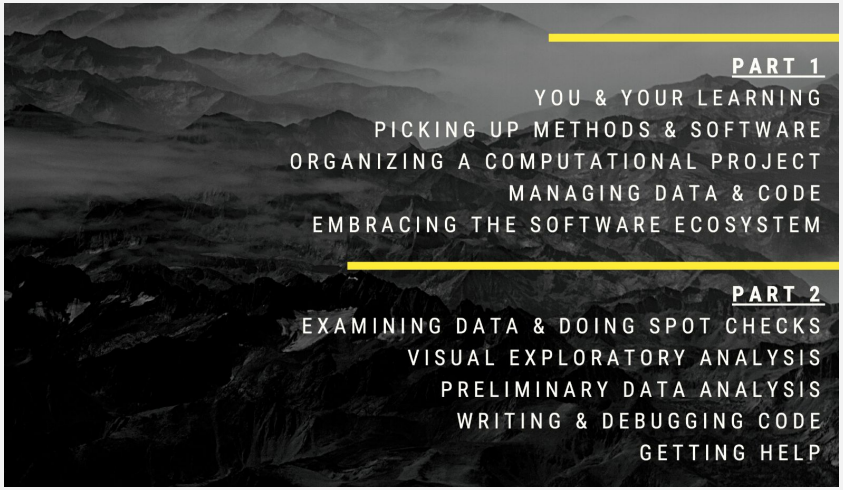
COMPBIO METASKILLS

MAR 16 | 4–5 PM MT

In-person: DBMI 6th Fl Conference room

Zoom: bit.ly/cm_zoom



Arjun Krishnan, Associate Professor, DBMI

thekrishnanlab.org

arjun.krishnan@cuanschutz.edu

@compbiologist

Sign-in: bit.ly/cm_signin

Receive the **slides+recording** and a PDF of "**The Pragmatic Scientist**", a cheatsheet of best practices for doing computational research.

CompBio Metaskills

Part 1

- You & your learning
- Picking up methods & software
- Organizing a computational project
- Managing data and code
- Embracing the software ecosystem

Part 2

- Examining data & doing spot checks
- Exploring and prototyping
- Preliminary data analysis
- Writing & debugging code
- Getting help

An informal discussion, so no grades/credits.

The focus is on general practical principles and global thinking. Not any specific tools/methods. No coding.

Notes about my specifics recommendations

For many topics, I'm going to recommend *specific* ways for doing things

1. Walk away with the broad strokes and motivations.
   You can imbibe the details only through practice.

2. Basic principles (e.g., reproducibility) are incontrovertible;
   the hyper-specific details can surely vary.

3. Much of this is also, obviously, an opinionated view.
   You don't need to agree with *everything* I recommend.

Think of everything discussed here as starting points for your thinking, learning, and development.

Stop me anytime and ask Qs!

CompBio Metaskills

Part 1

- You & your learning
- Picking up methods & software
- Organizing a computational project
- Managing data and code
- Embracing the software ecosystem

Part 2

- Examining data & doing spot checks
- Visual exploratory analysis
- Preliminary data analysis
- Writing & debugging code
- Getting help

Learning to do research from reading papers & listening to talks

Types of computational research studies

- New analytical/computational method
- Improvement of an existing method
- Evaluation of existing methods
- Development of (re-)usable software, web-service, or database
- New insights w/ new/existing methods

Learn from your favorite papers about…

- How to frame a problem
- Choose the methods/tools
- Set up an analysis workflow
- Establish groundwork, &
- Generate a series of supportive results towards answering the central question.

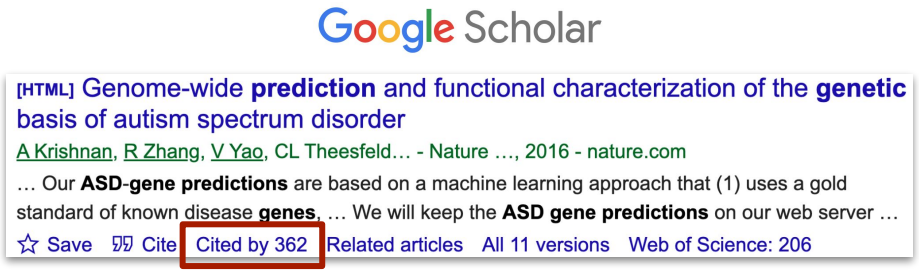
Look into sources beyond research articles

Reviews (biomedical & methodological)

- The “thinking” and vocabulary of a sub-field
- Major papers and scientific milestones
- Open questions

Other papers that cite a particular paper

- New perspective, description, and context



Look into sources beyond research articles

Blogs, tutorials, lectures, talks, podcasts

- Available at *all* levels of expertise
- Can be tastefully paired with primary research articles
- Cover many aspects of science absent in primary literature, including things not to do

