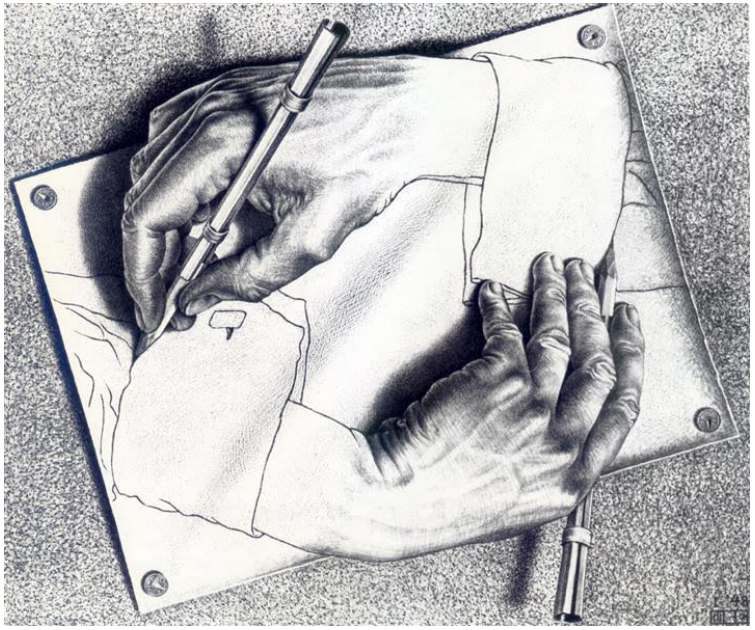
Why & how to make reproducible figures (in Python)

Thijs van der Plas,
lab meeting, 15 June 2022

vdplasthijs@gmail.com



Goal

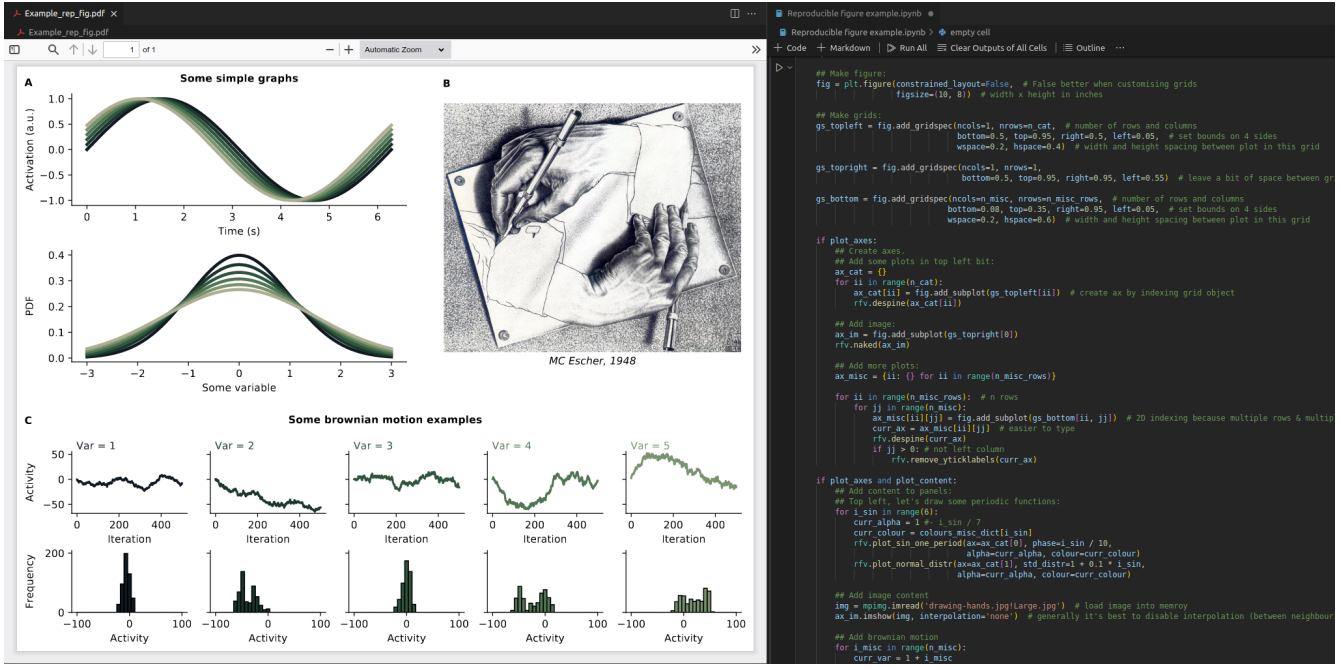
- Goal: provide ideas & resources, and make you excited to make reproducible figures!
- Not much code

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?*

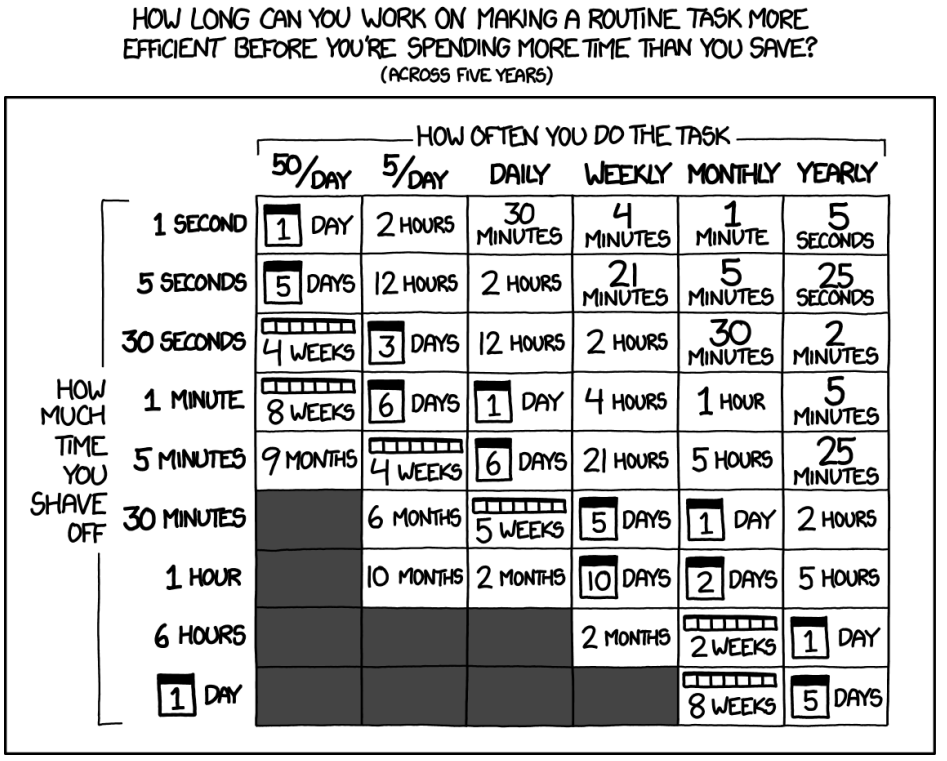
Why RFs?

- **Time-efficient**: Avoid repetition

  RFs avoid repetition.

  Repetition across:

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



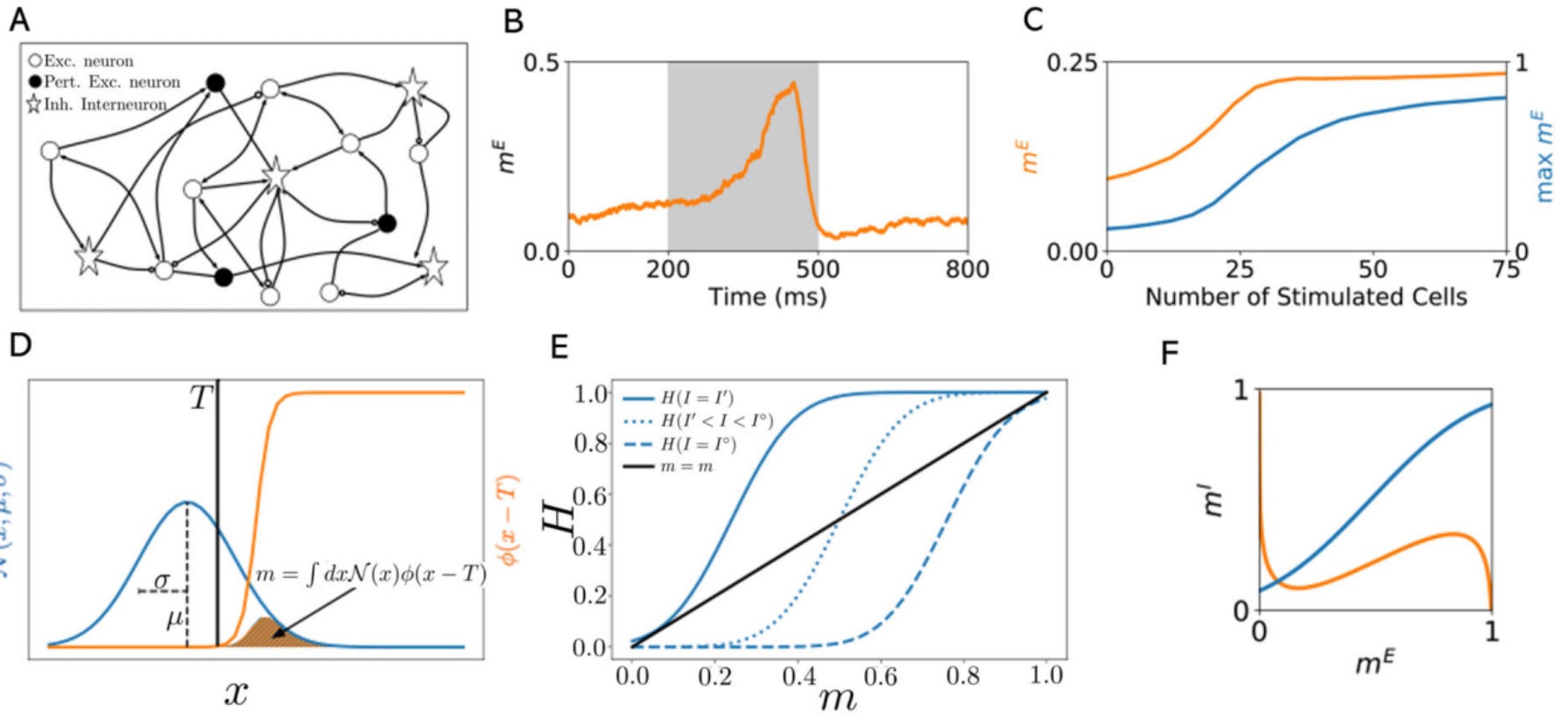
Why RFs?

- **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

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

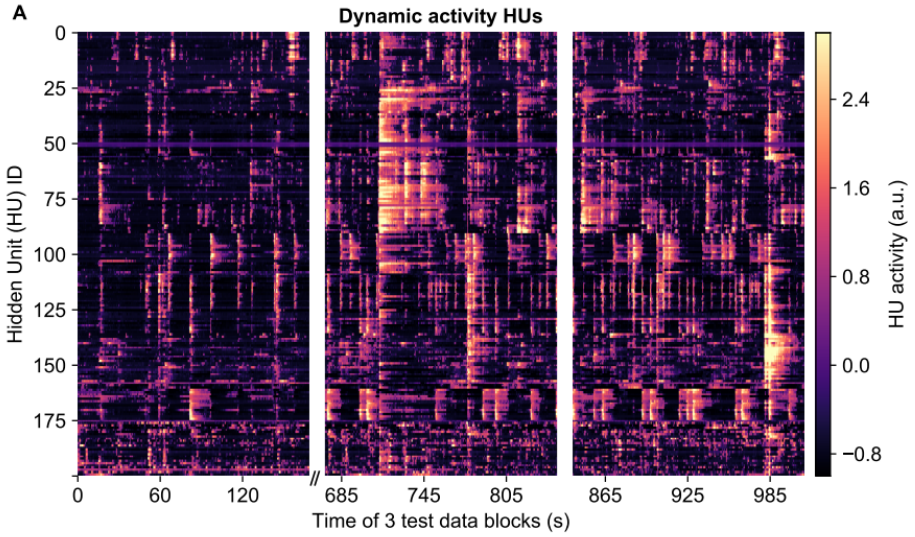
Why RFs?

