Chapter III: Development of Open-Source Bioinformatics Software in Python

Introduction

Site-saturation mutagenesis (also known as deep mutational scanning) allows for the simultaneous, unbiased, and comprehensive study of all possible amino acid substitutions for a protein (Tripathi and Varadarajan 2014; Fowler and Fields 2014; Nov 2012; Shin and Cho 2015; Wrenbeck et al. 2017; Araya and Fowler 2011). First, a library of single-codon variants is plugged into an assay, where mutations in the protein affect functional selection. Next, the library is retrieved from the before and after selection samples. Then, using next-generation sequencing, the counts of each variant are used to calculate an enrichment score. Numerous applications have been pursued using site-saturation mutagenesis, including drug resistance prediction (Pines et al. 2020), protein engineering (Shin and Cho 2015), allostery determination (Subramanian et al. 2021), and functional analysis of genomes (Li et al. 2016).

A substantial number of software tools for different parts of the bioinformatics pipeline have been described in the literature. For example, for DNA library design, *Mutation Maker* and *OneClick* let users design oligos according to different specifications (Tang et al. 2012; Hiraga et al. 2021). For the pre-processing of DNA reads, *FLASH* and *PEAR* tools merge paired-end reads, *Cutadapt* and *Trimmomatic* remove adapter sequences (Zhang et al. 2014; Bolger et al. 2014), and QUASR provides a framework for quantification and analysis of short reads (Gaidatzis et al. 2015). To calculate enrichment scores and statistical analysis, *DiMSsum*, *enrich2*, *dms_tools2,* and *DESeq2* use different statistical models to quantify the errors and treat replicates and time series (Love et al. 2014; Rubin et al. 2017; Bloom 2015; Faure et al. 2020). This last subset of tools may also integrate a pre-processing module. For identifying molecular constraints that affect enrichment scores, there is *dms2dfe* (Dandage and Chakraborty 2017). Lastly, the Bloom lab recently released *dms-view*, a web-based JavaScript tool that lets users map site-saturation mutagenesis datasets to a 3-D protein structure (Hilton et al. 2020).

While the current tools allow users to calculate enrichment scores and error estimates from raw sequencing data, they only have a few built-in static visualization options such as heatmaps, logos, and scatter plots. Customizing these plots, or creating different plots, requires knowledge of programming and extensive data manipulation, making the creation of publication-quality figures challenging and time-consuming. In addition, none of these packages allows plots to be integrated with *Pymol* (Schrödinger 2015)*,* the go-to molecular visualization tool for scientists. Thus, overlaying enrichment scores onto a PDB structure must be done by hand, slowing the data analysis process. Here, we describe *mutagenesis-visualization*, a Python package that addresses the aforementioned needs. This software has been used to process, analyze, and visualize all the datasets shown in this dissertation.

Implementation

Mutagenesis-visualization is a user-centered Python API for processing, analyzing, and creating high-quality figures for publication from pairwise site-saturation mutagenesis datasets. Unlike other packages, *mutagenesis-visualization* handles all the data manipulation internally, streamlining the workflow. The software workflow is shown in Figure 1. First, the user creates a list of the DNA variants to synthesize. Outside this software, these variants are screened and analyzed on an *Illumina* instrument. Next, the unpaired reads are joined and trimmed using available tools (*FLASH, PEAR*, *Cutadapt,* and *Trimmomatic)*. Once the reads are paired and trimmed, *mutagenesis-visualization* counts the reads of each variant from the selected and unselected samples. *Mutagenesis-visualization* offers a few quality assurance (QA) tools for analyzing the quality of the DNA libraries. Afterward, enrichment scores are calculated. *Mutagenesis-visualization* has several options for processing and normalizing datasets. Due to the modularity of our software, this first step can be conducted separately on other software packages, and the data can be integrated back into the *mutagenesis-visualization* pipeline for further visualization analysis. The following section of the chapter covers how this step is conducted in-depth. Once the enrichment scores are calculated, different types of visualizations can be conducted.

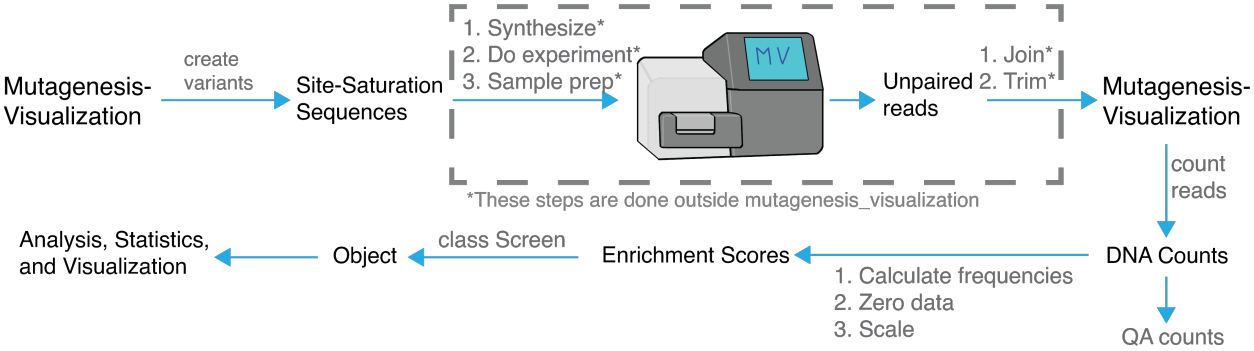

Figure 1. Workflow diagram of the *mutagenesis-visualization* package.

These datasets are commonly represented with heatmaps. In *mutagenesis-visualization*, heatmaps are highly tunable (Figure 2A). The user can change all stylistic aspects such as size, labels, color scheme, and protein cartoon. These heatmaps can also be customized by choosing protein residues of interest or selecting a subset of amino acids. In addition, heatmaps with average substitutions can be generated to easily visualize global phenotypic trends of the variants (Figure 2B). Since heatmaps are just one method of visualizing data, *mutagenesis-visualization* provides tools for plotting data sets through histograms, scatter plots, line plots, and bar charts (Figure 2.D, E, and H).