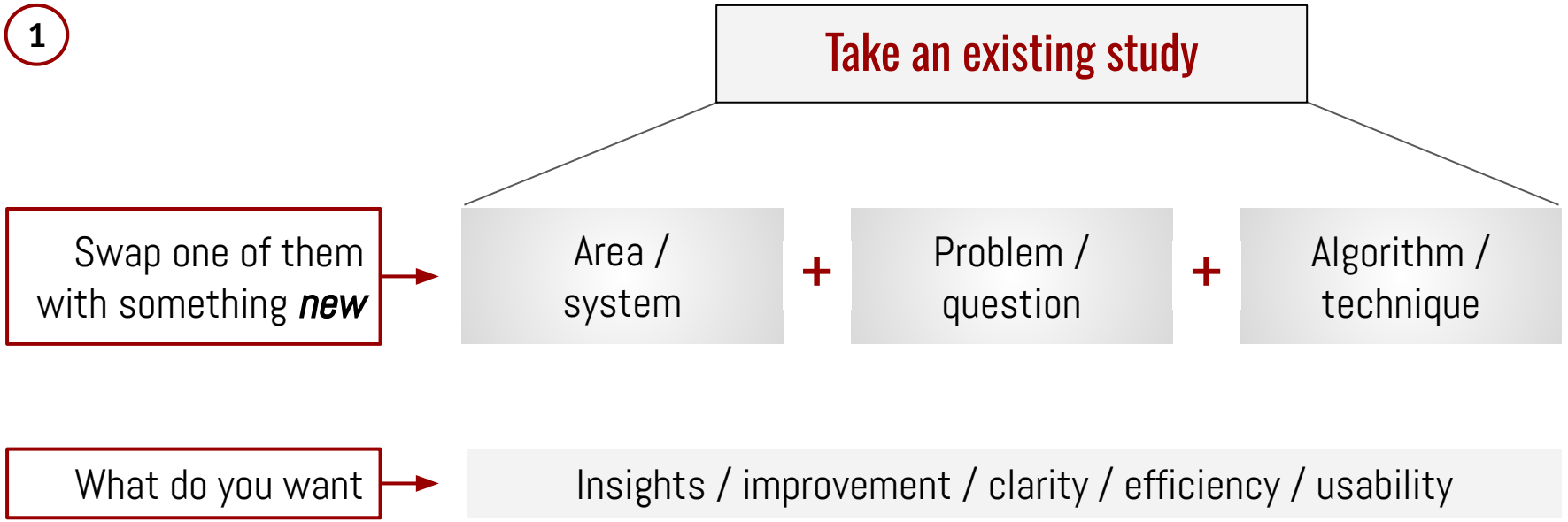
Great way to learn:

- Practical aspects of many theoretical ideas
- Visually, via demonstrations, plots, animations, videos

Reading, retention, & reuse

- **Make reading papers & listening to talks a habit**.
- **Critically analyze what you read/listen**. Don't be swayed by high-profile papers/speakers, media hype, or current dogma.
- **Don't repeat yourself**: Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.
  - Use a **reference manager** (e.g. Zotero), put *everything* you read into it. Use tags to group papers by subfield/method/data.
  - Create and maintain a **single notebook** (Google Doc; Evernote) with notes/text-excerpts/figures from all papers & reading materials. Add notes about each paper / dataset / method.
  - Create and maintain a **single glossary** of all the technical terms and vocabulary for your project.
- **Contextualize what you read/listen** in relation to everything else you know / have read. Specifically consider limitations. Analyze information in terms of you and your project.

How to identify new research problems?



How to identify new research problems?

2

Inspiration from biology / biomedicine

Interesting example → Generalized problem → Large-scale solution / insight

3

Inspiration from other fields (CS, math, statistics, etc.)

Interesting technique → Applied to my problem of interest → New method / analysis / insights

Picking up a new method or software

Read software/methods papers

- Read the Introduction & Discussion.
- Modern papers also have **graphical schematics of their methods/algorithms**.
- Use Google Scholar to find **recent application papers that use** the software/method & read those.
- Search and read **blogs** and watch YouTube **videos**.
- Together, these will not only help you understand the methods but also key **assumptions and parameters** that you need to think about for your project.

Picking up a new method or software

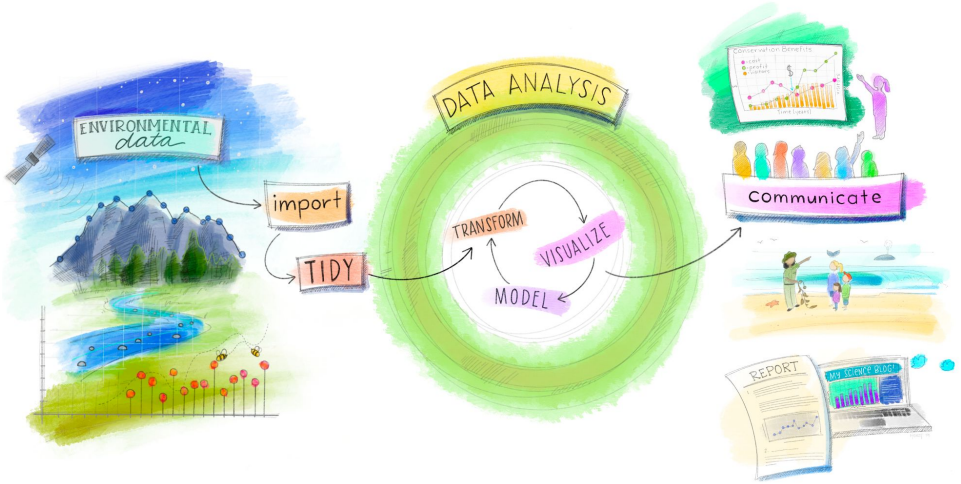
Explore the actual software/code

- Read the **documentation**: Overview and parts of it that correspond to the assumptions & parameters relevant to your project.
- Look into the exact data **input & output formats**.
- After installation, **replicate an example** run exactly as-is from the documentation/website or from an independent online tutorial.
  - If neither is available, email the (first & corresponding) authors asking for example data & detailed instructions on how to run their code.