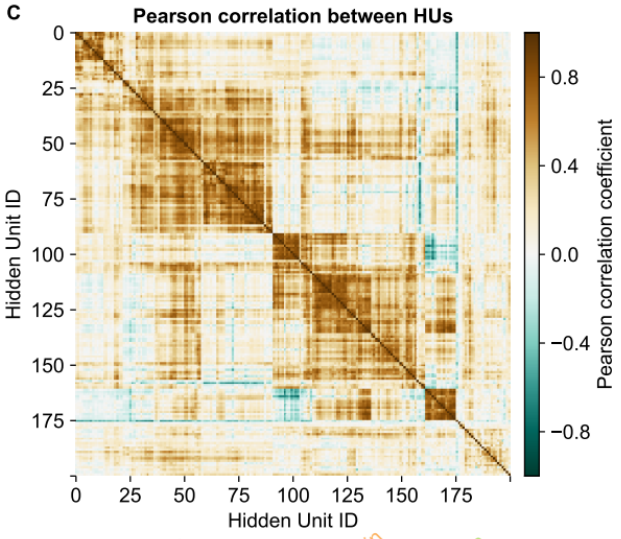
- **Time-efficient**: Avoid repetition
- Figures are **reproducible** & traceable
- **‘Good coding’** makes bugs less likely
- **Resource** for yourself, lab & community
- **Version control** of full figs (easy with for example pdf)
- Fun!! => great & **easy way to learn** better coding
  - Figure-coding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

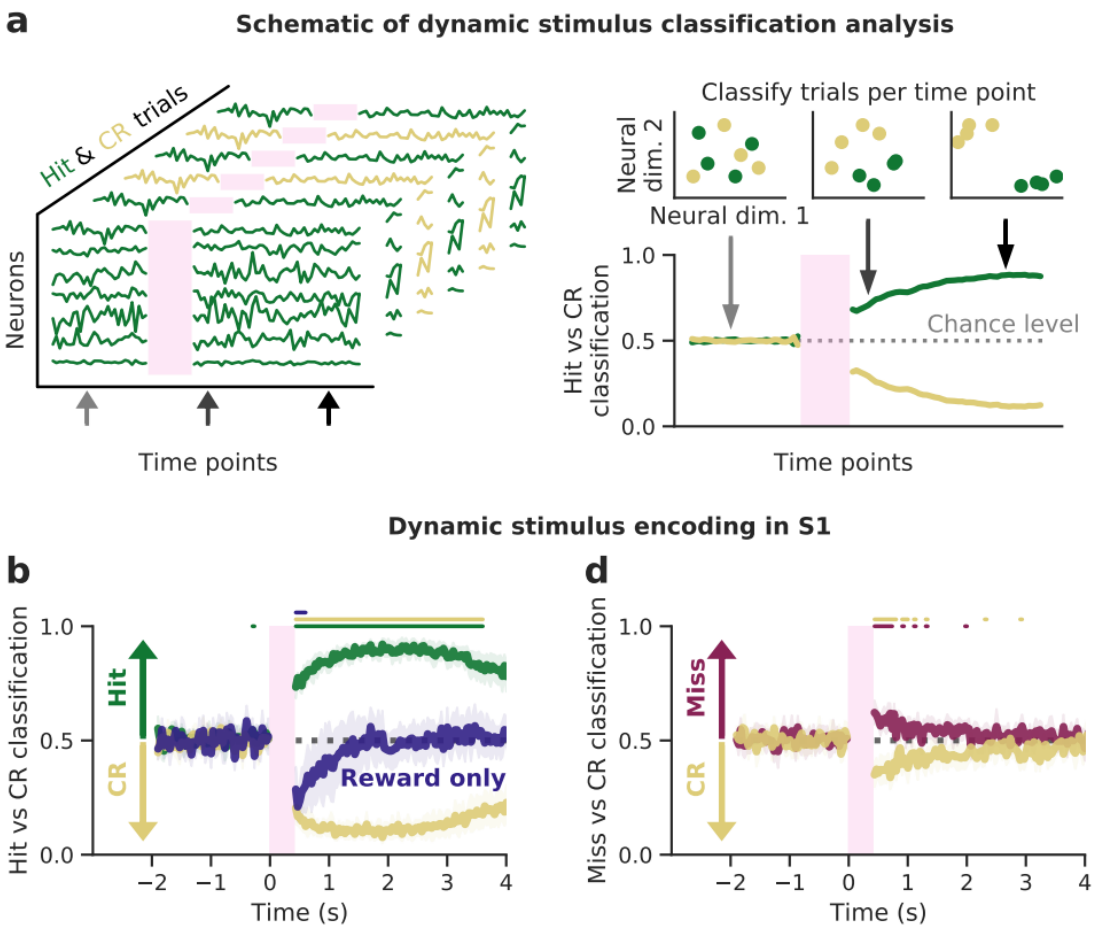
Example: across-panel alignment





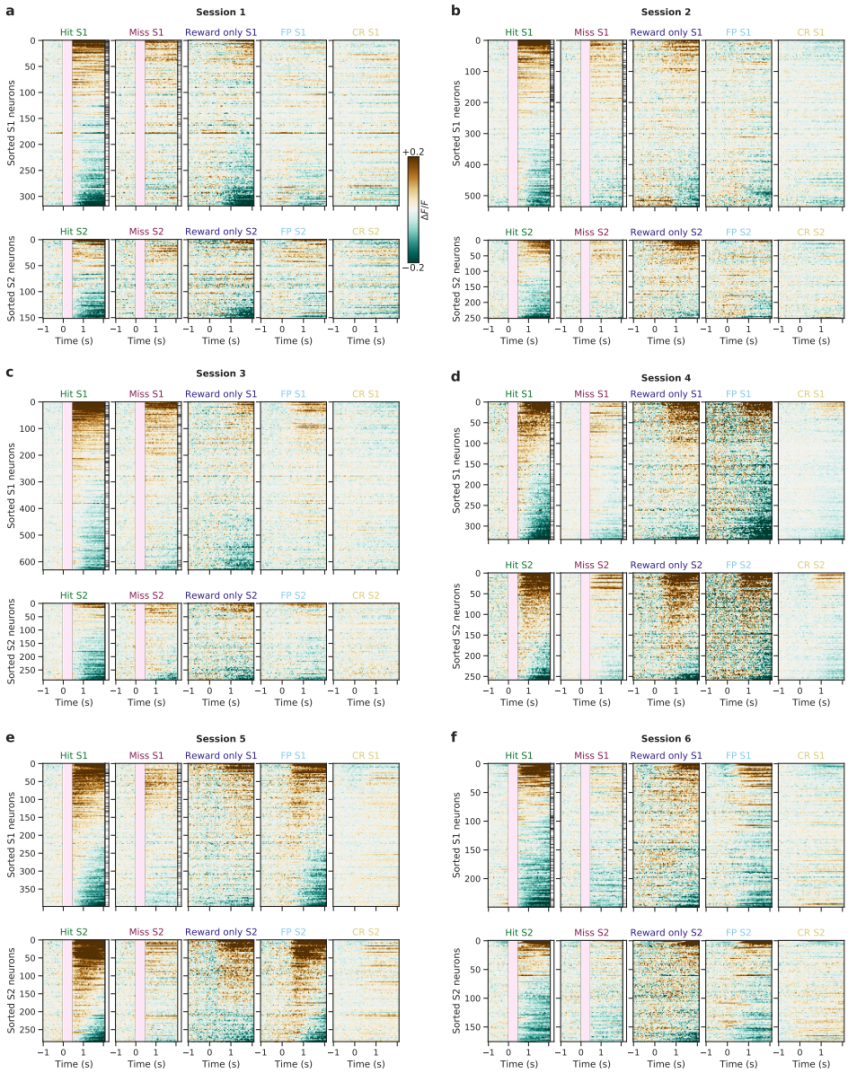
Van der Plas*, Tubiana* ea, 2021, bioRxiv

Example: consistent fig style with python-schematics



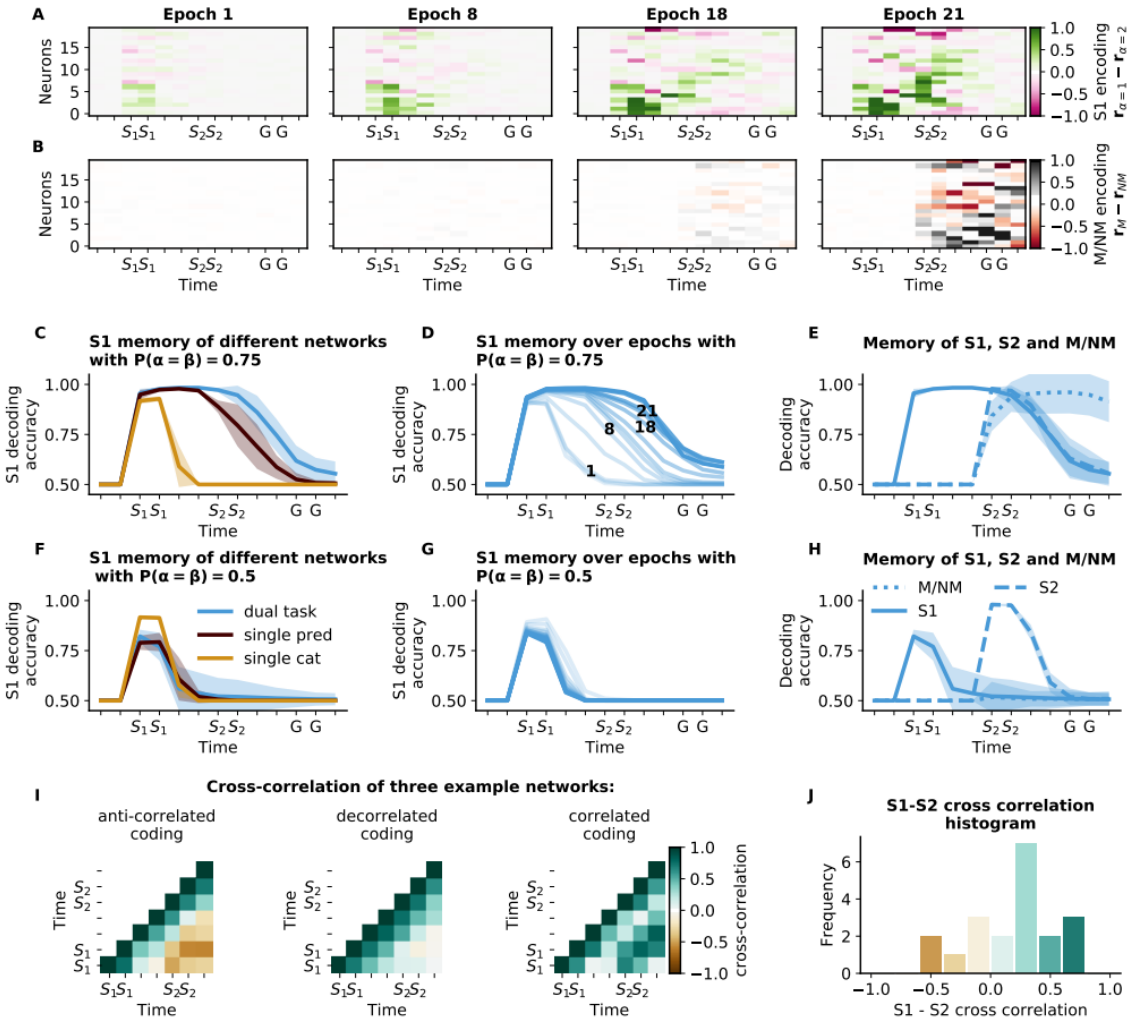
Rowland*, Van der Plas*, Loidolt* ea, 2021, bioRxiv