Furthermore, *mutagenesis-visualization* includes tools for hierarchical clustering and correlations (Figure 2C), PCA (Figure 2G), and ROC curves (Figure 2F). In addition, the *matplotlib* axes objects (Hunter 2007) can be retrieved, allowing for a high level of customization. Moreover, most visualizations, including heatmaps, can also be generated using *Plotly*, allowing interactive plots with hover features and dashboard creation. We have created a sample dashboard on *Heroku* to illustrate the capabilities of the software. Lastly, *mutagenesis-visualization* allows data to be easily exported and visualized onto a PDB structure in *Pymol* (Figure 2.I).

Mutagenesis-visualization is designed to be run locally and does not require a high-performance computing environment. The software can be installed from *PyPI* or *GitHub* using

the *pip* package manager and is compatible with Python ≥ 3.8. The source code is available in the *GitHub* repository. The documentation contains a step-by-step tutorial on the different types of plots, and a live version of the tutorial is hosted on *mybinder* and can be run remotely without any installation. The tutorial includes the analysis of sample site-saturation mutagenesis datasets from various studies (Dou et al. 2018; Fernandes et al. 2016; Livesey and Marsh 2020; Melnikov et al. 2014; Newberry et al. 2020; Stiffler et al. 2015; Bandaru et al. 2017).

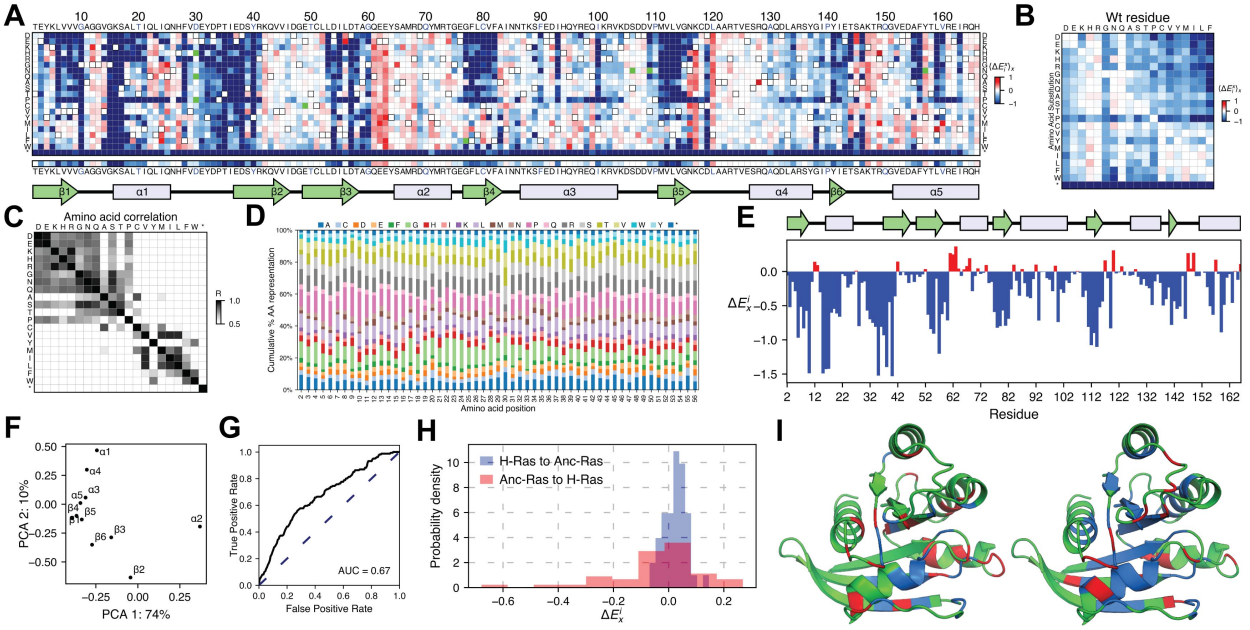

Figure 2. ***Mutagenesis-visualization*** **types of plots.** The user passes into the software a *pandas* dataframe (McKinney 2010) or *NumPy* array (Harris et al. 2020) containing the enrichment scores, the protein sequence, and, optionally, the secondary structure. The user can generate each of the following plots using a one-line command. The data used to generate the plots was obtained from replicating previous work on H-Ras (Bandaru et al. 2017). For figures A, B, E, and I, shades of red and blue indicate gain and loss of function, respectively. (**A**) A heatmap representing enrichment scores for substituting a particular residue in H-Ras with one of the 20 amino acids. The cartoon indicates the secondary structure motifs. (**B**) A heatmap with average amino acid substitution effects in H-Ras. (**C**) A correlation plot where the Pearson R-value is calculated for each amino acid substitution. (**D**) A bar plot representing the cumulative percentage of each amino acid substitution in the DNA library grouped by residue. The library was used to conduct the H-Ras site-saturation mutagenesis experiment. Each amino acid is coded with a different color. (**E**) A bar plot representing the average amino acid substitution effect per residue in H-Ras. (**F**) A scatter plot of the first two dimensions of a principal component analysis (PCA) conducted on the H-Ras dataset, which had been grouped by the secondary structure elements. (**G**) A receiver operating characteristic (ROC) plot used to make a quantitative determination of the fit between the measured enrichment scores and binary classification of the variants. The binary classification could represent any property such as the sequence conservation, whether found in a cancer screen database, or any other categorization that the user wants. (**H**) A histogram plot of the enrichment scores of all the sequence changes required for the H-Ras protein sequence to revert to Ancestral Ras and vice versa. There are forty-eight residues that differ between the two protein sequences (Bandaru et al. 2017). The data from the Ancestral Ras experiment are unpublished. (**I**) Two *Pymol*-generated figures where the alanine (left) and aspartate (right) enrichment scores were mapped onto the 5P21 PDB structure on *Pymol*. The user needs a valid *Pymol* license and the same Python path for *Pymol* and for the virtual environment used to run this software.