Organizing a computational project

`project_directory`

- `data`
  - primary & processed data + `readme.txt` + `runlog.sh`
- `src`
  - all your code/scripts
- `bin`
  - all compiled code + installed binaries + `readme.txt`
- `doc`
  - literature notes + analysis notes + intermediate/final report
- `results`
  - YYYY-MM-DD sub_directories
    - `runlog.sh` + R/Python notebooks



Based on Noble (2006) PLoS Comp. Biol.

Organizing a computational project

`project_directory`

- `data`
  - primary & processed data + `readme.txt` + `runlog.sh`
- `src`
  - all your code/scripts
- `bin`
  - all compiled code + installed binaries + `readme.txt`
- `doc`
  - literature notes + analysis notes + intermediate/final report
- `results`
  - YYYY-MM-DD sub_directories
    - `runlog.sh` + R/Python notebooks

No manual editing of data; Write scripts

Details on when & where data was downloaded

No code in this dir; Should point to & run code from `src`; this file should have all the command-lines used to run the code/scripts to process data here

Including those used for data download, processing, and analysis; Well documented with detailed comments within the code + external documentation.

Details on when and from where external software was downloaded; also include installation instructions if it was not straightforward.

Based on Noble (2006) PLoS Comp. Biol.

Organizing a computational project

`project_directory`

- `data`
  - primary & processed data + `readme.txt` + `runlog.sh`
- `src`
  - all your code/scripts
- `bin`
  - all compiled code + installed binaries + `readme.txt`
- `doc`
  - literature notes + analysis notes + intermediate/final report
- `results`
  - YYYY-MM-DD sub_directories
    - `runlog.sh` + R/Python notebooks

One file named with YYYY-MM-DD date of each analysis; Should contain free-text details on the thoughts/ideas behind that day's analyses.

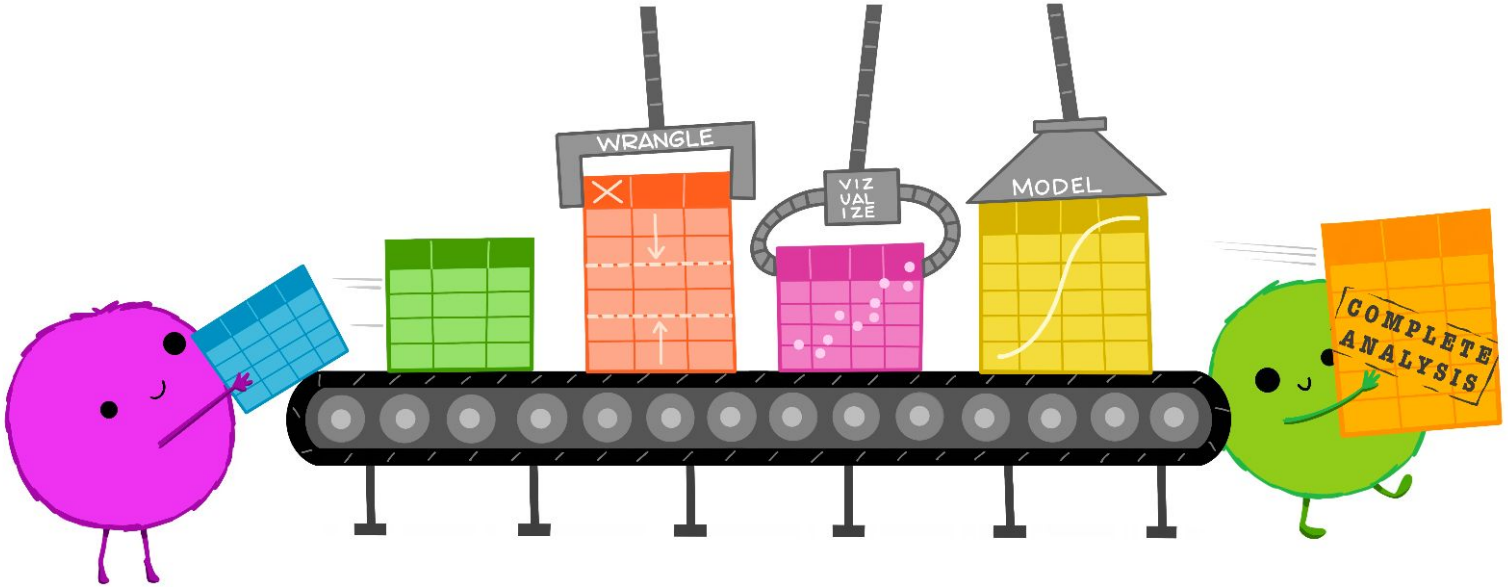
Used at the later stages of a project to pull all the results into a report/paper.

At each stage of an analysis, gather your results (as text files) & make plots to visualize & interpret.

Should point to & run code from `src`; This file should have all the command-lines used to run the code/scripts to produce the results here.

Based on Noble (2006) PLoS Comp. Biol.

