figures

- For each panel: **Key takeaway = key visual effect**

Separate topic(?) : Designing figures

- For each panel: **Key takeaway = key visual effect**

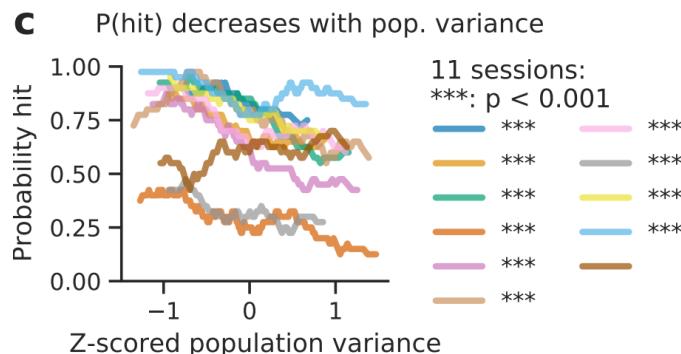


Key visual effect

Key takeaway

Miss > hit

Miss > hit



There are a lot
of lines

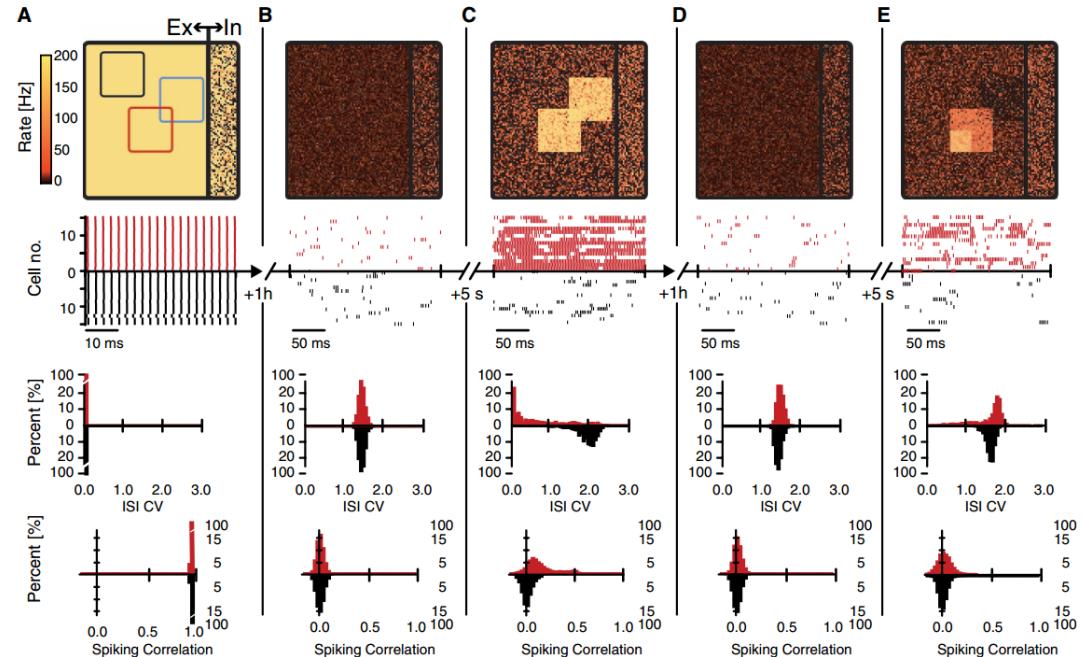
Most lines go
down

Separate topic(?): Designing figures

- For each panel: **Key takeaway = key visual effect**
 - Highlight small changes
 - Use visual aids or written text if needed