Example: easy to generate supplemental figures



Rowland*, Van der Plas*, Loidolt* ea, 2021, bioRxiv

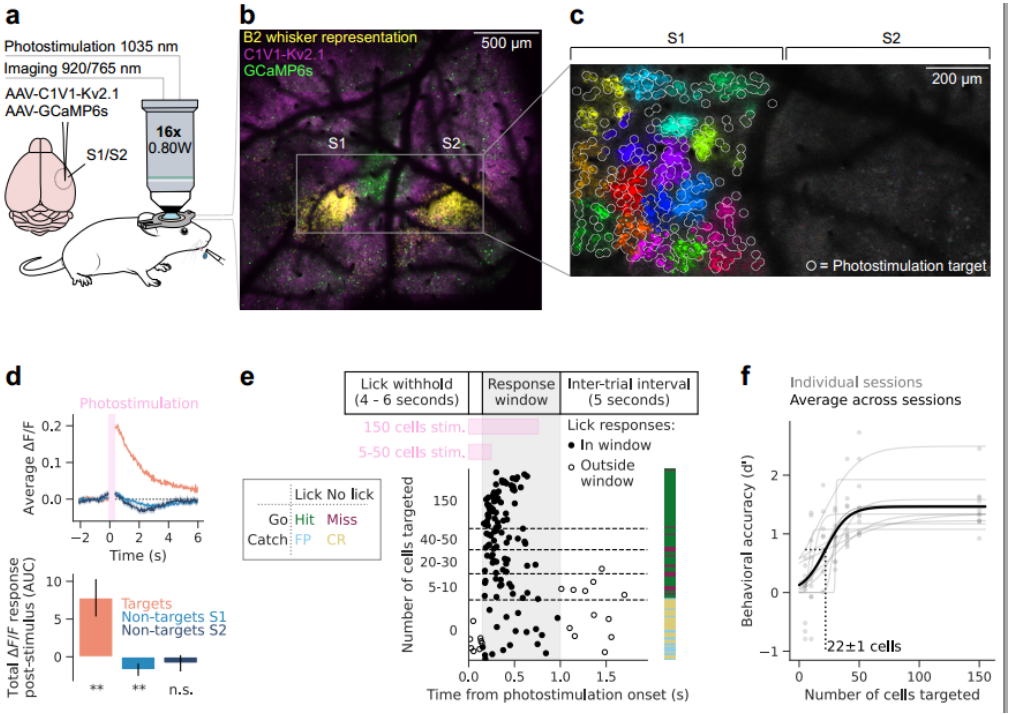
Example: easy to change highlighted examples



Van der Plas ea, 2022, PMLR

What about Illustrator?

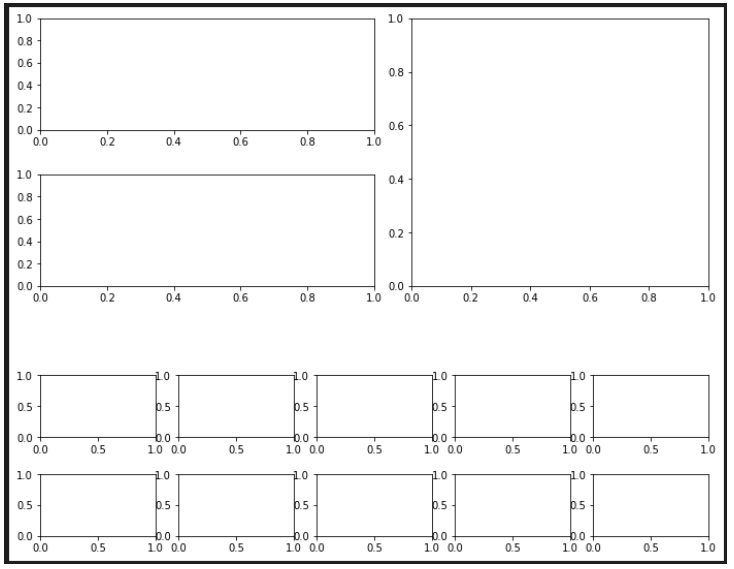
- Obvious use-cases exclusive to Illustrator
- These can be imported into code-generated RFs!
- Some elements *seem* easier in Illustrator (positioning text/ visual aids). But think about avoid-repetition efficiency!

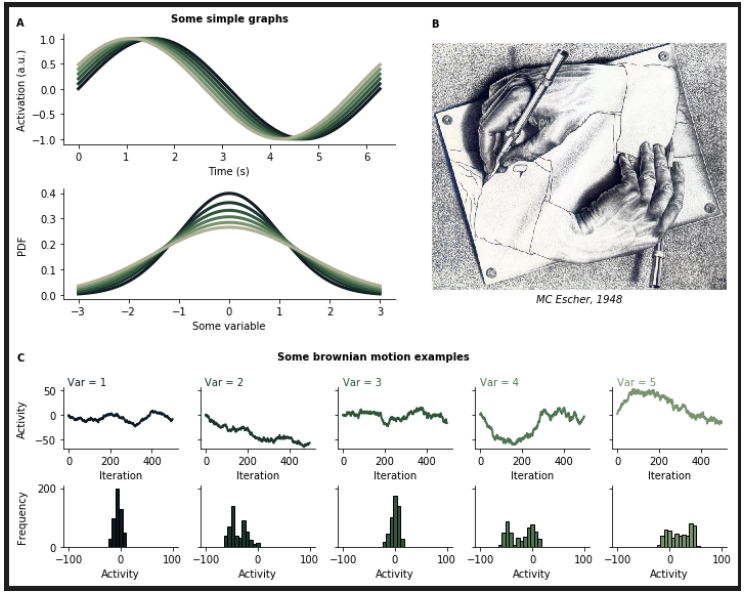


Rowland et al. Fig 1; top row contains illustrator-made elements that were imported into Python before generating full figure

4 principles to make RFs in Python

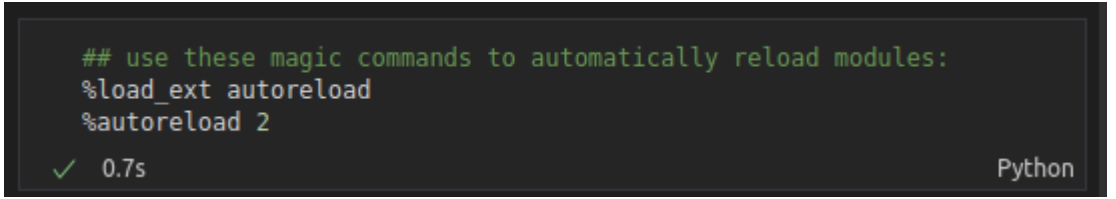
- Seperate panel *content* from panel *lay-out*
  - For content: functions that take 1) ax handle, 2) data, 3) parameters
  - For lay-out: use matplotlib's gridspec





4 principles to make RFs in Python

- Seperate 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



rep_fig_vis.py 3 | pop_off_plotting.py 9+

rep_fig_vis.py > plot_sin_one_period > colour

```python
############################
###   SPECIFIC FUNCTIONS FOR THIS TUTORIAL
############################

def plot_sin_one_period(ax=None, n_tp=500, phase=0, alpha=1, colour='k'):
    '''Create sine over 1 period with offset phase'''
    if ax is None:
        ax = plt.subplot(111)

    t_array = np.linspace(0, 2 * np.pi, n_tp)
    sin_array = np.sin(t_array + phase)

    ax.plot(t_array, sin_array, linewidth=3, alpha=alpha, c=colour)
    ax.set_xlabel('Time (s)')
    ax.set_ylabel('Activation (a.u.)')
    ax.set_title('Some simple graphs', y=1.05, fontdict={'weight': 'bold'})


def plot_normal_distr(ax=None, n_tp=500, mean_distr=0, std_distr=1, alpha=1, c
    '''Create sine over 1 period with offset phase'''
    if ax is None:
        ax = plt.subplot(111)

    t_array = np.linspace(-3, 3, n_tp)
    norm_array = scipy.stats.norm.pdf(t_array, loc=mean_distr, scale=std_distr

    ax.plot(t_array, norm_array, linewidth=3, alpha=alpha, c=colour)
    ax.set_xlabel('Some variable')
    ax.set_ylabel('PDF')

def plot_brown_proc(ax_trace=None, ax_hist=None, var=1, n_steps=500,
                    plot_ylabel=True, colour='k'):
    '''Sample brownian motion & plot trace and histogram'''
    gauss_samples = np.random.randn(n_steps) * np.sqrt(var)
    brown_motion = np.cumsum(gauss_samples)
    time_array = np.arange(n_steps)

    if ax_trace is None or ax_hist is None:
        fig, ax = plt.subplots(1, 2)
        ax_trace, ax_hist = ax

    ax_trace.plot(time_array, brown_motion, linewidth=2, c=colour)
    ax_trace.set_xlabel('Iteration')
    if plot_ylabel:
        ax_trace.set_ylabel('Activity')

    ax_hist.hist(brown_motion, bins=np.linspace(-100, 100, 30),
                 facecolor=colour, edgecolor='k', linewidth=1)
    ax_hist.set_xlabel('Activity')
    if plot_ylabel:
        ax_hist.set_ylabel('Frequency')
```

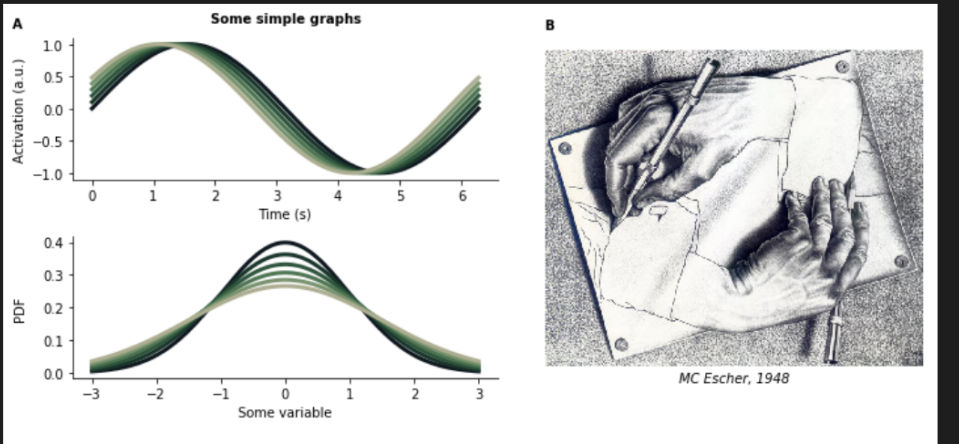
Reproducible figure example.ipynb

Reproducible figure example.ipynb > empty cell

+ Code + Markdown | Run All | Clear Outputs of All Cells | Outline ··· py37 (Python 3.7.0)

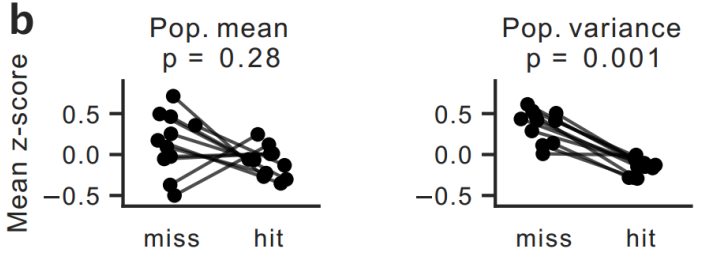
```python
                                    xycoords='axes fraction', c=colours_misc_dict[i_misc])
        if plot_ax_tidy:
            for i_row in range(n_misc_rows):
                rfv.equal_lims_n_axs(ax_list=list(ax_misc[i_row].values()))

            fig.align_ylabels(axs=[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', x=-1.29, y=1.25,  # specify coords in data coord system of this ax
                       fontdict={'weight': 'bold'})
        ax_cat[0].text(s='C', x=-1.29, y=-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 specificy 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')
```