Managing data & code: automate everything (as much as possible)



Managing data

- Give all files meaningful, interpretable, & computable names
  - Machine readable, human readable, works well with default ordering.
- Do not tamper with original/source files
  - `readme.txt` should contain detailed information about when & from where each piece of data was obtained.
- Do not make changes by hand; Automate everything
  - Write scripts that read in the file and generates the desired file.
- Document everything
  - Keep track of all your commands (Linux & running code) in a `runlog.sh`.

Examples of bad vs. good filenames

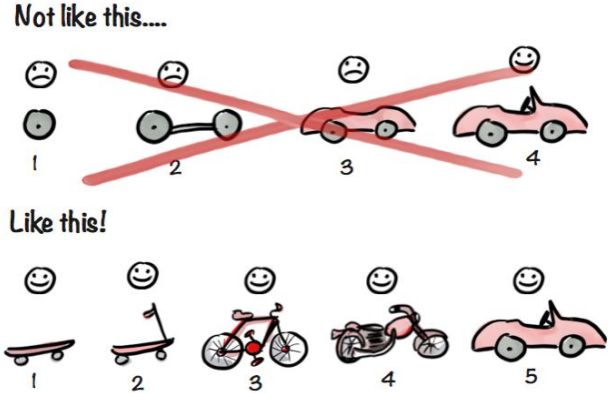
BAD	BETTER
`01.R`	`01_download-data.R`
`abc.R`	`02_clean-data_functions.R`
`fig1.png`	`fig1_scatterplot-bodymass-v-brainmass.png`
`IUCN's metadata.txt`	`2016-12-01_IUCN-reptile_shapefile_metadata.txt`

https://speakerdeck.com/jennybc/how-to-name-files

Managing code

- Write code for both computers & humans.
  - Give descriptive, interpretable variable & function names.
  - Comment your code at the top: purpose, expected usage, example inputs/outputs, dependencies.
  - Record imports, constants, random seeds at the top.
  - Comment each block/function: the intended computation, arguments, return values.
- Program for the general case, and put the specifics outside the code as arguments & parameters.
- Eliminate effects between unrelated things.

Continuously functional & testable



Spotify

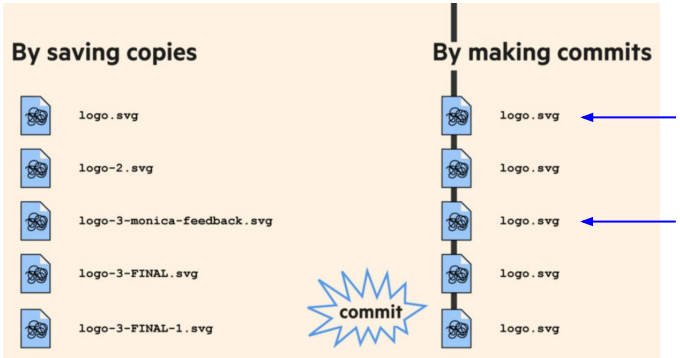
Reusing existing code:

- Begin by adding detailed comments.
- Properly acknowledge code borrowed from elsewhere; Check license.

Managing data & code

Version control

- Storify your project
- Travel back in time
- Experiment with changes
- Backup your work
- Collaborate effectively



Arjun Krishnan
12:34pm January 3th 2018

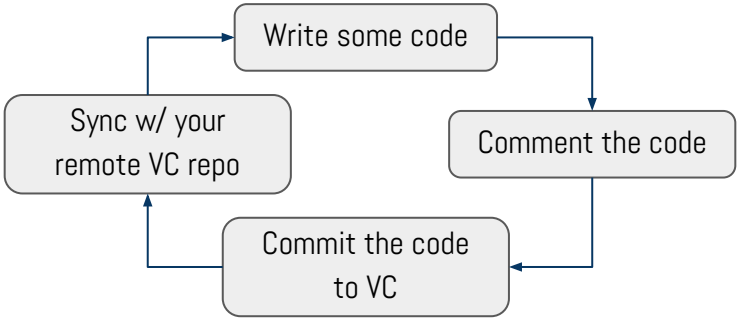
Updated background color

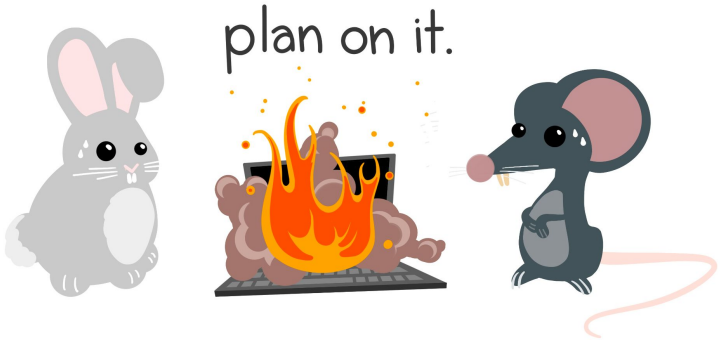
Changed background color to improve contrast.