Code Organization and Object-Oriented Programming

Mutagenesis-visualization is written using Python's object-oriented programming (OOP) style, which helps improve efficiency by adding class re-usability, eliminates data redundancy, and facilitates code maintenance. The repository is hosted on *GitHub,* and it is divided into four main folders*: /data*, */main*, */tests,* and */tutorial*. The */data* folder contains all the site-saturation mutagenesis sample datasets used in the tutorial. The */tutorial* folder contains the seven *Jupyter* notebooks that comprise the tutorial. The */tests* folder contains the different unit tests. The percentage of code covered by the tests (code coverage) is 86%. Code coverage is calculated using *Codecov.* Finally, the */main* folder contains the data-processing functions and the plotting classes. The two classes that are reviewed here are *Pyplot and Screen* (Figure 3).

Pyplot is the skeleton for any class that analyzes and graphs the dataset. In Python, to make a new class inherit the properties and methods of *Pyplot*, the abstract class is passed in parenthesis after the new class [e.g., *class Heatmap*(*Pyplot)*]. This code structure avoids having to copy all the code every time a new plotting class is defined. Furthermore, a code update on *Pyplot* automatically updates *Heatmap* and all the other *Pyplot*-derived classes. This OOP style also makes it easier to test and fix bugs. The *__init__* method in *Pyplot* (Figure 3) takes as an input the data that is going to be analyzed and plotted. This input is handled automatically by the class *Screen* (described below). Then, the user will call the method *plot* and input specific graph parameters such as the axes labels.

Screen is the class that takes the raw data input by the user and checks that the inputs are correct. For instance, if the user passes an array with 40 columns, but the protein sequence only has 30 amino acids instead of 40, the software will raise an error. Then, the class handles the conversion of the raw data into a format suitable for analysis. For instance, if the user passes three replicates, *Screen* will internally calculate the average of the three, avoiding the user to do it manually. Finally, *Screen* instantiates each *Pyplot*-derived class and passes the data in the format needed by that specific class. By making a compact data pipeline, the number of data manipulation steps is reduced, lowering the burden on the user.

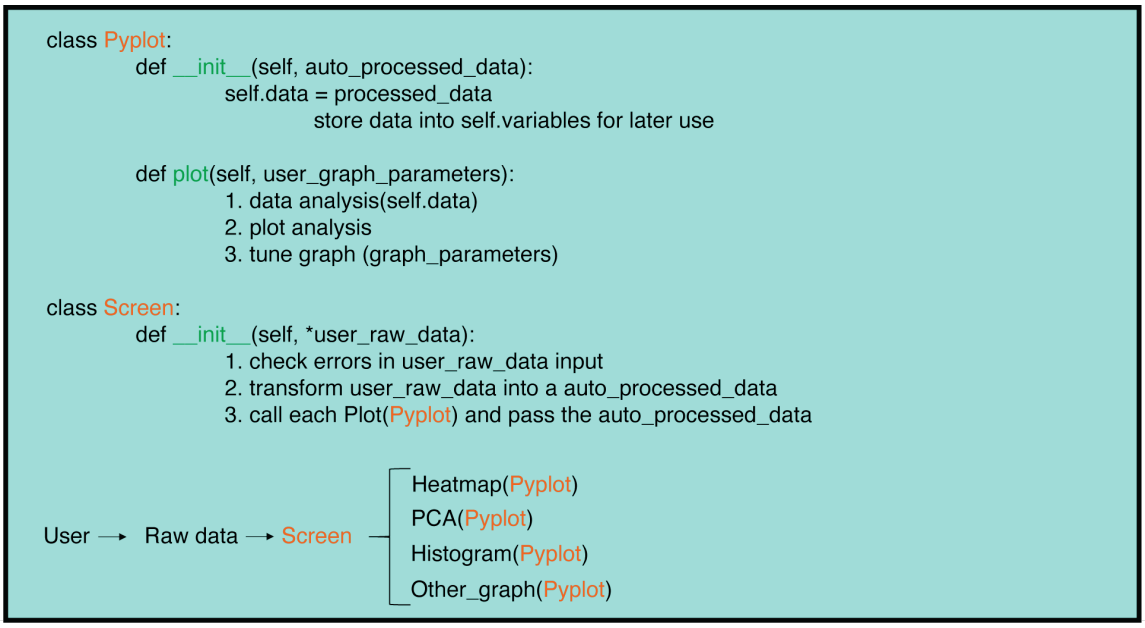

Figure 3. Pseudocode for the two main classes of the Python package, *Pyplot,* and *Screen*.

Dependencies Management and Virtual Environment Creation

By default, on Python, every project in the system uses the same directories to store and retrieve third-party libraries. When installing a new dependency, the current versions of the Python dependencies are usually not compatible with the requirements of the new dependency. For instance, *package_A* needs *NumPy* version 1.2, and *package_B* needs *NumPy* version 1.3. The two versions of *NumPy* are stored in the same directory with the same name. Therefore, only one version of *NumPy* will be used, and one of the packages will not work.

Virtual environments are used to circumvent such issues. A virtual environment is a tool that creates a new and isolated project, where the user can specify which version of each package to use. Following the example above, *package_A* would be in a virtual environment with *NumPy* 1.2, and *package_B* would be in a different virtual environment with *NumPy* 1.3. As a user, having to specify by hand the version of each library that goes into the virtual environment is an arduous task. Thus, when building a software package, it is essential to specify the exact requirements of the library for the user.

Mutagenesis-visualization has been written using *Poetry* for dependency management and packaging. For more information on *Poetry*, see its documentation. This tool allows us to declare explicitly the libraries *mutagenesis visualization* depends on, and it installs and updates them automatically. In addition, any user who wants to create a virtual environment to modify the *mutagenesis-visualization* source code simply needs to clone the repository and install the *poetry.lock* file in it. Furthermore, *Poetry* converted the *.lock* file into a *requirements.txt* file compatible with other package management tools such as *anaconda*. Having such a seamless code organization allows users to install and use the software, tweak the code to satisfy their own needs, and integrate the code back into the central repository, expanding the software's capabilities.