[20] ✓ 1.2s Python



4 principles to make RFs in Python

- Seperate 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: seperate data selection & plotting => pure function



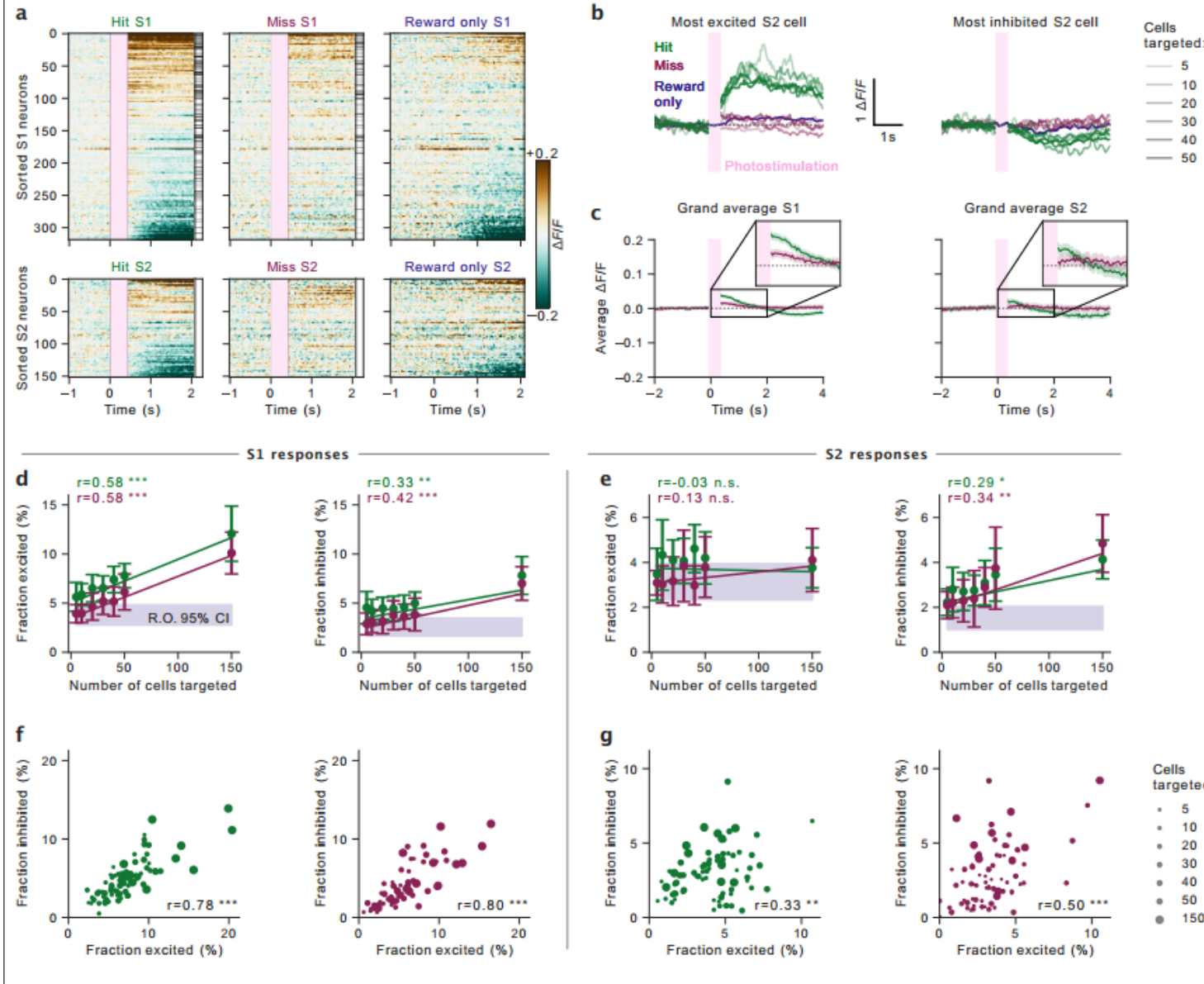
4 principles to make RFs in Python

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

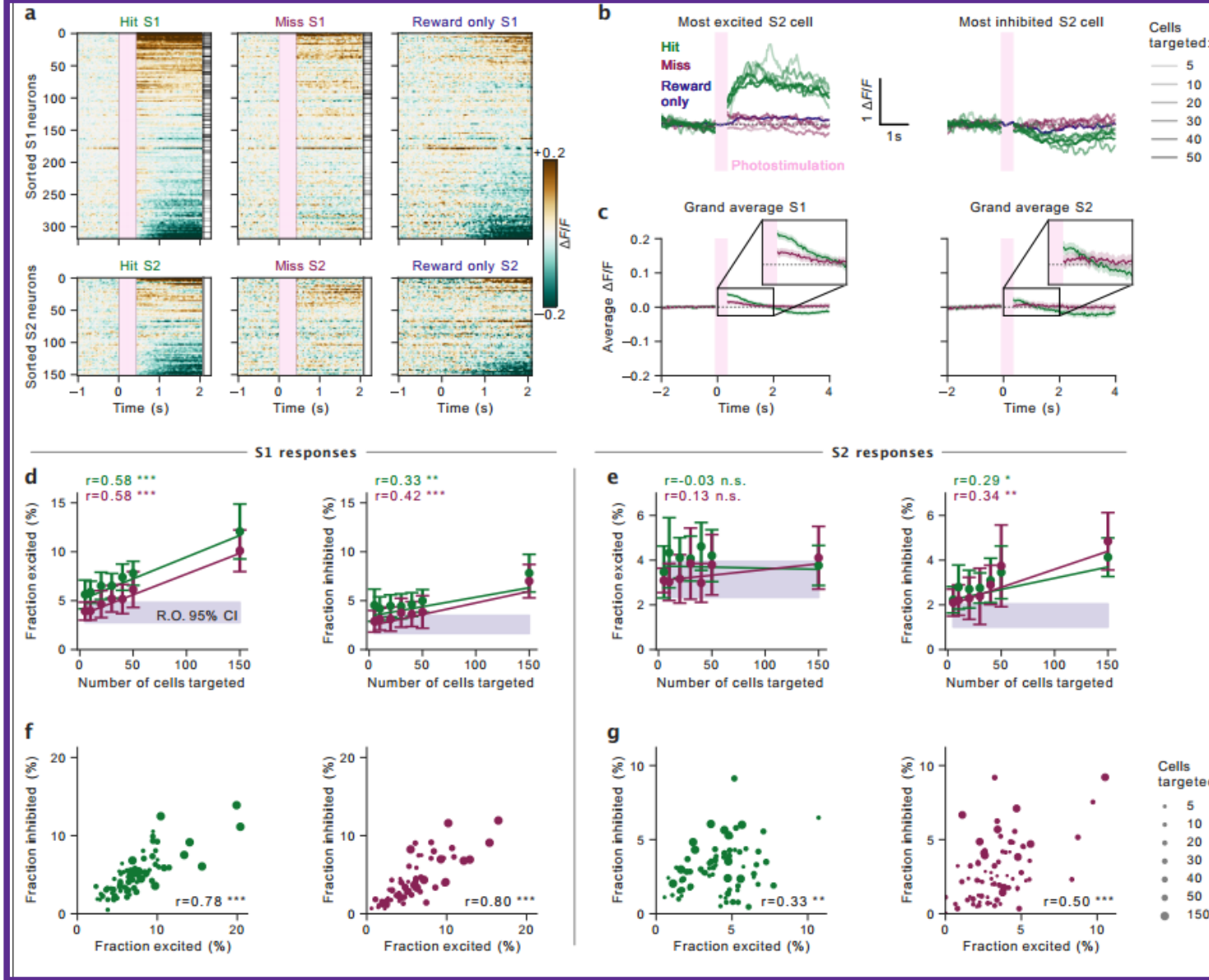
Example

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



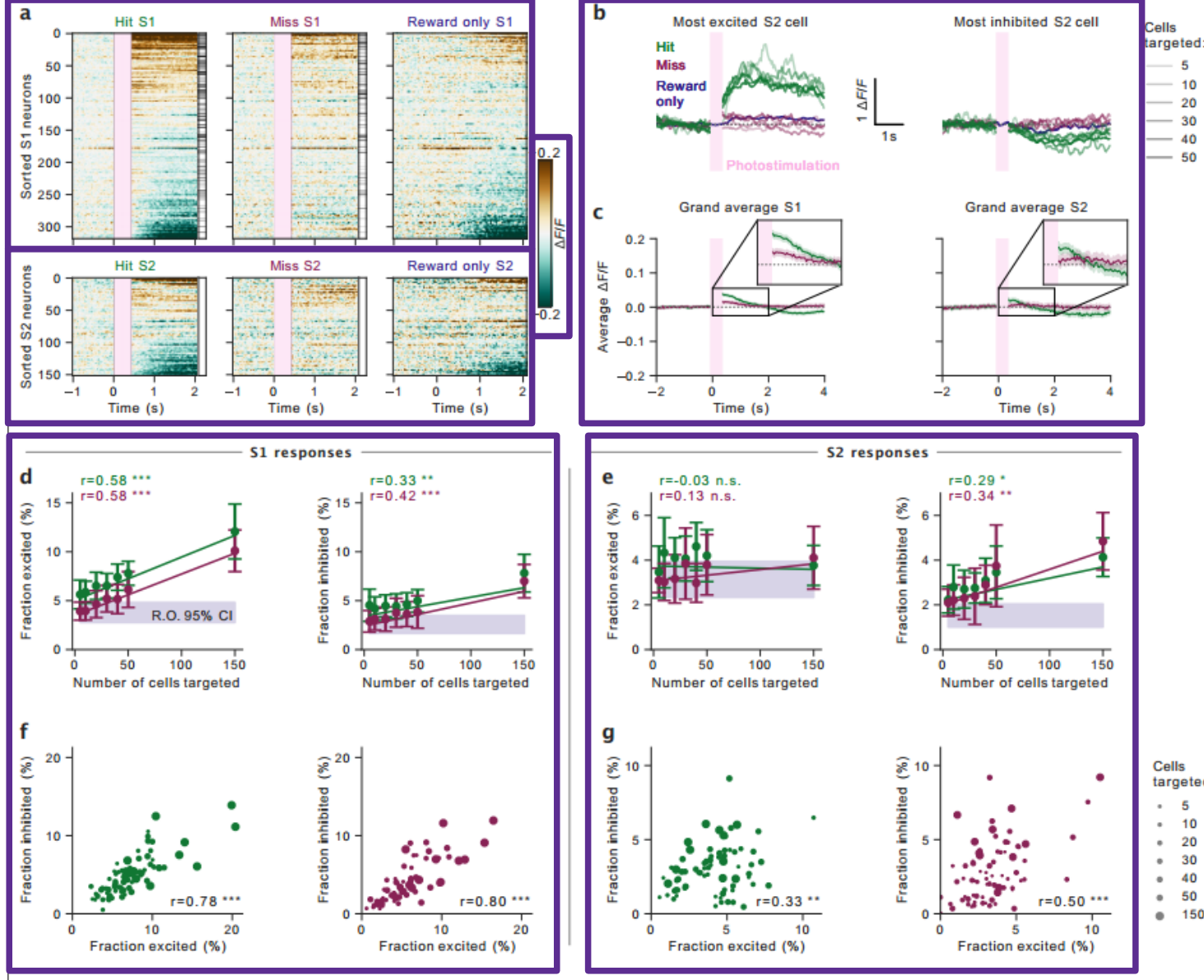
Example

- Figure



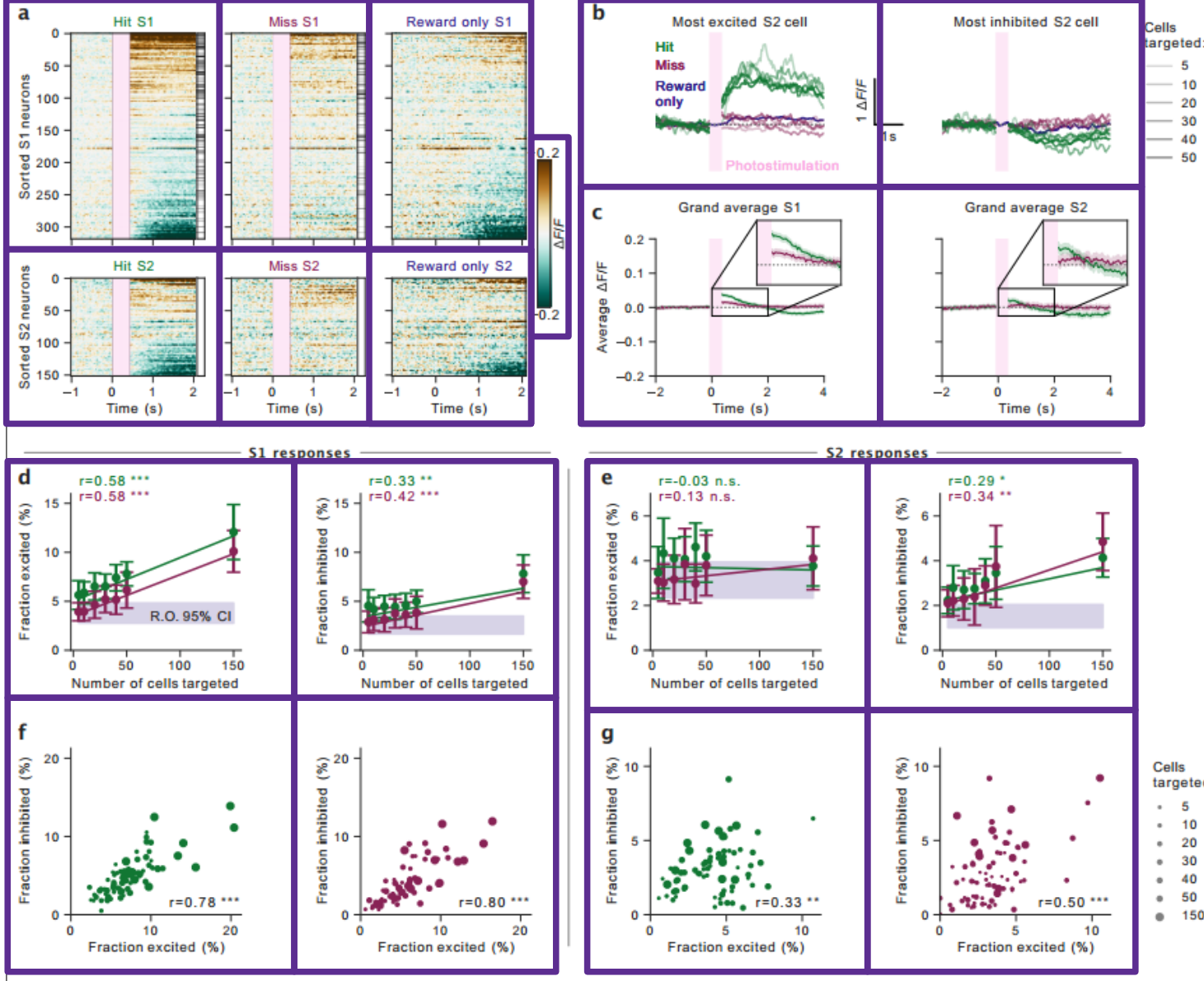
Example

- Figure
- Grids



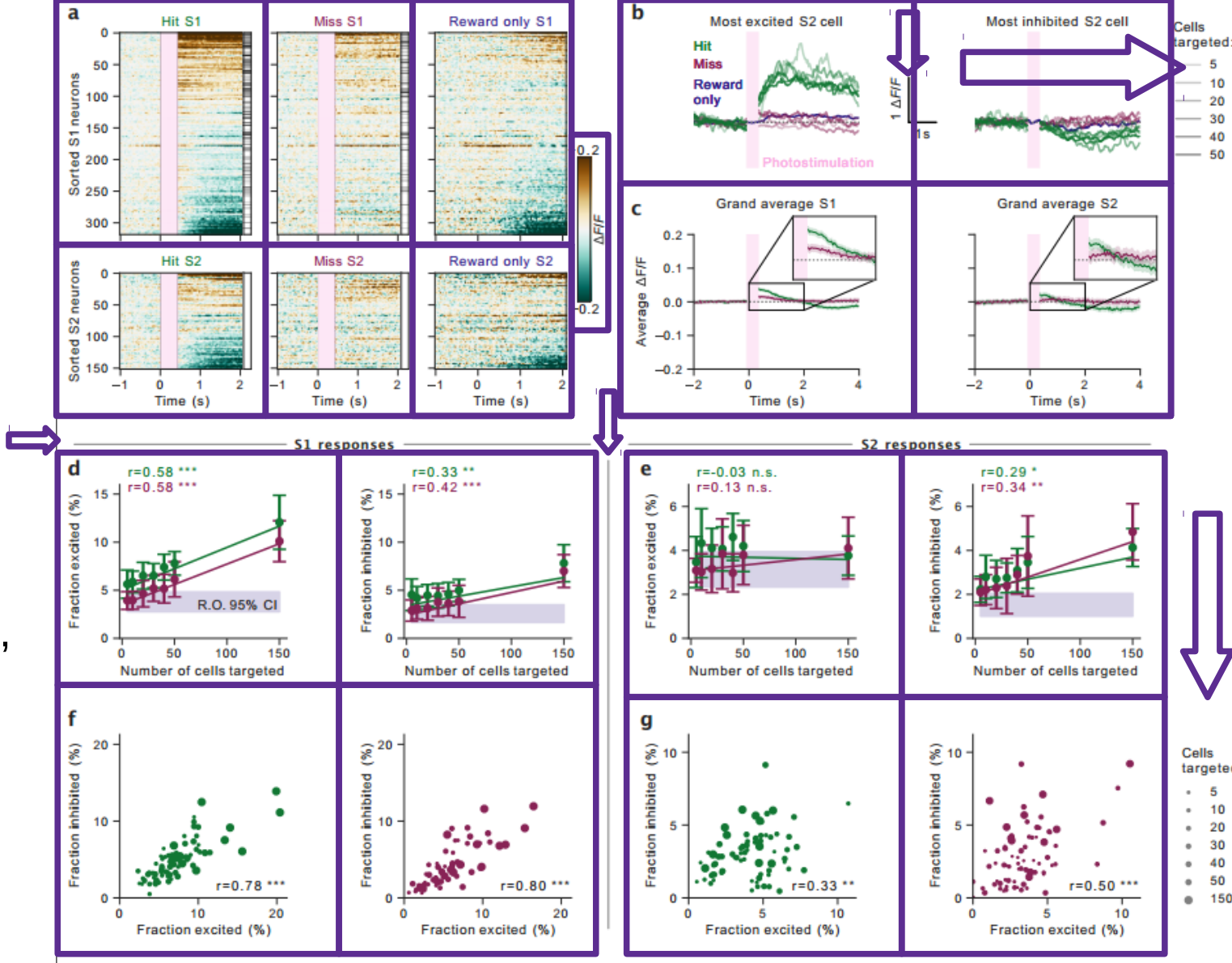
Example

- Figure
- Grids
- Axes



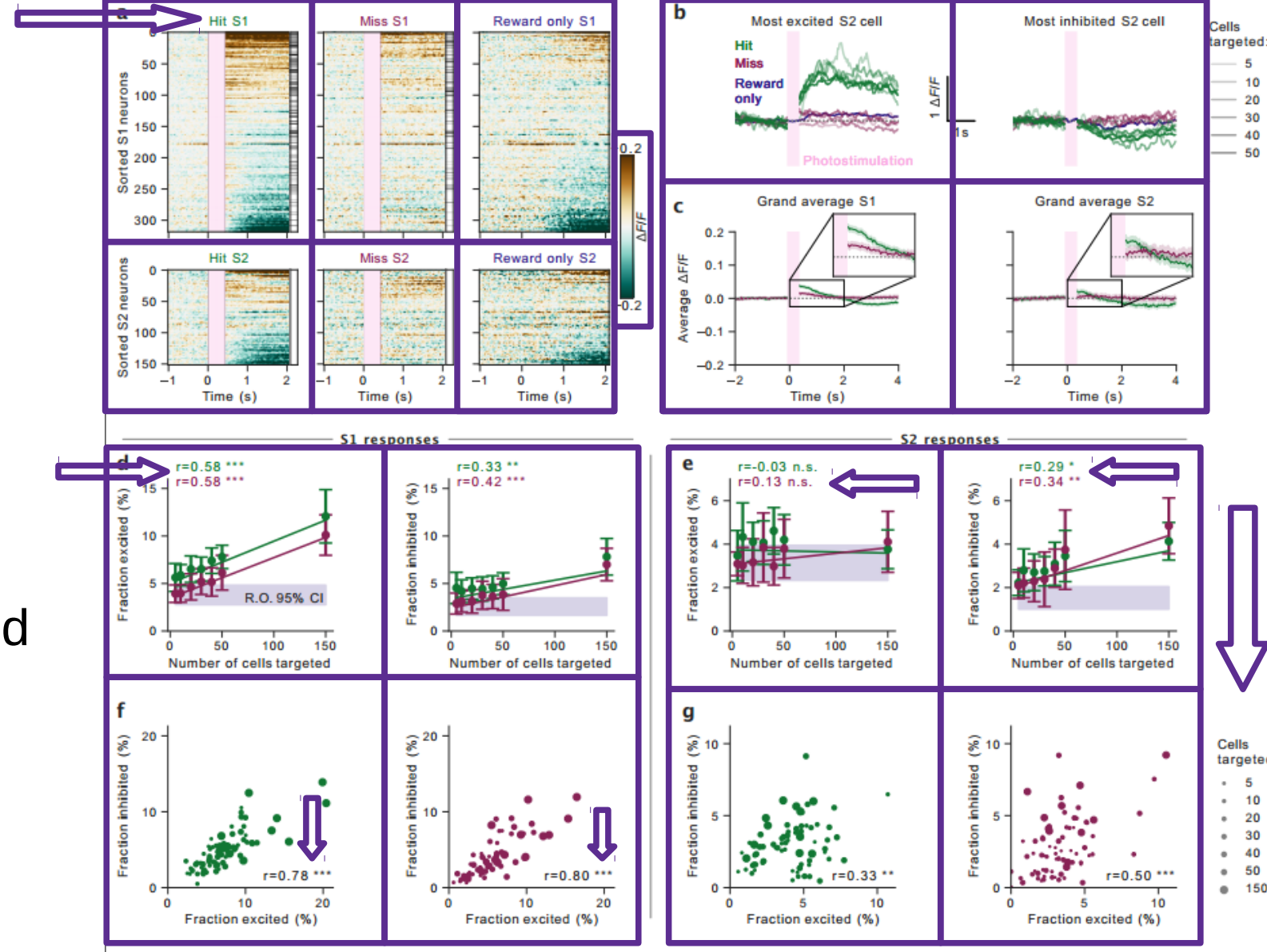
Example

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



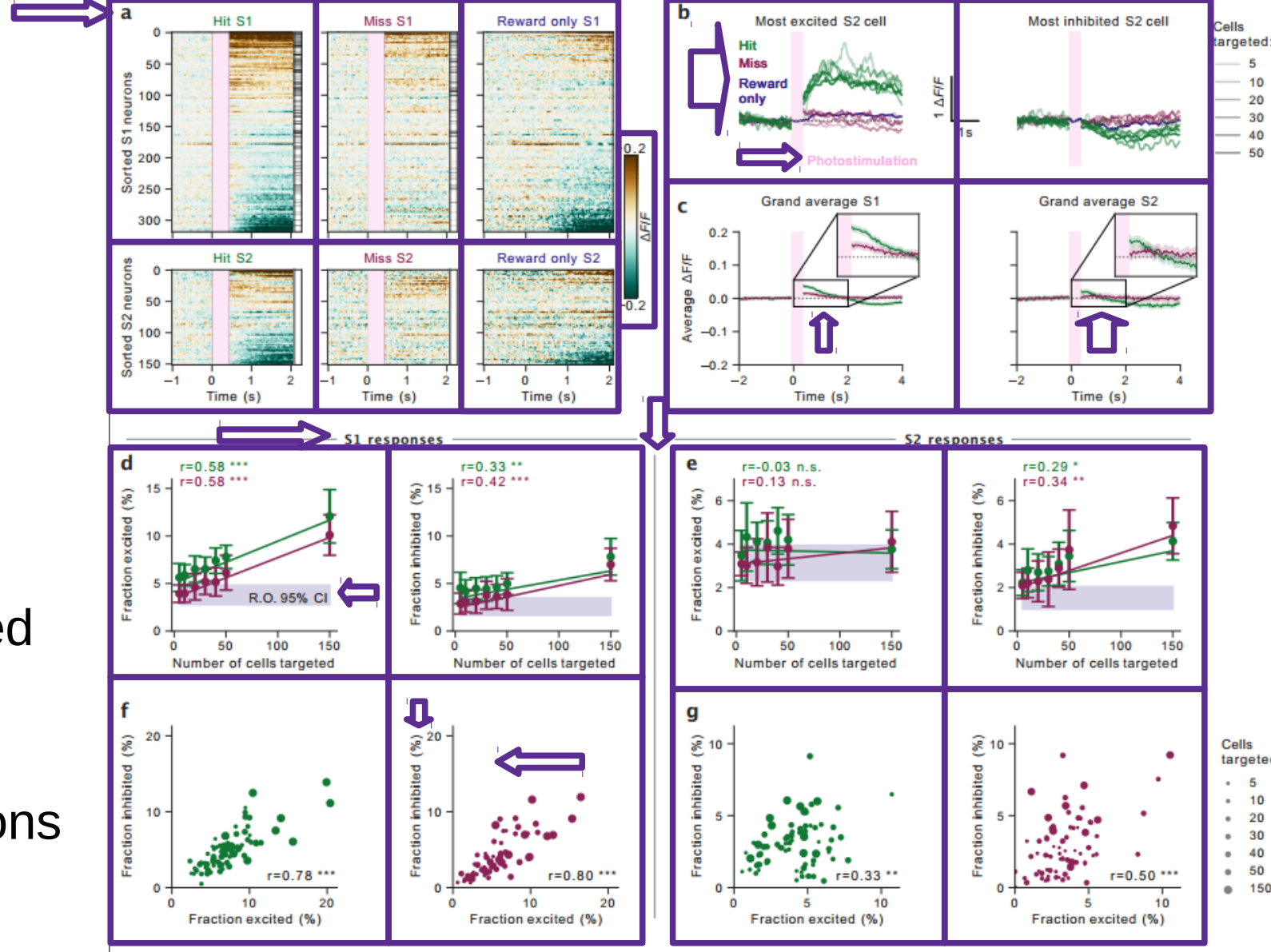
Example

- 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

