Practical tips and tricks

Approaching/vetting a new dataset

Data pre-processing

Data munging/storage

Null/control datasets

Statistical testing

Choosing an analysis method

Analyzing fit models

Designing custom analyses

Leveraging LLMs

Coding strategies

Reproducibility

Data sharing

Practical tips and tricks - Approaching/vetting a new dataset (part 1)

- Look at the data
  - Before trying anything sophisticated, spend time inspecting the data to get a feel for what it looks like overall. Is there clear signal? What clear phenomena are present?
  - Plot a bunch of representative or “typical” examples of what the data look like. (This is very crucial for communicating with others about the data too.)
  - Plot some of the “atypical” (if any) data, which could be interesting or artifactual.

- Understand how it was collected. Experimental context is variable yet extremely important.
  - Talk to the people who collected it (if possible).
  - Read about the experimental procedures and what their advantages and drawbacks are.
  - Identify which phenomena are likely artifacts.

- Compute all the basics
  - Quantify all the basic histograms, correlations, and timescales.
  - Quantify the standard things specific to your data type (e.g. PSTHs).
  - Quantify anything else obvious, e.g. statistics reflecting how common and variable the representative phenomena are.

- Think about how would you make surrogate datasets (e.g. shuffling across trials). This is the best way to test whether analyses results are meaningful or artifactual.

Practical tips and tricks - Approaching/vetting a new dataset (part 2)

- Make a report with all the basics.
  - Representative/typical and atypical examples
  - Notes on experimental context.
  - Histograms, correlations, timescales, and basics quantities specific to the dataset.
  - Brief write-ups in each section describing basic scales, numbers and patterns.
  - Save as PDF.
  - This is your executive summary of the data that will make it much easier to communicate with others and to contextualize more sophisticated analyses.

- Make predictions
  - Write down all your own predictions about the data and whether they align with your quantifications so far.
  - More importantly, identify what predictions experts in the field would make about your data and whether they are aligned or violates by the data.

- Identify specific scientific questions
  - What are the obvious questions raised by the dataset (e.g. violations of predictions)?
  - Write down any questions you were originally interested in about the system and how straightforward it seems to address them using this dataset

Practical tips and tricks - Data pre-processing

- Identify the most common pre-processing steps for the data.
  - E.g. spike sorting, keypoint tracking for behavior videos, ROI extraction for calcium imaging.
  - Pick parameters for the pre-processing algorithms, noting what effects they are likely to have on downstream analyses (e.g. down-sampling/smoothing will make high frequency content invisible to later analyses).
  - Justify your parameter choices as well as you can, but more importantly write down what parameters you used. Sometimes it may seem like the most rigorous thing to do is to leave a parameter free/variable so that you can assess how it will affect downstream analyses, but doing this for every parameter quickly becomes impractical. It's okay to pick something reasonable and move on.
  - Some pre-processing steps will be very standard and expected by readers (e.g. spike sorting), but others will be specific to your dataset/question and require more justification.

- Pre-process your data.
  - Make figures comparing your raw to pre-processed data.
  - Write a report detailing the pre-processing steps and showing examples of the raw vs. pre-processed data side-by-side or overlaid.

Practical tips and tricks - Data munging/storage

- Data munging (transforming/formatting data) is very time consuming and error-prone. Thinking carefully about how to format/store your data is very valuable in the long run.

- Keep your data in one place. For scientific projects, keeping an entire dataset in a single directory usually works pretty well. Database software used in commercial applications *can* be done, but it will mostly introduce unnecessary complexity.

- Choose a format that is easy to read by both computers and humans
  - Less code to load data means less complexity and fewer mistakes.
  - Human readable data is crucial for quick sanity checking.
  - E.g. CSV files are quick to load in Python and quick to scan by eye for basic errors.

- Most datasets are unique
  - If your data naturally fits with common standards, feel free to employ these.
  - BUT good organization/documentation is more important than following standards.

- Sometimes experimental details will change during the development phase of an experiment
  - This means data format/relevant measured variables can change too
  - It's okay to run one-off sanity checks/analyses during this process, but once experimental details have stabilized take the time to organize your data well before continuing.

Practical tips and tricks - Statistical testing

- Statistical tests and P-values are important for convincing yourself and your audience that phenomena you observe in a dataset did not emerge by chance.

- Most statistical tests, however, rest on key assumptions that are violated by time-series data
  - E.g. statistical tests usually assume data points are independent, but timepoints in a naturalistic time-series dataset are rarely independent

- Luckily, all statistical tests boil down to one idea
  - Define a statistic (scalar quantity) T that captures “strength” of the phenomenon/effect of interest.
  - Compute T_{data} for your true dataset
  - Create a chance/null distribution of *data*
    - One null dataset instance should have the same format as your true dataset (e.g. your original dataset with stimulus time-series randomly shuffled across trials)
    - The null data distribution is a collection of null datasets (e.g. many random shuffles)
  - Compute the same statistic T for each null dataset instance, leading to a null distribution over T
  - P-value ≈ estimated probability that a null dataset would have yielded a $T \geq T_{data}$
  - See Allen Downey’s blog post: https://allendowney.blogspot.com/2016/06/there-is-still-only-one-test.html

- The art is creating a reasonable null dataset. This is non-trivial and there can be many ways to do this.

Practical tips and tricks - Null/control datasets

- As analyses get more sophisticated, it becomes much easier to mistake a feature of one's analysis for a feature of one's data, but this is a huge error.
  - E.g. a specific clustering method may naturally tend to produce 4 clusters even in data without clusters, and it would be a mistake to apply this to your data and conclude there are 4 clusters.