Incorrect Centering of the Distribution of Variants Leads to Inaccurate Subsequent Biological Conclusions

Mutagenesis-visualization contains the code used in this thesis to process the mutagenesis datasets. The *materials and methods* section in Chapter I explains briefly explains how enrichment scores were calculated. The following two sections expand on Chapter I content, and also explain alternative ways of analyzing the data. When analyzing a dataset, an arbitrary protein sequence (e.g., wild-type) is used to determine if a mutation is activating, neutral or deleterious. Often, an enrichment of score above zero is used to determine gain-of-function, and a score below zero is considered loss-of-function. If the distribution of variant enrichment scores is not centered correctly, subsequent biological observations will be impacted.

This section analyzed the same dataset in three ways and looked at the heatmaps (Figure 4) and data distribution (Figure 5). The enrichment scores displayed in the top and bottom heatmaps (Figure 4A and Figure 4C) have been generated using the "population" and "wt synonymous" methods, respectively (*see detailed methods* in *Statistics of Data Normalization*). Both heatmaps look similar; many variants are neutral (*white*) or loss-of-function (*blue*), and a few variants are activating (*red*).

In contrast, the middle heatmap (Figure 4B) is entirely blue because the enrichment scores have been generated using a different method, "counts." The method "counts" shifts the variant distribution below zero, placing all substitutions in the loss-of-function range. Consequently, the biological interpretation of Figure 4B is that mutating any residue in Ras is deleterious to the Ras•Raf-RBD interaction. However, this observation is a product of inaccurate data normalization. As a rule of thumb, most variants are expected to be neutral (*white*) in this type of experiment. The following section contains a detailed explanation of the statistics that *mutagenesis-visualization* offers using different normalization methods.

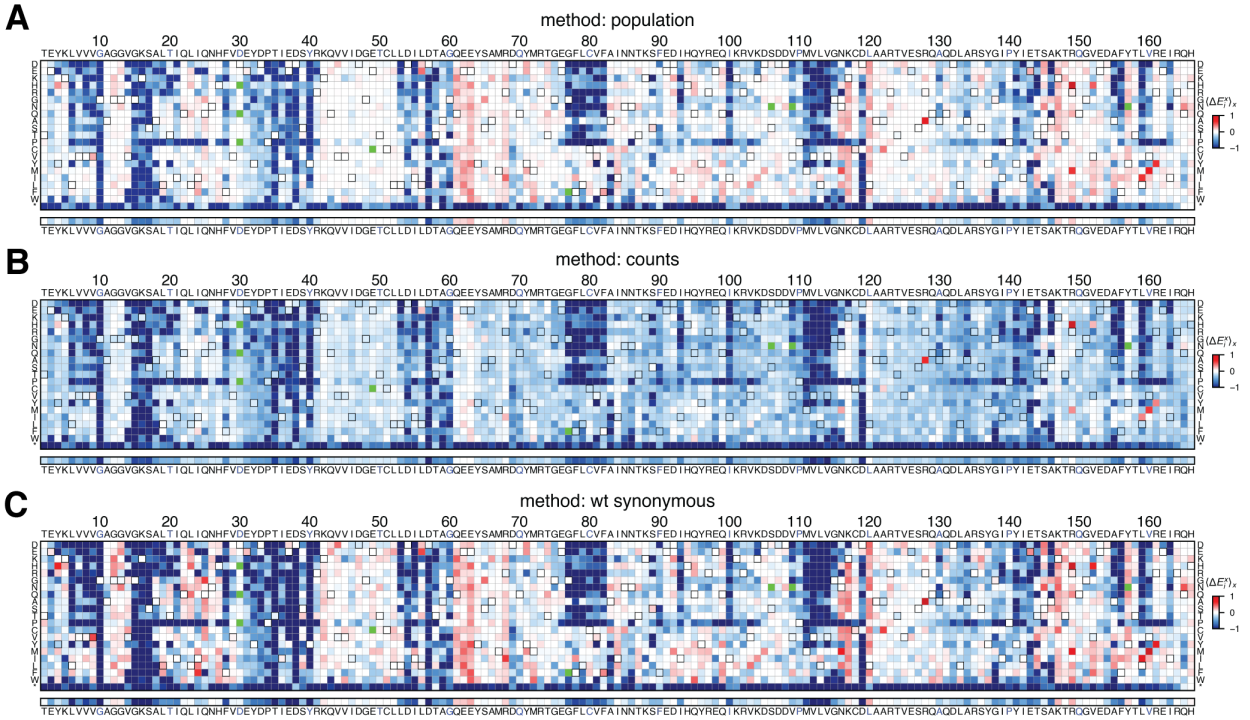

Figure 4. Processing a dataset via three different methods. The following heatmaps display the data of the H-Ras$^{2-166}$ bacterial-two-hybrid screen. (**A**) The data were normalized using the method "population" displayed in Figure 7E (**B**) The data were normalized using the method "counts" displayed in Figure 7B (**C**) The data were normalized using the method "wt synonymous" displayed in Figure 7.D

Figure 5 shows the kernel distributions of each of the three sub-libraries that contained the H-Ras$^{2-166}$ substitutions. The distributions are bimodal, with a left small peak that contains deleterious mutations, and the main peak that contains the vast majority of mutations. The two differences between the distributions displayed in three histograms are the amplitude and the main peak center score. The "wt synonymous" and "population" methods have the three sub-libraries centered at zero. Previous studies using the bacterial-two-hybrid have also reported this type of distribution (McLaughlin et al. 2012). Differences in the amplitude are a product of how the data is scaled. The kernel distributions of the data shown in the blue middle heatmap (Figure 4B) are shifted to the negative enrichment scores (Figure 5B). The subtle shift is causing the vast majority of enrichment scores to be negative, which, taken at face value, implies that the variants are deleterious from a biological standpoint. Standardizing the data correctly is an important step, and different datasets may require different processing methods. The following section reviews the equations of the different processing methods.