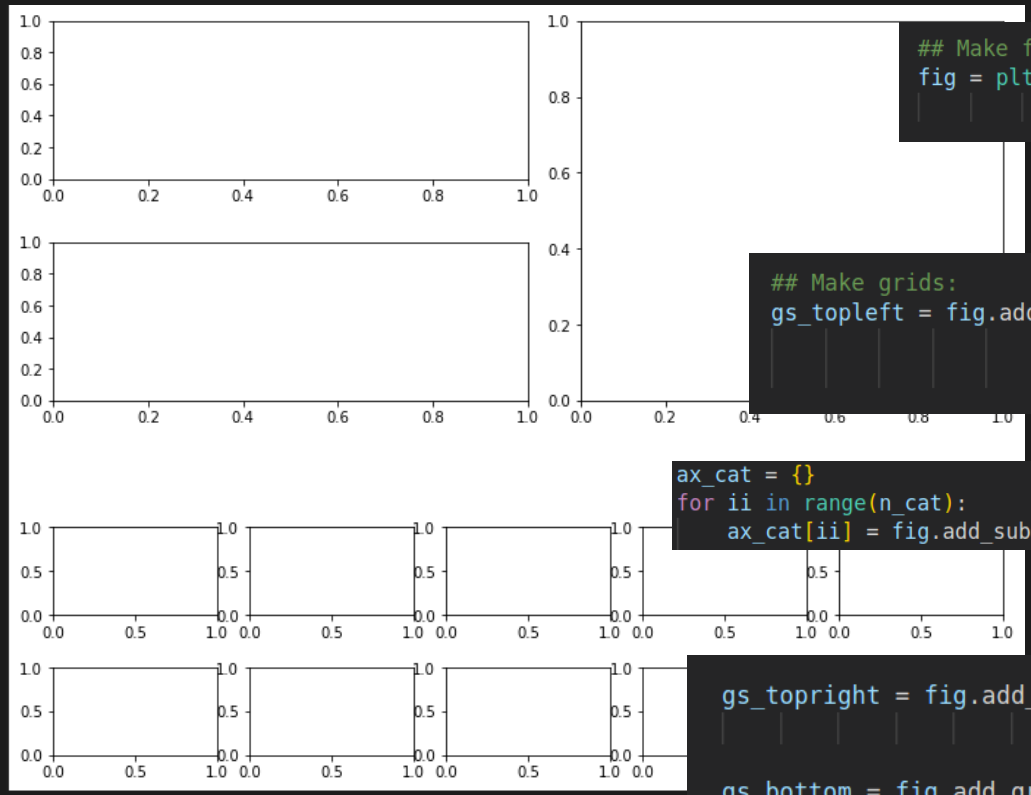


Example code

- 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



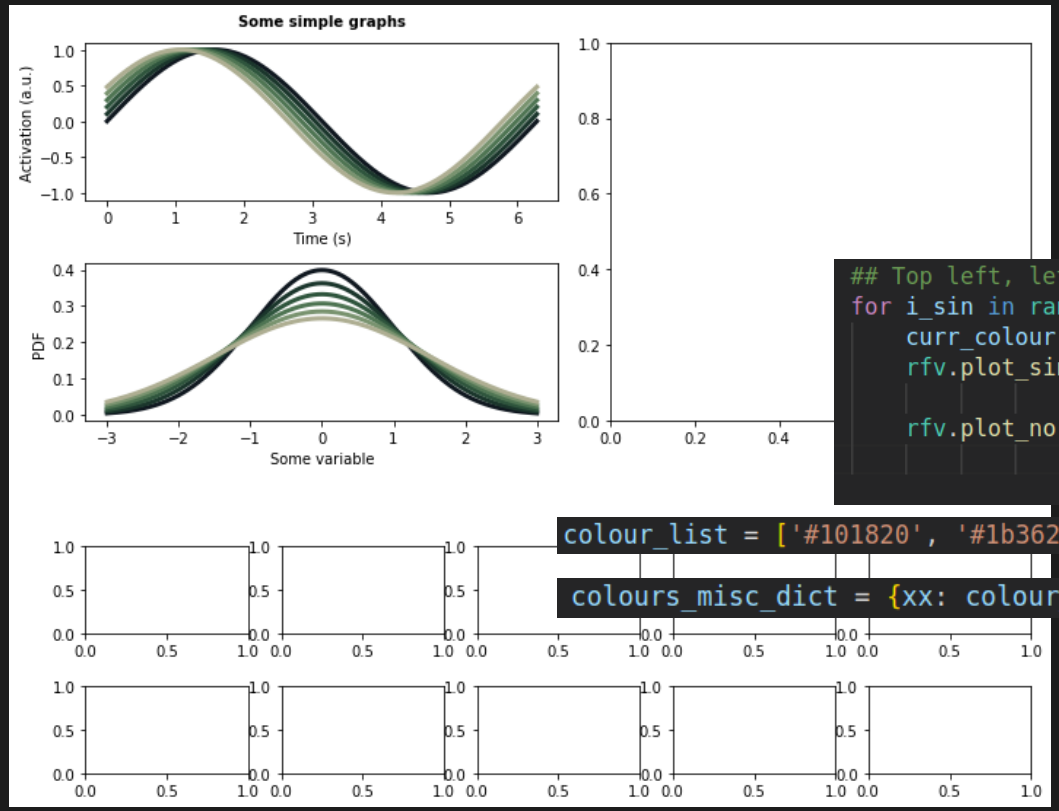
```
## Make figure:
fig = plt.figure(constrained_layout=False,  # False better when customising grids
                 figsize=(10, 8))  # width x height in inches
```

```
## Make grids:
gs_topleft = fig.add_gridspec(ncols=1, nrows=n_cat,  # number of rows and columns
                              bottom=0.5, top=0.95, right=0.5, left=0.05,  # set bounds
                              wspace=0.2, hspace=0.4)  # width and height spacing betwee
```

```
ax_cat = {}
for ii in range(n_cat):
    ax_cat[ii] = fig.add_subplot(gs_topleft[ii])  # create ax by indexing grid object
```

```
gs_topright = fig.add_gridspec(ncols=1, nrows=1,