Arjun Krishnan
9:15am January 4th 2018

Incorporated feedback from team

Made all changes based on team.org/feedback314





Adapted from @alicebartlett; allisonhorst.com

Embrace the software ecosystems; Don't pick a camp

Language, IDE, Notebook	• R | RStudio | R Notebook	• Python | Rodeo | Jupyter
Pre-built external packages	• CRAN, Bioconductor	• PyPI, Biopython
Scientific computing	• In-built + Hundreds of packages	• NumPy, SciPy + Hundreds of packages
Data wrangling & visualization	• Tidyverse	• Pandas, Seaborn

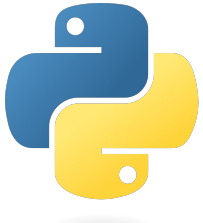
There are hundreds of software packages for bioinformatics & computational biology written in various languages (C, C++, R, & Python) that can be run from the command-line.

- Linux command-line
  - Navigating the file system
  - Running code
  - Manipulating data
  - Writing shell scripts

Embrace the software ecosystems; Don't pick a camp

Different language ecosystems are better at different things.

Data wrangling	Genomics
Visualization	Network analysis
IDEs/Notebooks	EHR
Package management	Machine learning
...	...



Focus on the problem and pick & choose tools freely from all ecosystems.

Embrace the software ecosystems

Notebooks

- Code
- Documentation
- Results: plots, tables, or any other output
- Text descriptions of background/motivations/conclusions

CompBio Metaskills

Part 1

- You & your learning
- Picking up methods & software
- Organizing a computational project
- Managing data and code
- Embracing the software ecosystem

Part 2

- Examining data & doing spot checks
- Visual exploratory analysis
- Preliminary data analysis
- Writing & debugging code
- Getting help

Time for some reflection: 3 min

What are three take-homes for you from Part 1?

Folks **in-person**:

- Pick up sticky notes and pens and, individually, record your thoughts.
- Then, find one other person next to you and exchange what you wrote.

Folks on **zoom**:

- Go to **bit.ly/cm_jamboard** and, on board 1, record your thoughts using sticky notes.
- A couple of you can talk about what you wrote.

CompBio Metaskills

Part 1

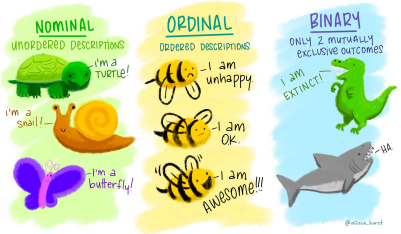
- You & your learning
- Picking up methods & software
- Organizing a computational project
- Managing data and code
- Embracing the software ecosystem

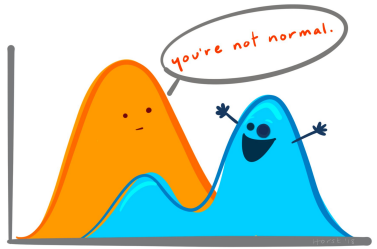
Part 2

- Examining data & doing spot checks
- Visual exploratory analysis
- Preliminary data analysis
- Writing & debugging code
- Getting help

Examining data & doing spot checks

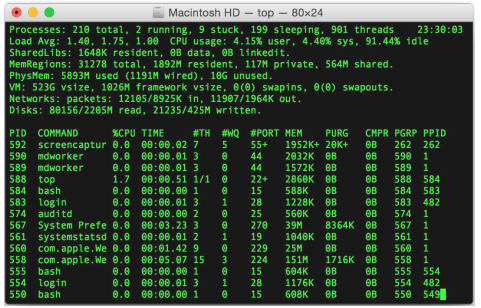
- Data structure, dimensions, and scale:
  - Structure (rectangular, list of entries, dictionary, etc.) & format (plain text, spreadsheet)
  - Top & bottom entries; Number of rows/columns
- Exploring specific aspects of the data (e.g., different columns)
  - Top & bottom 10 entries sorted by values in that column
  - Continuous variables: Central & spread of values (mean, variance, quartiles, IQR)
  - Discrete variables: Unique values and their frequencies
  - Check data type (e.g., string, integer, float). Any columns with mixed data types?
- Missing values & outliers
  - Number of rows/columns with MVs and outliers (Histograms of # MVs or outliers across rows/columns)





Examining data & doing spot checks – the Linux command-line

`less`	Peruse file
`head`	Print top of the file
`tail`	Print bottom of the file
`cat`	Print the whole thing
`wc`	Word count
`cut`	Cut columns
`sort`	Sort lines
`uniq`	Report/omit repeating lines
`grep`	Print lines matching pattern



Visual exploratory analysis

1. Distribution of each continuous variable (each column) [boxplot & violin plot]
2. Proportion of observations (x) vs. Cumulative proportion of values (y) [line plot]
3. Mean vs. variance of all observations (across samples; dispersion) [scatter plot + smooth]
4. Correlation of mean value of each observation with additional metadata [scatter plot + smooth]
5. Correlation of variables to each other [scatter plot + smooth]
6. Clustering/similarity of variables [heatmap, PCA]
7. Concordance with prior knowledge/expectations [barplot + boxplot]
8. Agreement with existing data [scatter plot, heatmap]

Visual exploratory analysis

1. Distribution of gene expression in each sample [boxplots]
2. Proportion of genes (x) vs. Cumulative proportion of reads (y) [lineplot]
3. Mean across samples vs. variance across samples (dispersion) [scatter plot + smooth]
4. Plot correlation of expression with length, GC-bias, etc. [scatter plot + smooth]
5. Are replicates highly correlated with each other? [scatter plot + smooth]
6. Cluster samples and visualize similarities [heatmap, PCA]
7. Do known genes show the right pattern of expression? [barplot of single genes across samples; boxplot of groups of genes across samples] (e.g. genes that show a phenotype)
8. Does it agree with existing transcriptome data? [scatter plot, heatmap]