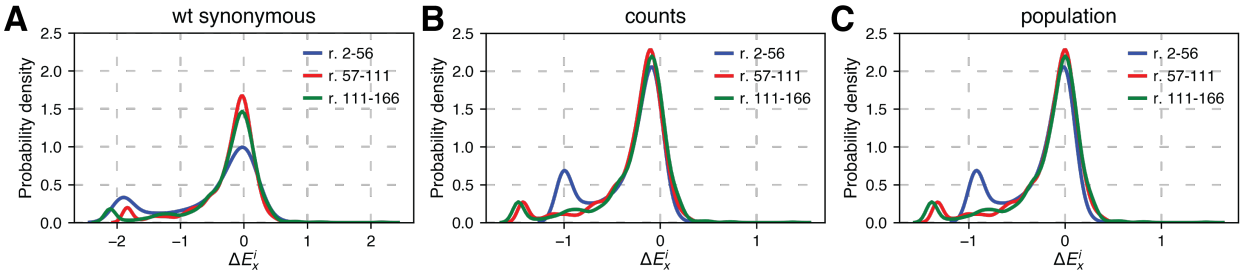

Figure 5. Sub-library kernel distribution of dataset analyzed via three different methods. The following kernels display the sub-library distributions of the heatmaps shown in the previous figure (Figure 4). Given that the maximum read length of the Miseq chips is 500 base pairs, we pooled the variants into three different sub-libraries (residues 2-56, 57-111, and 111-166, respectively). The sub-libraries were screened in separate flasks under identical experimental conditions. (**A**) The data were normalized using the method "population". (**B**) The data were normalized using the method "counts". (**C**) The data were normalized using the method "wt synonymous".

Statistics of Data Normalization

This section covers the statistics of the five different methods of obtaining enrichment scores included in the software ("counts", "wt allele only", "wt synonymous", "population", and "kernel"). Theoretically, the growth rate of a variant in the bacterial-two-hybrid assay (**Error! Reference source not found.**) is bound by the [0, α] limits, whereby a variant that impairs binding to the effector does not translate the antibiotic resistance gene, and, therefore, does not promote growth; and α is the growth rate of the organism without antibiotic exposure. However, in practice, there is leaky expression of the antibiotic resistance gene, and, therefore, some basal growth. For example, despite Ras variants with stop codons are not expressed, and therefore cannot bind to Raf-RBD, they have growth and a few counts are found in the selected samples.

Mutagenesis-visualization performs a baseline correction to account for the basal growth by using the counts of the stop codons present in the selected library. First, the counts of each amino acid *x* at each position *i* in the unselected sample ($c_i^{x,unsel}$) multiplied by the median ($\tilde{x}_{f^*}$) of the stop codon frequencies ($\frac{C^{*,sel}}{C^{*,unsel}}$) are subtracted from the counts in the selected sample ($c_i^{x,sel}$) (*Equation 1*). Then, the "corrected" selected counts ($c_{i,corrected}^{x,sel}$) are used to calculate the enrichment scores (ΔE_x^i) using any of the methods outlined below.

Equation 1: $$c_{i,corrected}^{x,sel} = c_i^{x,sel} - c_i^{x,unsel}\tilde{x}_{f^*} \; ; \; \tilde{x}_{f^*} = \text{median of } \left(\frac{C^{*,sel}}{C^{*,unsel}}\right)$$

While the correction does not affect the relative order of the variants with respect to the population, it changes the shape of the enrichment score distribution (Figure 6). Before the baseline correction, the enrichment scores distribution is bimodal (Figure 6A), and the new distribution is unimodal (Figure 6B). A scatter plot of the two distributions (Figure 6C) also depicts how the data is transformed. This baseline correction cannot be applied if the growth rate of the stop codon variants is the same as most of the other variants (i.e., Ras+GAP datasets).

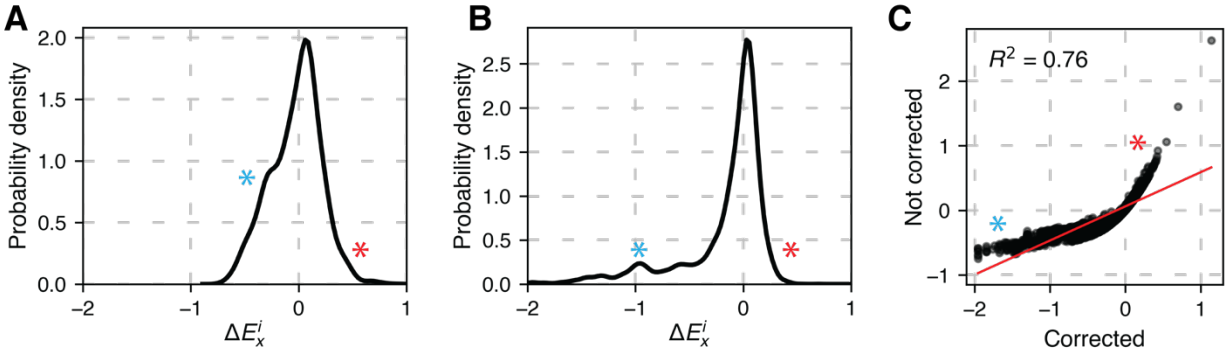

Figure 6. **Comparison of the same dataset when the stop codon correction has or has not been applied.** The blue "*" is situated at the loss-of-function shoulder. The red "*" is situated next to the activating mutations. (**A**) Kernel density estimation plot of the uncorrected dataset. (**B**) Kernel density estimation plot of the corrected dataset. (**C**) Scatter plot of the corrected dataset vs. the uncorrected dataset.

After the baseline correction, the first step in the calculation of enrichment scores is to obtain the $\log_{10}$ ratio of counts (*c*) of observing each amino acid *x* at each position *i* in the selected (sel) and unselected (unsel) samples:

Equation 2: $$\Delta E_x^i = \log_{10}\left[\frac{c_i^{x,sel}}{c_i^{x,unsel}}\right]$$

Figure 7 shows histogram plots of the same dataset processed using a different method each time. Figure 7A shows a histogram of the enrichment scores after using *Equation 2*. The histogram shows a bimodal distribution. The left peak represents the loss-of-function residues at the interface with the effector (Raf-RBD) or ligands, and the central peak represents the remaining residues. In the example, the center of the main peak of the distribution is at ~ -0.1. While close to zero, enrichment scores vary based on the ratio of selected and unselected DNA counts. That is, if the selected sample has a ten-fold increase in counts because more sample was loaded into the Miseq chip, then the enrichment scores will carry that factor over. This is an important consideration when comparing replicates or other datasets. For that reason, the data needs to be standardized. While the choice of standardization technique is arbitrary, it is crucial to use the same technique throughout all the datasets.