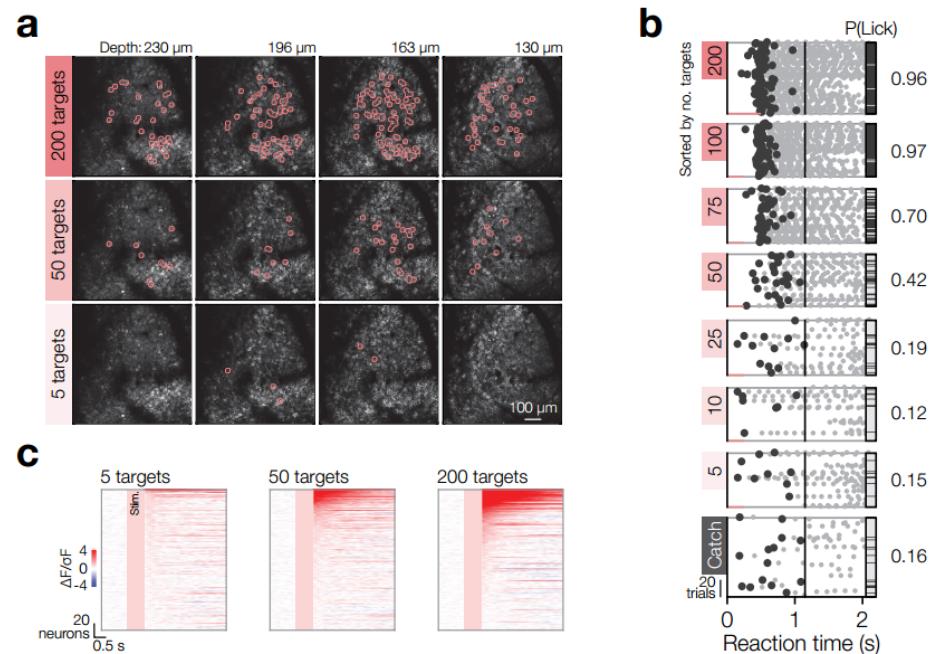
Separate topic(?) : Designing figures

- For each panel: **Key takeaway = key visual effect**
- For each figure: **Minimise number of elements**
 - Use ‘multiples’ in lay-out



Separate topic(?) : Designing figures

- For each panel: **Key takeaway = key visual effect**
- For each figure: **Minimise number of elements**
 - Use ‘multiples’ in lay-out
 - Maintain colour schemes as much as possible



Separate topic(?): Designing figures

- For each panel: **Key takeaway = key visual effect**
- For each figure: **Minimise number of elements**
 - Use ‘multiples’ in lay-out
 - Maintain colour schemes as much as possible
 - Eliminate duplicate elements (labels, ticklabels, colour bars etc)
 - Eliminate unnecessary elements (spines, grids, number of ticks)
 - Make sure there is enough whitespace

Publishing RFs

- Import pdfs automatically with Latex
- Github version of paper
- Elife ERA
- Nextjournal
- Other languages: R + Markdown, Julia + Pluto
- <https://www.nature.com/articles/s42005-020-00403-4>
- <https://www.nature.com/articles/d41586-021-01174-w>

Conclusions

- In short, I believe that RFs improve scientific quality, efficiency & impact of data visualisation, compared to MCFs.
- Code of all my figs is available on Github (for templates) & always happy to help
- Would like to get a ‘working document’ version of this presentation online at some point, let me know what you think!

Code repos – if you need access please just ask me!

- Today's talk & example figure: Packer lab drive and https://github.com/vdplasthijs/reproducible_figures
- Figures from Rowland, Van der Plas, Loidolt ea, bioRxiv: <https://github.com/Packer-Lab/popping-off>
- Figures from Van der Plas, Tubiana ea, bioRxiv: <https://github.com/vdplasthijs/zf-rbm>
- Figures from Van der Plas ea, PMLR: <https://github.com/vdplasthijs/rotation>

Acknowledgements

- Many encouraging discussions (over the years), in particular Adam Packer, Prajay Shah, Jimmy Rowland, Fons van der Plas
- Many coding mentors; including Fons, Remi Proville
- Some ideas come from online tutorials, books and stackoverflow. I always try to give credit where due in code (comments).
- Thank you for listening!