Preliminary data analysis – Fail fast and learn

- Exploration + prototyping: critical for determining if the problem is well-defined & tractable.
- Perform preliminary analysis using simple baselines and sample datasets
- Make visualization an integral part of every stage of your project, including early exploration.
- Don't speculate. Instead, implement something and check your assumptions.
- The value lies not in the code/plots you produce, but in the lessons you learn.

twitter.com/JennyBryan/status/952285541617123328

One of the most useful things I've learned from hanging out with (much) better programmers: don't wring hands and speculate. Work a small example that reveals, confirms, or eliminates something.

Preliminary data analysis

- Simple baselines
  - Most frequent value
  - Average/median value
- Randomized baselines
  - Identical method run on permuted data (randomized based on various aspects)
- Sample datasets & Toy examples
  - Make small datasets by hand to make sure your code or external software works exactly as expected.
- Positive and negative controls
  - Outputs you expect and don't expect

Preliminary data analysis: public data repositories

Genomes & proteomes

all encompassing
Ensemble

comparative genomics
COGs | InParanoid | OrthoMCL

ref. gene/transcript sequences & annotations
RefSeq | Entrez | GENCODE

sequences variation
1000 Genomes | dbSNP

everything protein
UniProt | InterPro | SCOP | CATH | PDB

Annotations & relationships

process, function, component
Gene Ontology

pathways
Reactome, KEGG, WikiPathways

networks
BioGRID, STRING

Phenotype/Disease

OMIM | GWAS Catalog | ClinVar | COSMIC

Genome-Phenome

dbGaP | UK Biobank | FinnGen

Functional/regulatory genomics

data sets
NCBI GEO | EBI ArrayExpress

raw reads
NCBI SRA | EBI ENA

consortia
ENCODE | GTEx | TCGA

curated public data
Expression Atlas, Recount2, ARCHS4

Model organism databases

MGI | RGD | TAIR | FlyBase | ZFin | WormBase | SGD

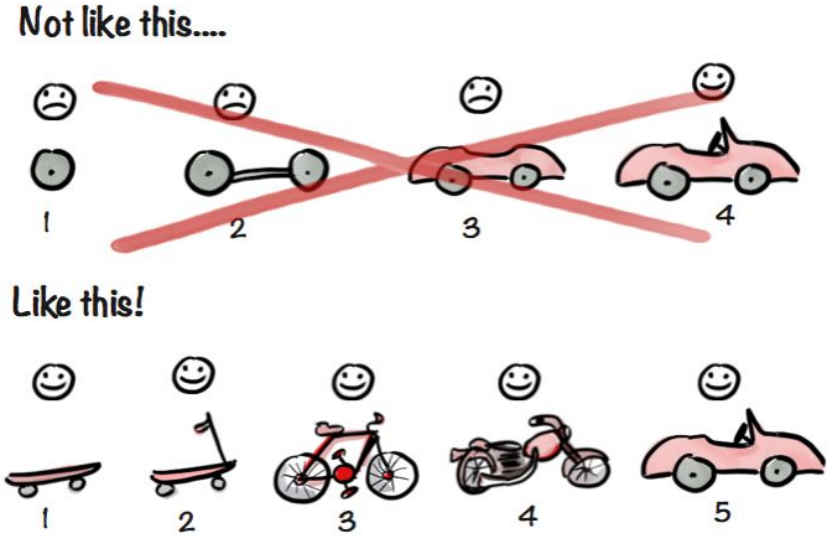
Writing code for the first time

- Give variables and functions longer meaningful names.
  - Use autocomplete to call functions/variables
- Make the code talk to you.
  - Generously add `print` statements that help you closely monitor the progress of your code through the various steps
    - Time stamps
    - Description of which step is currently running
    - File names
    - File paths
    - Variable names
    - Variable dimensions
    - Variable type/class
    - The first few elements of large variables

Writing code for the first time

- Write modular code
  - Decouple unrelated parts
  - Test at every stage of development