                               bottom=0.5, top=0.95, right=0.95, left=0.55)  # l

gs_bottom = fig.add_gridspec(ncols=n_misc, nrows=n_misc_rows,  # number of rows
                             bottom=0.08, top=0.35, right=0.95, left=0.05,  # set
                             wspace=0.2, hspace=0.6)  # width and height spacing
```

Example code – loop for soft-coded colours

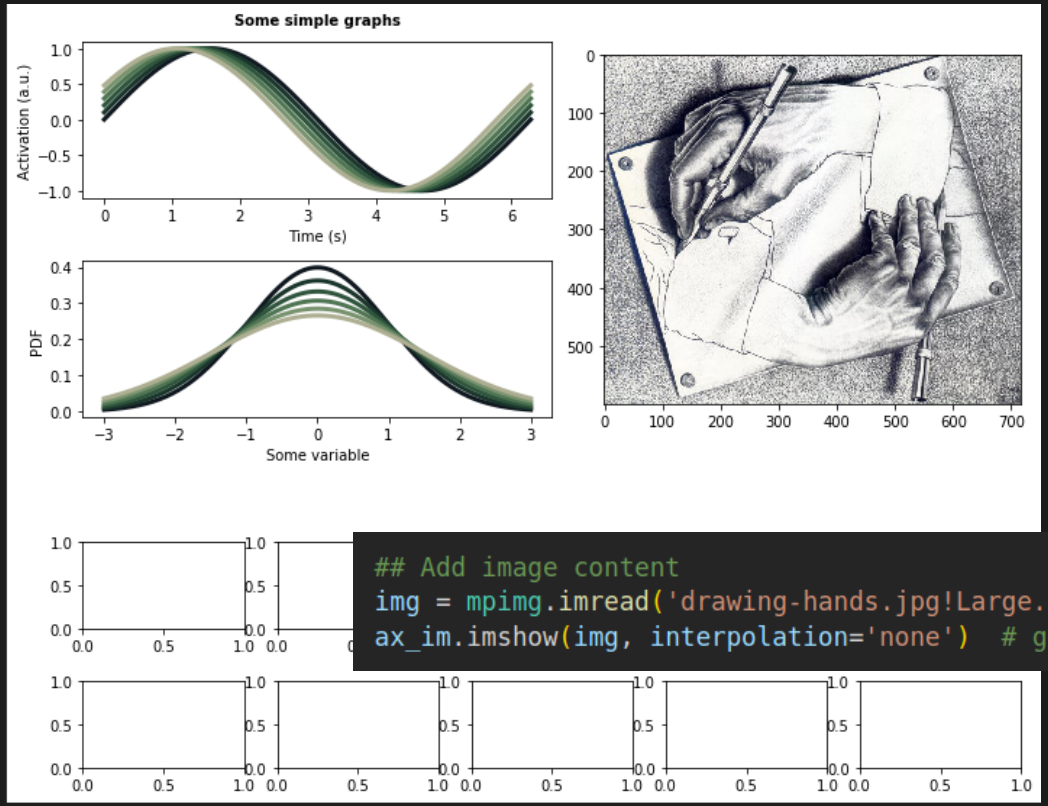



```
## Top left, let's draw some periodic functions:
for i_sin in range(6):
    curr_colour = colours_misc_dict[i_sin]
    rfv.plot_sin_one_period(ax=ax_cat[0], phase=i_sin / 10,
                            colour=curr_colour)
    rfv.plot_normal_distr(ax=ax_cat[1], std_distr=1 + 0.1 * i_sin,
                          colour=curr_colour)
```

```
colour_list = ['#101820', '#1b362c', '#2f553d', '#4f7553', '#79936f', '#aeae92']
```