The first method, named "counts", standardizes the data by subtracting the ratio of total counts of selected and unselected samples (*Equation 3*), and can also be expressed as the $\log_{10}$ frequency (*f*) of observing each amino acid at each position (*Equation 4*):

Equation 3: $$\Delta E_x^i = \log_{10}\left[\frac{c_i^{x,sel}}{c_i^{x,unsel}}\right] - \log_{10}\left[\frac{\sum_{i=1}^{n} c_i^{x,sel}}{\sum_{i=1}^{n} c_i^{x,unsel}}\right]$$

Equation 4: $$\Delta E_x^i = \log_{10}\left[\frac{f_i^{x,sel}}{f_i^{x,unsel}}\right]$$

This method brings the center of the right distribution to ~-0.2 (Figure 7B), which brings most of the variants to the loss-of-function range. The second method, "wt allele only", subtracts the$\log_{10}$ ratio of wild-type counts in the selected and unselected samples rather than subtracting the ratio of counts (*Equation 5*). The wild-type allele is the DNA template used to generate the NNS library.

Equation 5: $$\Delta E_x^i = \log_{10}\left[\frac{c_i^{x,sel}}{c_i^{x,unsel}}\right] - \log_{10}\left[\frac{c_i^{wt,sel}}{c_i^{wt,unsel}}\right]$$

The resulting kernel is shown in Figure 7C. Each of the three histograms has a different mode, which will add variance to the normalization process. This method has been used in many papers (Bandaru et al. 2017; McLaughlin et al. 2012). Indeed, when this project started, the "wt allele only" method was the go-to method in the Kuriyan lab. However, we started observing inconsistent results between experiments. After investigating, we discovered this method was highly susceptible to outliers, hence the inconsistency between experiments. This section will cover in further detail the reasons for these inconsistencies (Figure 8). This roadblock was an impetus for the creation of the open-source Python software.

The "wt synonymous" method solves the problems that the "wt allele only" method presents. It works by subtracting the mean, mode, median of the $\log_{10}$ ratio of the frequencies of synonymous variants with the wild-type allele (*Equation 6*).

Equation 6: $$\Delta E_x^i = \log_{10}\left[\frac{c_i^{x,sel}}{c_i^{x,unsel}}\right] - \frac{\sum_{j=1}^{n} \log_{10}\left[\frac{c_j^{wt,sel}}{c_j^{wt,unsel}}\right]}{n}$$

Figure 7D shows the histograms for the three replicates. The center of the right peak falls at precisely zero.

The "population" method subtracts the mean, mode, or median of the $\log_{10}$ ratio of the selected and unselected counts for the entire population of variants (*Equation* 7):

Equation 7:
$$\Delta E_x^i = \log_{10}\left[\frac{c_i^{x,sel}}{c_i^{x,unsel}}\right] - \frac{\sum_{i=1}^{n}\log_{10}\left[\frac{c_i^{x,sel}}{c_i^{x,unsel}}\right]}{n}$$

The results of using the "population" method can be seen in Figure 7E, where the center of the main peak is also at zero. Given that the distribution of the Ras data is Gaussian, using mean, mode, or median offers the same results. In other types of distributions, they may differ.

A variant of this method is the "kernel" method. First, a Gaussian kernel is fitted to the enrichment score distribution (*Equation 8*). The kernel density estimation (KDE) smooths each data point X_i into a density function and then sums each density function to obtain the final density estimate; n is the number of data points; h is the smoothing bandwidth. Then, the mean, mode, or median of the KDE is subtracted to standardize the data.

Equation 8:
$$KDE = \frac{1}{nh}\sum_{i=1}^{i=n}\frac{1}{\sqrt{2\pi}}\, e^{-\frac{1}{2}\left(\frac{X_i - x}{h}\right)^2};\ X_i = \log_{10}\left[\frac{c_i^{x,sel}}{c_i^{x,unsel}}\right]$$

Figure 7F shows the histograms that result from using the "kernel" method. The main peak is, again, centered at zero. As in many other data-science applications, the best-performing methods change depending on the dataset. If a result is the byproduct of a specific normalization method, then it may be incorrect. *Mutagenesis-visualization* offers multiple options to process the datasets, allowing the user to compare and discard outlier methods.