Continuously functional & testable

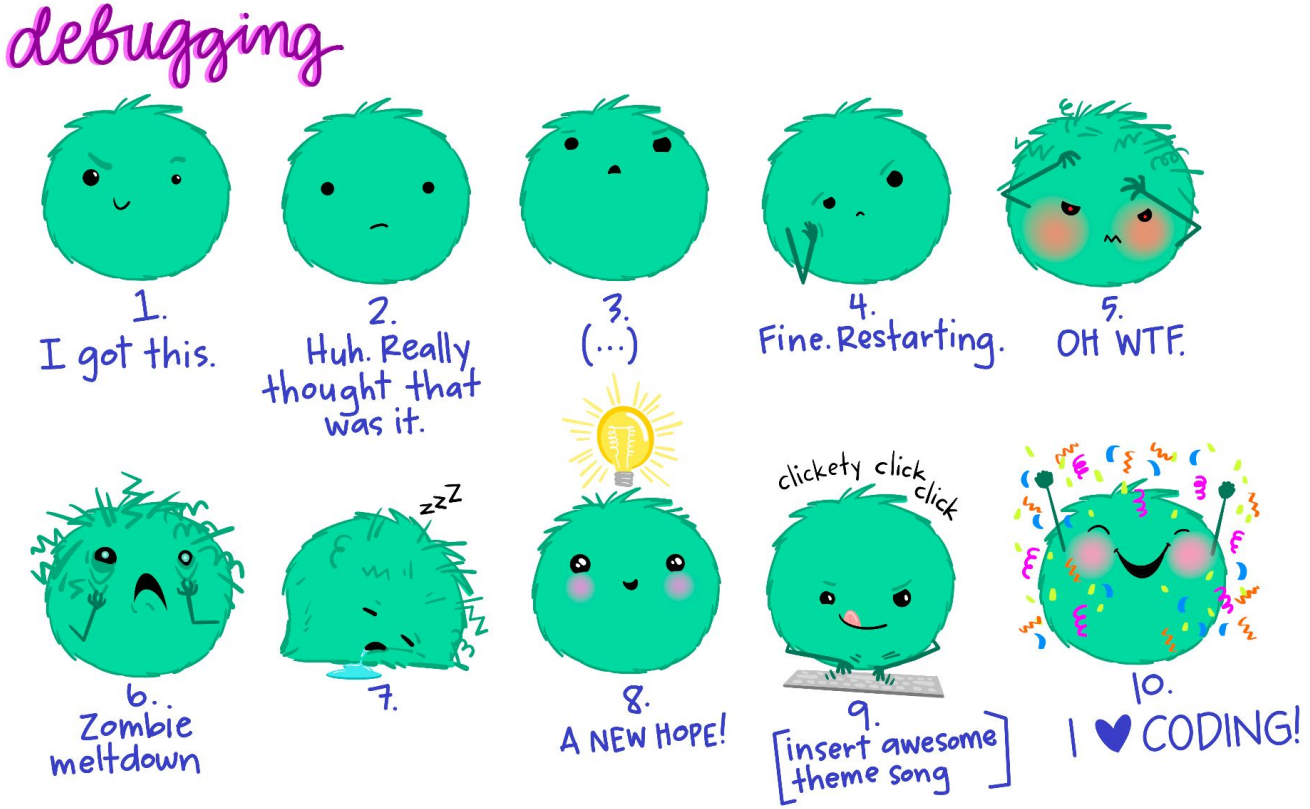


Spotify

Writing code for the first time

- Work with a smaller but representative input dataset that you *fully* understand and from which you know what the output(s) should exactly look like.
- Learn to create data inside your code
  - sampling, repeating, distributions, letters, built-in-datasets
- Run your code on shuffled/permuted data to see if the patterns/signals remain.

Debugging your code



Debugging your code

Debugging Bingo

Extra comma	Misspelled variable	Confused factor variable with a numeric one	Extra quotation marks	Put code into a markdown cell
Missing a tilda	Extra parentheses	Used wrong case (upper vs. lower case)	Misspelled function name	Didn't import the data
Confused when using $	Wrong argument(s) to a function		Confused a data frame for a variable	Missing pipe operator (%>%)
Didn't load libraries	Missed a comma	Forgot to ask R to print the object	Didn't close parentheses	Misspelled data frame
Confused = with ==	Used color when you meant fill	Forgot to save	Put regular text into a code cell	Missing quotation marks

https://twitter.com/cogscimom/status/1354508785365078016

Debugging your code

- Comment things out
- Print things out:
  - Contents of variables, variable dimensions, variable types/classes
- Print a file that contains the output of a code block
- Add conditional statements based on expectations
- Run loops just once (or just a few times)
- Terminate loops when something unexpected happens

Getting help with your code: just search

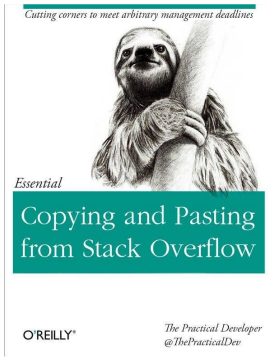
- Search error messages *as-is*
- "How to ..."
  - Add "R" or "Python"
  - Add the names of the function or package
  - Remove anything that's specific to your ecosystem
- Try variations/phrasings
- Learn to adapt solutions to related problems
- Post new questions/issues
  - Code snippets, not screenshots

Google

So many excellent blog posts & discussion forums!

stackoverflow

GitHub



ChatGPT

Getting help with your code: reproducible examples

Your code must be easily reproducible for someone else on their computer.

- If they cannot reproduce your error, they may not be able to help.
- If you do not provide an example, there is nothing for others to play with to figure out your problem.

stackoverflow

Getting help with your code: reproducible examples

Capture everything

- Include code you tried
- Any library/package calls
- Create all necessary objects (such as workable data)

The reproducible example needs to be complete but minimal.

- Strip away everything that is not directly related to your problem.
- This usually involves creating a much smaller and simpler dataset than the one you're facing in real life.

stackoverflow

Getting help with your code: reproducible examples

It's worth the work!

- Often, you solve your problem in the process of creating a reproducible example.
- If you do not solve your problem, then you've made it as easy as possible for someone else to help you from their computer (increasing the chances you get help!)