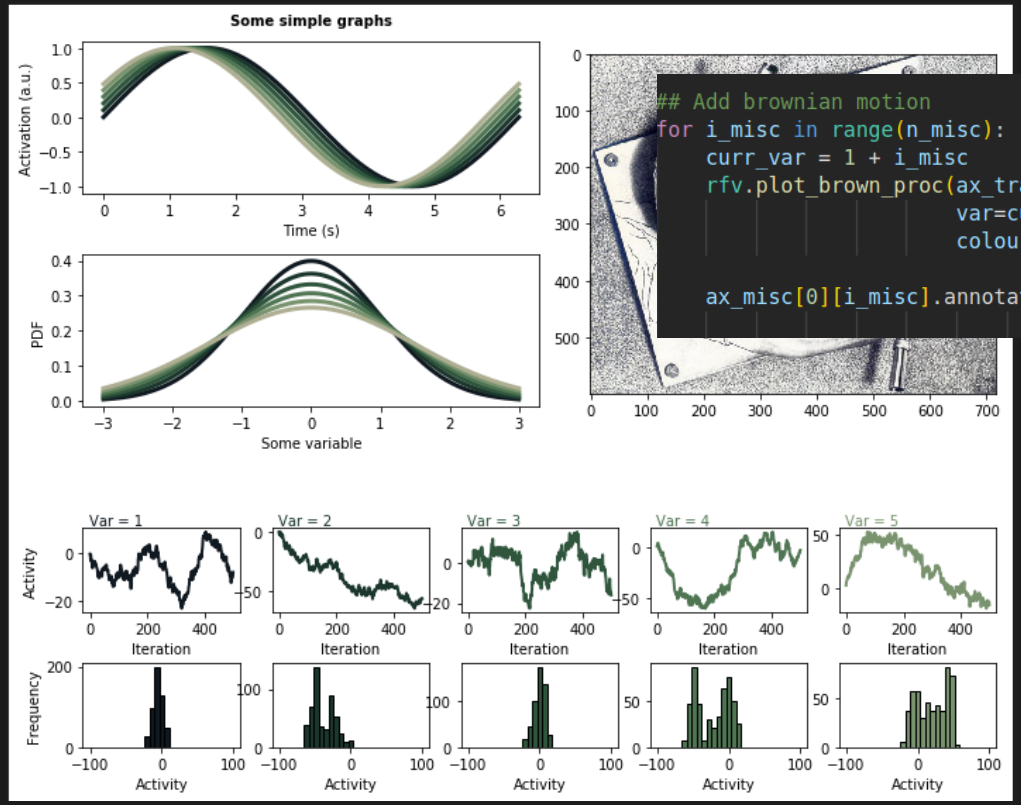
```
colours_misc_dict = {xx: colour_list[xx] for xx in range(len(colour_list))}
```

Example code – insert jpg/png images



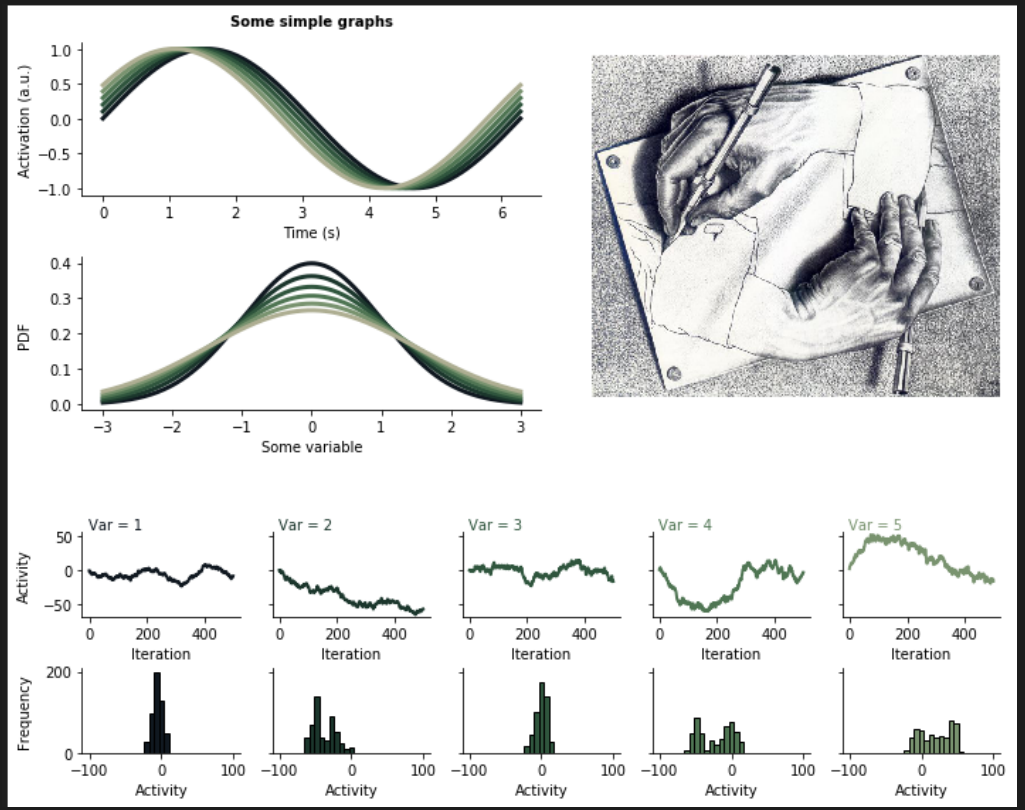

```
## Add image content
img = mpimg.imread('drawing-hands.jpg!Large.jpg')  # load image into memroy
ax_im.imshow(img, interpolation='none')  # generally it's best to disable interpolation
```

Example code – more loop use cases




```python
## Add brownian motion
for i_misc in range(n_misc):
    curr_var = 1 + i_misc
    rfv.plot_brown_proc(ax_trace=ax_misc[0][i_misc], ax_hist=ax_misc[1][i_misc],
                        var=curr_var,
                        colour=colours_misc_dict[i_misc], plot_ylabel=(i_misc == 0))

    ax_misc[0][i_misc].annotate(s=f'Var = {curr_var}', xy=(0.04, 1), va='bottom',
                                xycoords='axes fraction', c=colours_misc_dict[i_misc])
```