- The best way to ensure your results are a consequence of your data is to run the same analyses on control/null datasets.
  - Positive control data = artificial dataset that exhibits the phenomenon of interest (e.g. 4 clusters)
  - Negative control data = artificial dataset that does not have the phenomenon of interest

- If you have designed/chosen an analysis method with a specific purpose in mind, run it on negative and positive control data first before applying it to your real data.
  - Helps you understand the nuances of the analysis method (which often have implicit parameterizations or biases).
  - Makes results applied to your data much more convincing, especially to careful reviewers.

- Also crucial for statistical testing (see previous section).

Practical tips and tricks - Comparing/deconstructing fit models

- Fitting models to data is often subtle and time consuming, and can easily appear to be the central obstacle in a project.

- But fitting a model to data does not generally constitute a scientific result.

- Model-based scientific insight comes from *comparing* different models against data. (Remember that ideally you will fit on one set of data and test on an independent held-out set of data.)

- A fairly standard and reliable approach is to remove/lesion features of the model to see which features are needed to fit data (generally want to re-fit and evaluate each model version).
  - E.g. Model A predicts $y = \beta_1 x_1 + \beta_2 x_2$, Model B: $y = \beta_1 x_1$, and Model C: $y = \beta_2 x_2$
  - If A significantly outperforms B and C, can conclude: “Under assumption of linear model, both x_1 and x_2 influence y.

- Think carefully about which models to compare, and advantages/limitations of set of models.

- In general there may not be a “best” model, since this depends on a specific objective function.
  - Different models may simply capture different aspects of the data (e.g. fast vs slow modes).

- Even if most of the technical challenge is in model fitting, always keep in mind what sets of fit models you’ll compare against data to produce your scientific result.

Practical tips and tricks - How to not make mistakes

- Inspect your code. Re-read the entire codebase line-by-line (out loud, ideally)
  - Work through logic in your head as you go.
  - This is an EXCELLENT way to catch mistakes.
  - It also takes much less time than it sounds and introduces zero moving parts.

- Test your code like a scientist
  - Most commercial software testing strategies are irrelevant (unless you're building software you plan to share with the community and actively maintain).
  - Controls (e.g. running your analysis on different types of artificial data) are much more important for convincing your audience that your science is sound.

- Constantly run sanity checks.
  - After every small change or update to your code/analysis, predict how you expect it to affect your results/plots, then validate that this is what occurred.

- Exercise EXTREME caution when re-using others' research code. Most research code may not work correctly with new data (e.g. it is very common for specific details of a dataset to be hardcoded somewhere in the codebase, which invalidates its application to other data.)
  - It's almost always be more efficient+sound to understand their method then reimplement it.
  - Note: OK to use well maintained scientific libraries e.g. numpy, scipy, pytorch, etc.

Practical tips and tricks - Leveraging LLMs

- LLMs are a tool. Like any tool they can be used skillfully or unskillfully.

- Helpful ways to use LLMs
  - Hypothesis/prediction generation
  - Literature reviews/getting up to speed quickly with current thought in a new field
  - Generation of strategies for organizing/munging data
  - Generation of analysis ideas for addressing specific questions
  - Code generation

- Risks of using LLMs
  - Cited literature doesn’t exist/
  - Code doesn’t run
  - Code runs but does wrong thing
  - Code runs but is not interpretable, hard to edit later

- Tips for using LLM-generated output
  - Verify references
  - Inspect generated code line-by-line
  - Test generated code on artificial datasets where ground truth is known.

Practical tips and tricks - Reproducibility

- Reproducibility is fundamental to science and seems like it should be fully solvable in the realm of computational modeling/data analysis.

- But it's also important to be efficient about it and not take too much time away from other work.

- For typical science projects focus on *primary* rather than *secondary* reproducibility
  - Primary reproducibility = Code is very clear and convincing that it's doing what it claims.
    - Makes it easy for someone else to reimplement your analysis in their own language.
    - Think of your code as a precision methods section.
    - Choose clear variable names, good comments, key plots/controls for validation.
    - Does not depend much on software details or maintenance.
    - Requires thought + care but does not take that much time.
    - Arguably more crucial for long-term reproducibility.
  - Secondary reproducibility = Other scientists can download your code and run/extend it.
    - Much harder/time-consuming + requires significant additional software engineering.
    - Depends on software details and requires active maintenance.
    - Code can still be massively confusing, even if it runs on someone else's machine.
    - Sometimes introduces additional software/moving parts, etc, increasing complexity.
    - Mostly useful when developing software you plan to distribute and actively maintain.
  - Incidentally, prioritizing primary reproducibility makes secondary reproducibility much easier.

Practical tips and tricks - Data sharing

- Data without good documentation is not very useful
- Sharing data with others requires time investment
- To make data really accessible to others:
  - Format in friendly way that does not require complex software to open
    - E.g. CSV files, clear naming conventions
  - Provide documentation on data organization
  - Provide documentation on experimental context + likely artifacts
  - Provide executive report on data, showing representative samples and basic stats
    - Useful for sanity checks when others are loading/viewing data
- When requesting data from others, be mindful of:
  - The huge effort that may have gone into collecting the data
  - The time involved on their end to format/document the data to make it user friendly.

Practical tips and tricks - Coding strategies

Choose a great color palette

Practical tips and tricks - Designing custom analyses

Be creative

Combine domain and technical knowledge

Test on simulated null/control data