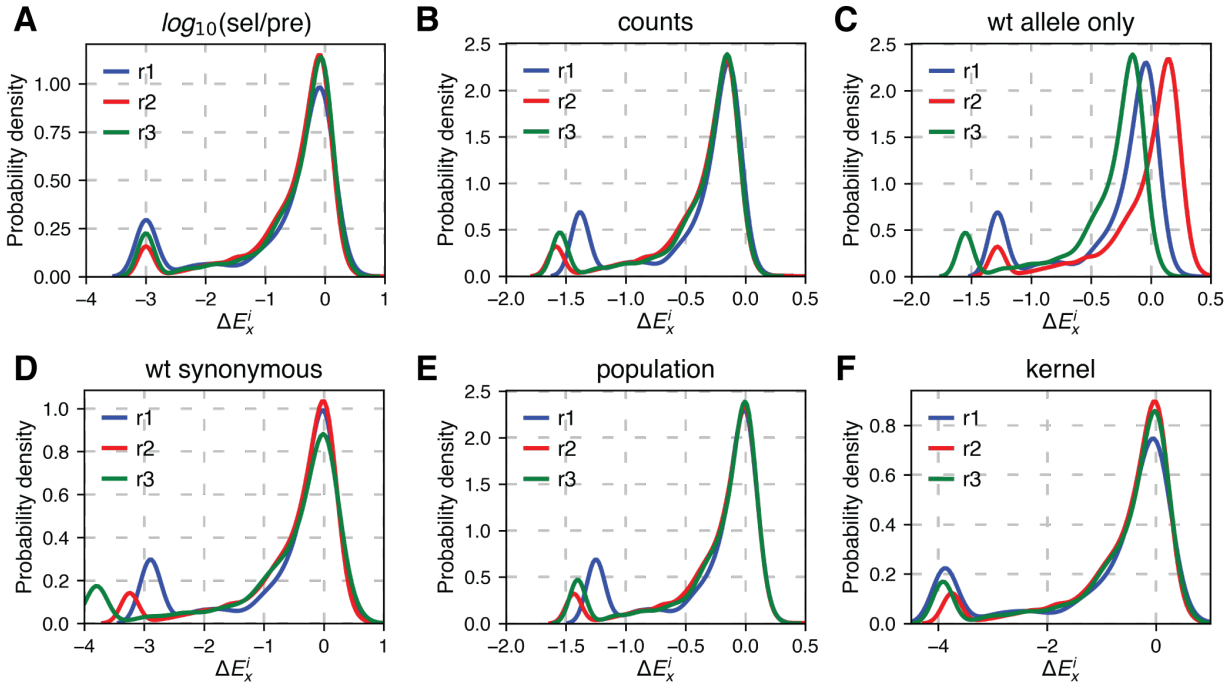

Figure 7. Histograms of the three replicates of the first sub-library of the H-Ras$^{2\text{-}166}$ bacterial-two-hybrid screen in the presence of Raf-RBD. (**A**) Enrichment scores calculated as the $\log_{10}$ of the ratio of counts for each variant from the selected and pre-selected samples. The following figures show the enrichment scores with a different way of standardizing the data. (**B**) The enrichment scores have been standardized by subtracting the ratio of total counts from the selected and pre-selected samples. (**C**) The enrichment scores have been standardized by setting the wild-type allele enrichment score to zero. The appearance of this graph has been dramatized to exemplify the shortcomings of this method. (**D**) The enrichment scores have been standardized using the mode of the enrichment scores of all the synonymous wild-type alleles. (**E**) The enrichment scores have been standardized by setting the mode of the histogram to zero. (**F**) The enrichment scores have been standardized by setting the center of the fitted kernel to zero.

As discussed earlier, the "wt allele only" method is susceptible to outliers. In these types of experiments, there are different sources of error, such as from the next-generation sequencing procedure, PCR amplification steps, counting statistics, or experimental conditions during the selection process (e.g., changes in antibiotic concentration from replicate to replicate). In addition, because the "wt allele only" relies on a single DNA chain as the reference to center the data, a small noise factor will dramatically impact the data processing. Hence, in Figure 7C each of the three replicates was shifted. This drawback can be overcome by increasing the sample size with more data points to make the statistics more robust. A way to obtain more data points is to use the synonymous wild-type alleles. These alleles comprise all the sequences in the DNA library that differ in DNA sequence to the wild-type gene used as the template to create the library, yet they translate to the same protein. There are more than a hundred synonymous wild-type alleles in the Ras libraries used in this paper, making the "wt synonymous" method more reliable than the "wt allele only" method. Figure 8A shows a histogram of the wild-type allele enrichment scores for the three replicates analyzed in Figure 7. Figure 8B shows scatter plots of all the pairwise combinations of the three replicates. The low R^2 values indicate no correlation between replicates, ruling out the possibility that different DNA sequences have a different expression level affecting the dynamics of the experiment.

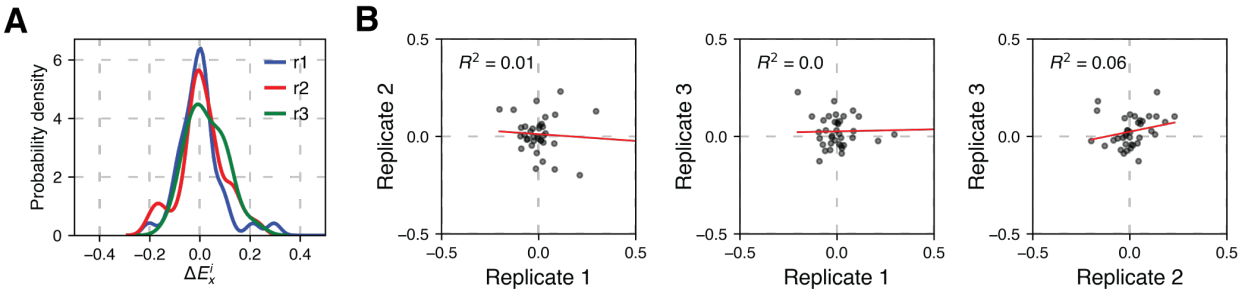

Figure 8. Enrichment scores of the synonymous wild-type alleles of the three replicates of the first sub-library of the H-Ras$^{2\text{-}166}$ bacterial-two-hybrid screen in the presence of Raf-RBD. The data normalization was conducted using the method shown in Figure 7D. (**A**) Kernel density estimation of the three replicates. (**B**) Correlation plots and R^2 values of the three replicates.

Summary

Site-saturation mutagenesis experiments have been transformative in our study of protein function. However, despite the rich data generated from such experiments, current tools for processing, analyzing, and visualizing the data offer a limited set of visualization tools that are difficult to customize. Furthermore, usage of the tools requires extensive experience and programming knowledge, slowing the research process for those in the biological field who are unfamiliar with programming.

Mutagenesis-visualization fills a need in the Python and scientific community for user-friendly software that streamlines the pairwise site-saturation mutagenesis bioinformatics pipeline, where each of the graphs can be generated with a one-line command. While the workflow allows for end-to-end analysis of the pre-processed datasets, its modularity makes it compatible with existing software platforms used to process site-saturation mutagenesis data. The plots can be rendered as native *matplotlib* objects (easy to stylize) or *Plotly* objects (interactive graphs). Additionally, the software offers the possibility to visualize the datasets on *Pymol*. The software can be installed from *PyPI* or *GitHub* using the *pip* package manager and is compatible with Python ≥ 3.8. The documentation can be found at *readthedocs,* and the source code can be found on *GitHub*.