Example code – tidy up ax spines & labels

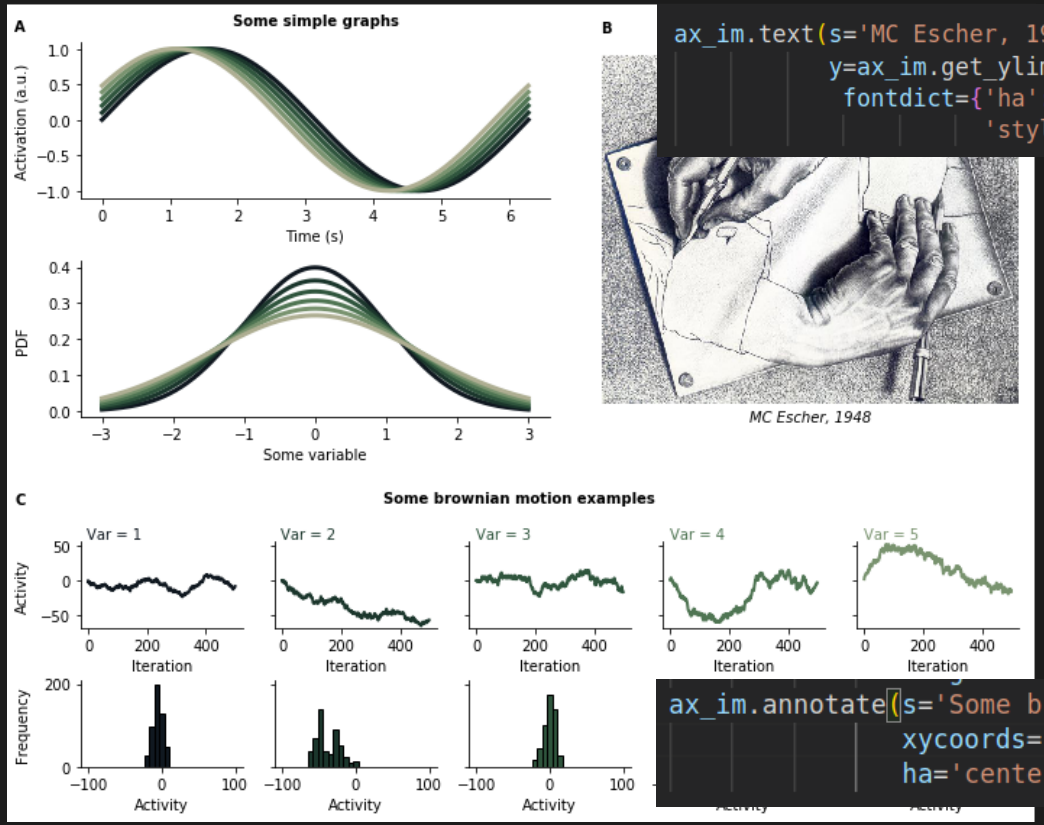



```
rfv.despine(ax_cat[ii])
```

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

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

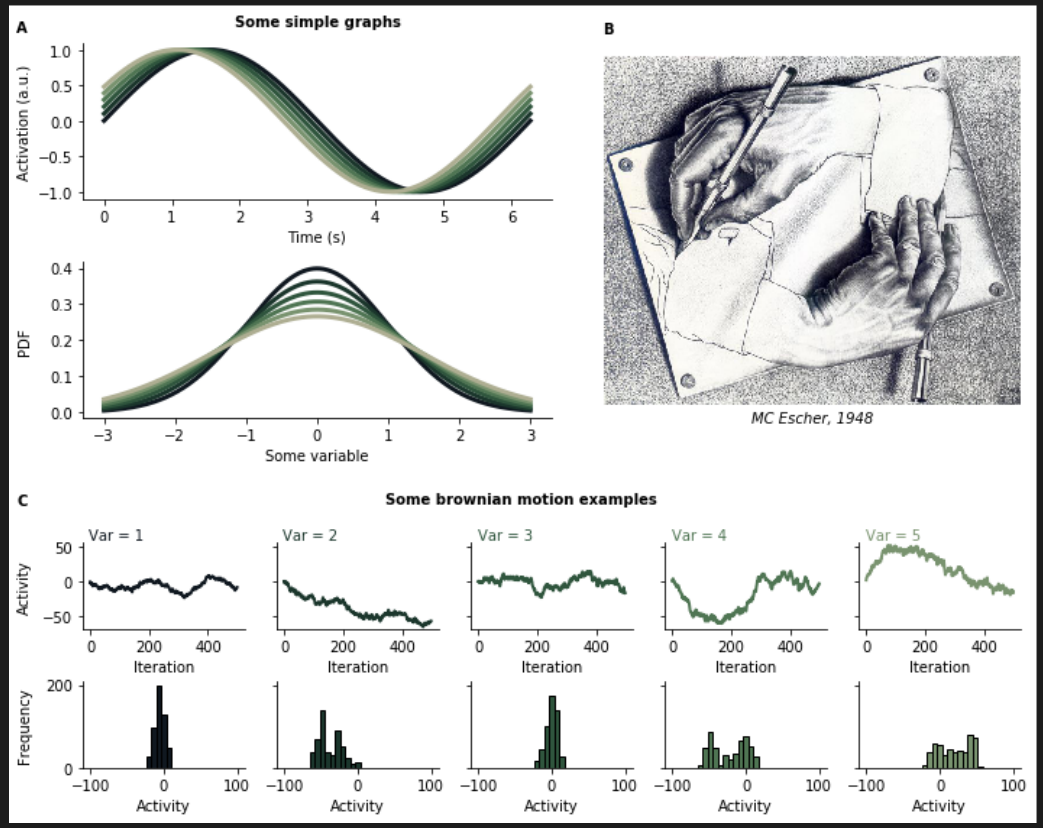
Example code – add text elements




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

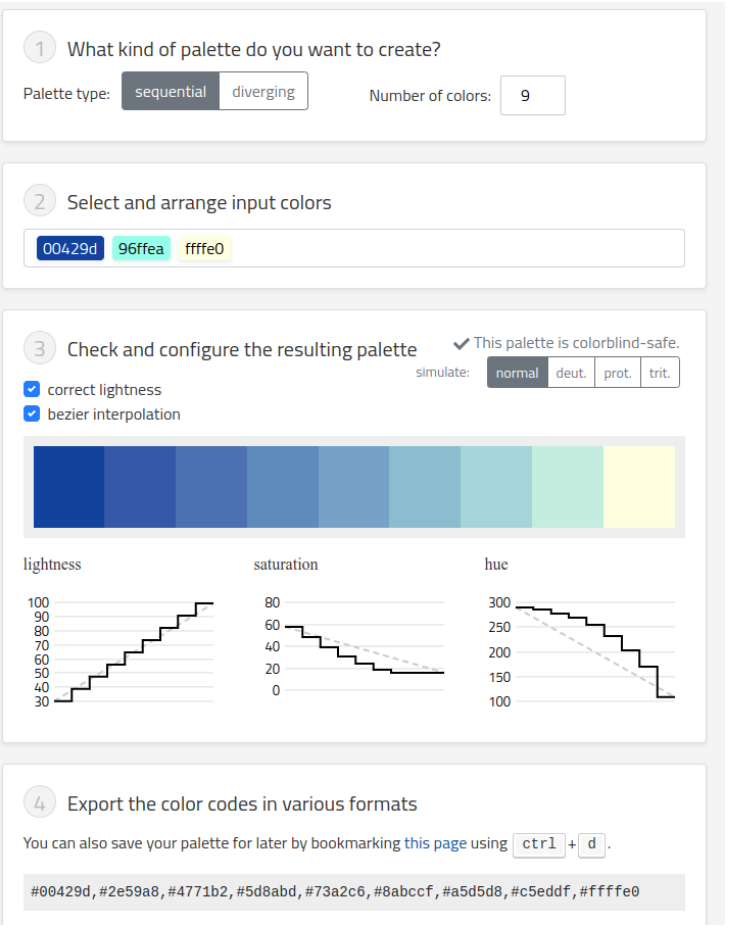
  numpydoc.readthedocs.io/en/latest/format.html

Colours

Moonrise Kingdom, 2012 Colour Palette by Wes Anderson Palettes

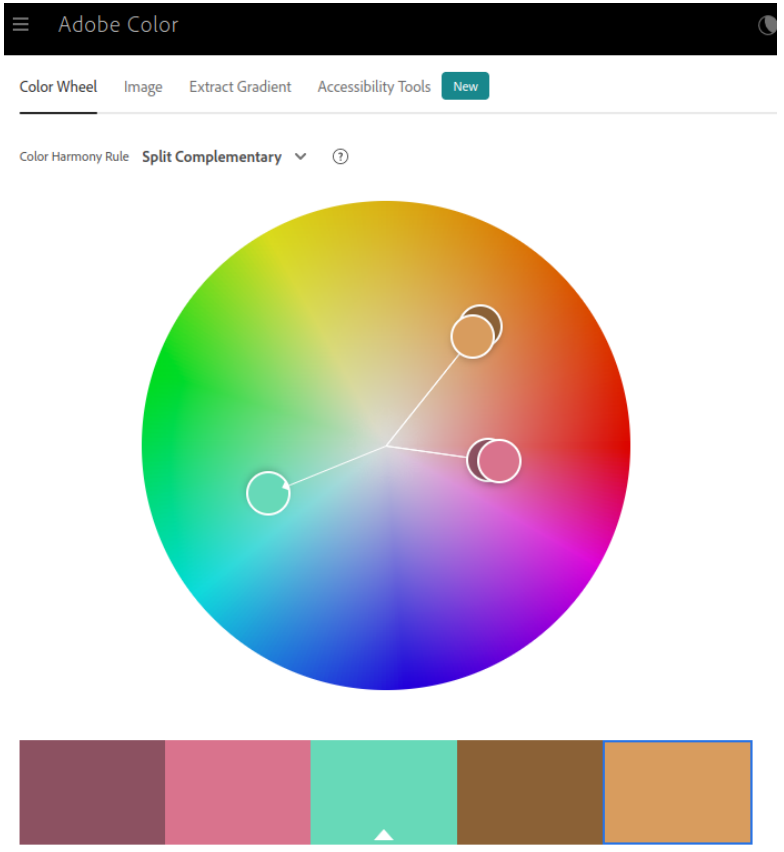
Colours

- 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



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

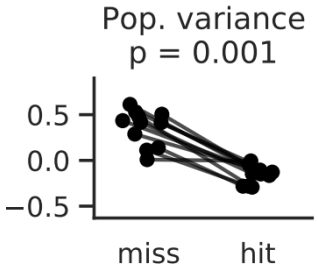


Separate topic(?): Designing figures

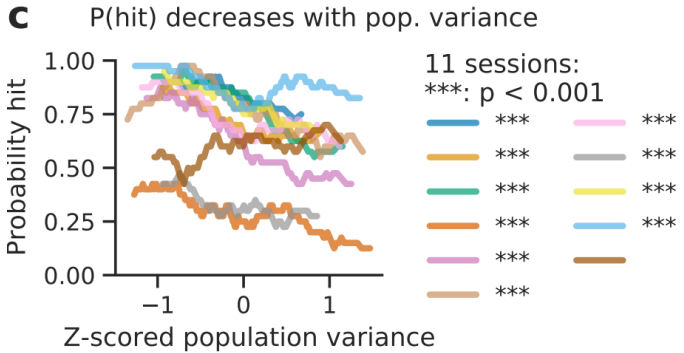
- For each panel: **Key takeaway = key visual effect**

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

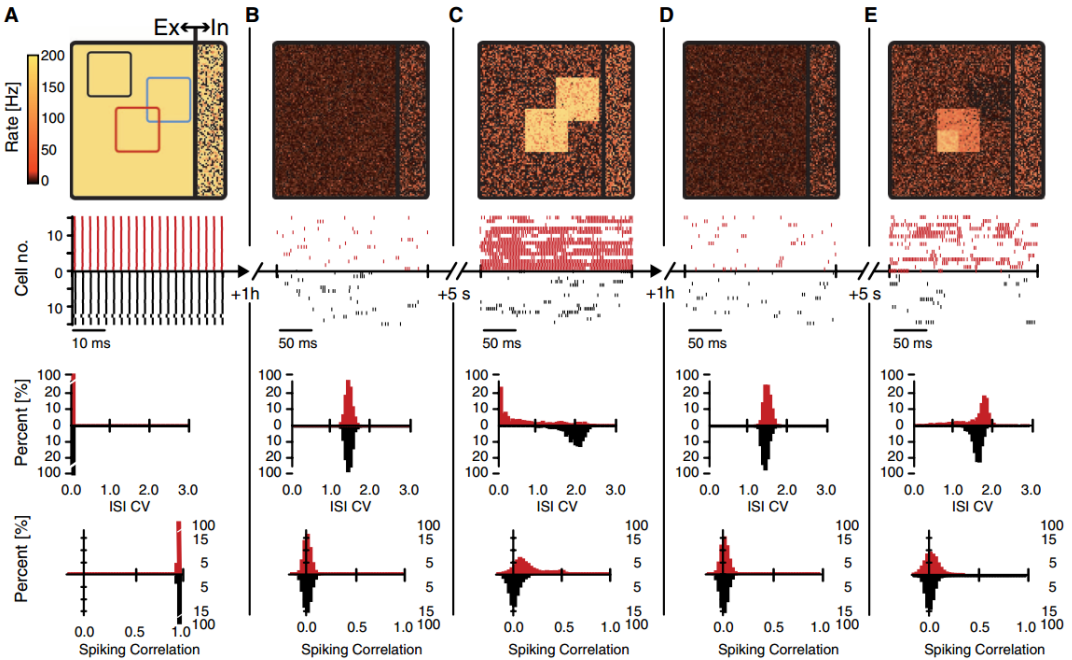
Rowland, Van der Plas, Loidolt ea, 2021 bioRxiv

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

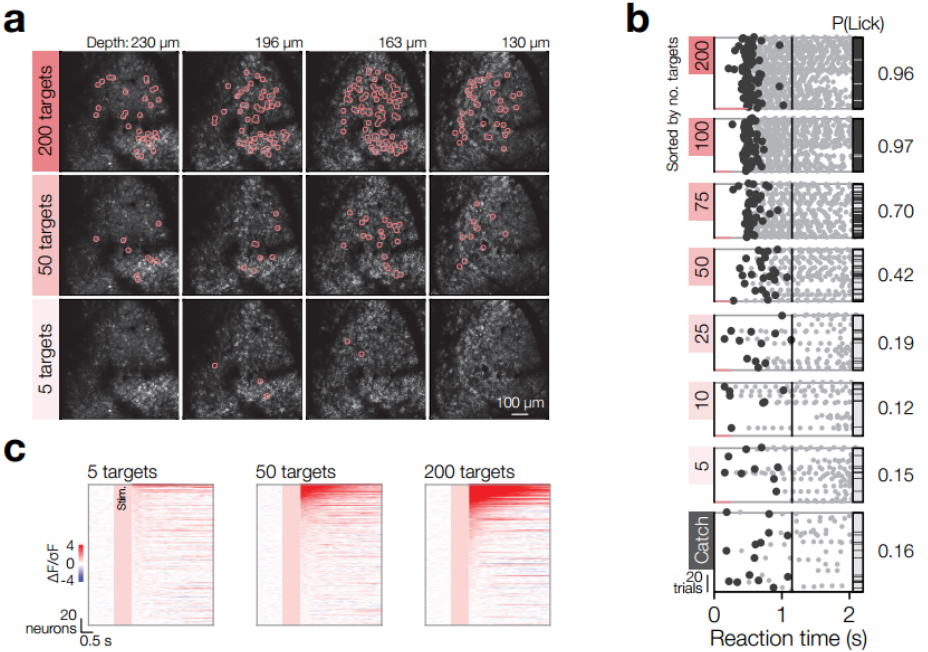
- For each panel: **Key takeaway = key visual effect**
- For each figure: **Minimise number of elements**
  - Use ‘multiples’ in lay-out



Vogels ea, 2011, Science

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



Dalgleish, Russell, Packer ea, 2020, elife

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

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