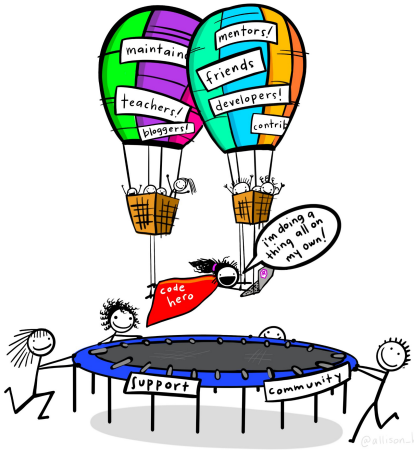
stackoverflow

You and your community

- **Be a catalyst for change**
  - Share new ideas, concepts, and software with your peers & colleagues.
  - Help in lifting-up the folks around you & create a vibrant, supportive community.

- **It's both what you say and the way you say it**
  - There's no point in having great ideas if you don't communicate them effectively.
  - Also needs to invite folks in, excite & educate them, and help build trust.

- **Carry the spirit of computational biology with you**
  - A confluence of biology, statistics, applied math, engineering, & computer science.
  - *Good ideas can come from anywhere and from anyone.*



Time for some reflection: 3 min

What are three take-homes for you from Part 2?

Folks **in-person**:

- Pick up sticky notes and pens and, individually, record your thoughts.
- Then, find one other person next to you and exchange what you wrote.

Folks on **zoom**:

- Go to **bit.ly/cm_jamboard** and, on board 2, record your thoughts using sticky notes.
- A couple of you can talk about what you wrote.

CompBio Metaskills

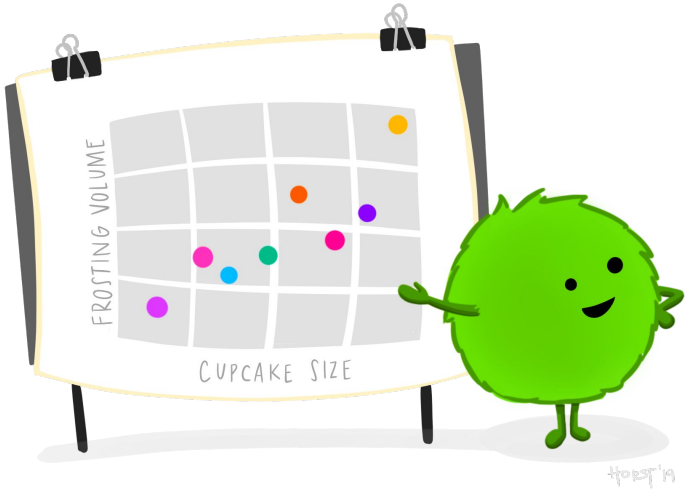
Part 1

- You & your learning
- Picking up methods & software
- Organizing a computational project
- Managing data and code
- Embracing the software ecosystem

Part 2

- Examining data & doing spot checks
- Visual exploratory analysis
- Preliminary data analysis
- Writing & debugging code
- Getting help

CompBio Metaskills



Part 3

- The purpose of data visualization
- Choice of visualization
- Typical issues in aesthetics & data handling
- Improving visualization for clarity

Time for some feedback from you: 3 min

Overall, what worked, what needs improvement, what is missing?

Record them each type of feedback on a green, orange, and red sticky note.

Folks **in-person**:

- Pick up sticky notes and pens and, individually, record your thoughts.

Folks on **zoom**:

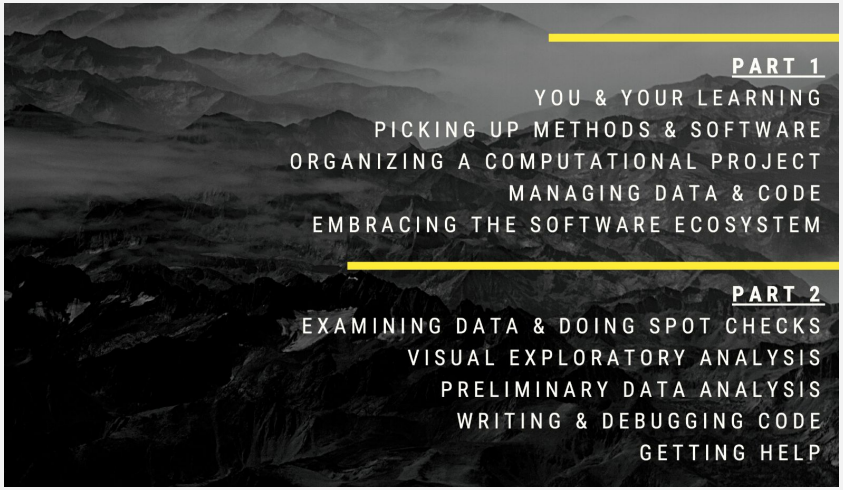
- Go to **bit.ly/cm_jamboard** and, on board 3, record your thoughts using sticky notes.

COMPBIO METASKILLS

MAR 16 | 4–5 PM MT

In-person: DBMI 6th Fl Conference room

Zoom: bit.ly/cm_zoom



Arjun Krishnan, Associate Professor, DBMI

thekrishnanlab.org

arjun.krishnan@cuanschutz.edu

@compbiologist

Sign-in: bit.ly/cm_signin

Receive the **slides+recording** and a PDF of "**The Pragmatic Scientist**", a cheatsheet of best practices for doing computational research.