Bibliography

Araya, C.L. and Fowler, D.M. 2011. Deep mutational scanning: assessing protein function on a massive scale. *Trends in Biotechnology* 29(9), pp. 435–442.

Bandaru, P., Shah, N.H., Bhattacharyya, M., et al. 2017. Deconstruction of the Ras switching cycle through saturation mutagenesis. *eLife* 6.

Bloom, J.D. 2015. Software for the analysis and visualization of deep mutational scanning data. *BMC Bioinformatics* 16, p. 168.

Bolger, A.M., Lohse, M. and Usadel, B. 2014. Trimmomatic: a flexible trimmer for Illumina sequence data. *Bioinformatics* 30(15), pp. 2114–2120.

Dandage, R. and Chakraborty, K. 2017. dms2dfe: Comprehensive Workflow for Analysis of Deep Mutational Scanning Data. *The Journal of Open Source Software* 2(20), p. 362.

Dou, J., Vorobieva, A., Sheffler, W., et al. 2018. De Novo Design Of A Fluorescence-Activating B-Barrel. *Zenodo*.

Faure, A.J., Schmiedel, J.M., Baeza-Centurion, P. and Lehner, B. 2020. DiMSum: an error model and pipeline for analyzing deep mutational scanning data and diagnosing common experimental pathologies. *Genome Biology* 21(1), p. 207.

Fernandes, J.D., Faust, T.B., Strauli, N.B., et al. 2016. Functional segregation of overlapping genes in HIV. *Cell* 167(7), p. 1762–1773.e12.

Fowler, D.M. and Fields, S. 2014. Deep mutational scanning: a new style of protein science. *Nature Methods* 11(8), pp. 801–807.

Gaidatzis, D., Lerch, A., Hahne, F. and Stadler, M.B. 2015. QuasR: quantification and annotation of short reads in R. *Bioinformatics* 31(7), pp. 1130–1132.

Harris, C.R., Millman, K.J., van der Walt, S.J., et al. 2020. Array programming with NumPy. *Nature* 585(7825), pp. 357–362.

Hilton, S.K., Huddleston, J., Black, A., et al. 2020. dms-view: Interactive visualization tool for deep mutational scanning data. *Journal of open source software* 5(52).

Hiraga, K., Mejzlik, P., Marcisin, M., et al. 2021. Mutation maker, an open source oligo design platform for protein engineering. *ACS synthetic biology [electronic resource]* 10(2), pp. 357–370.

Hunter, J.D. 2007. Matplotlib: A 2D Graphics Environment. *Computing in science & engineering* 9(3), pp. 90–95.

Li, C., Qian, W., Maclean, C.J. and Zhang, J. 2016. The fitness landscape of a tRNA gene. *Science* 352(6287), pp. 837–840.

Livesey, B.J. and Marsh, J.A. 2020. Using deep mutational scanning to benchmark variant effect predictors and identify disease mutations. *Molecular Systems Biology* 16(7), p. e9380.

Love, M.I., Huber, W. and Anders, S. 2014. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology* 15(12), p. 550.

McKinney, W. 2010. Data structures for statistical computing in python. In: *Proceedings of the 9th Python in Science Conference*. Proceedings of the python in science conference. SciPy, pp. 56–61.

McLaughlin, R.N., Poelwijk, F.J., Raman, A., Gosal, W.S. and Ranganathan, R. 2012. The spatial architecture of protein function and adaptation. *Nature* 491(7422), pp. 138–142.

Melnikov, A., Rogov, P., Wang, L., Gnirke, A. and Mikkelsen, T.S. 2014. Comprehensive mutational scanning of a kinase in vivo reveals substrate-dependent fitness landscapes. *Nucleic Acids Research* 42(14), p. e112.

Newberry, R.W., Leong, J.T., Chow, E.D., Kampmann, M. and DeGrado, W.F. 2020. Deep mutational scanning reveals the structural basis for α-synuclein activity. *Nature Chemical Biology* 16(6), pp. 653–659.
Nov, Y. 2012. When second best is good enough: another probabilistic look at saturation mutagenesis. *Applied and Environmental Microbiology* 78(1), pp. 258–262.
Pines, G., Fankhauser, R.G. and Eckert, C.A. 2020. Predicting drug resistance using deep mutational scanning. *Molecules (Basel, Switzerland)* 25(9).
Rubin, A.F., Gelman, H., Lucas, N., et al. 2017. A statistical framework for analyzing deep mutational scanning data. *Genome Biology* 18(1), p. 150.
Schrödinger, L.L.C. 2015. The {PyMOL} Molecular Graphics System, Version~1.8.
Shin, H. and Cho, B.-K. 2015. Rational protein engineering guided by deep mutational scanning. *International Journal of Molecular Sciences* 16(9), pp. 23094–23110.
Stiffler, M.A., Hekstra, D.R. and Ranganathan, R. 2015. Evolvability as a function of purifying selection in TEM-1 β-lactamase. *Cell* 160(5), pp. 882–892.
Subramanian, S., Gorday, K., Marcus, K., et al. 2021. Allosteric communication in DNA polymerase clamp loaders relies on a critical hydrogen-bonded junction. *eLife* 10.
Tang, L., Gao, H., Zhu, X., Wang, X., Zhou, M. and Jiang, R. 2012. Construction of "small-intelligent" focused mutagenesis libraries using well-designed combinatorial degenerate primers. *Biotechniques* 52(3), pp. 149–158.
Tripathi, A. and Varadarajan, R. 2014. Residue specific contributions to stability and activity inferred from saturation mutagenesis and deep sequencing. *Current Opinion in Structural Biology* 24, pp. 63–71.
Wrenbeck, E.E., Faber, M.S. and Whitehead, T.A. 2017. Deep sequencing methods for protein engineering and design. *Current Opinion in Structural Biology* 45, pp. 36–44.
Zhang, J., Kobert, K., Flouri, T. and Stamatakis, A. 2014. PEAR: a fast and accurate Illumina Paired-End reAd mergeR. *Bioinformatics* 30(5), pp. 614–620.