Implementation of a Model System to Probe the Relationship Between Protein Higher Order Structure and HX-MS Data Significance

By

Hayley Blockinger
B.Sc., California State Polytechnic University, Pomona, 2018

Submitted to the graduate degree program in Chemistry and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

Co-Chair: David Weis, Ph.D.

Co-Chair: Heather Desaire, Ph.D.

Meredith Hartley, Ph.D.

Date Defended: 02 December 2022

The dissertation committee for Hayley Blockinger certifies that this is the approved version of the following dissertation:

Implementation of a Model System to Probe the Relationship Between Protein Higher Order Structure and HX-MS Data Significance

Co-Chair: David Weis, Ph.D.

Co-Chair: Heather Desaire, Ph.D.

Date Approved: 02 December 2022

Abstract

Hydrogen exchange-mass spectrometry (HX-MS) is a method that can be used to study the higher order structure (HOS) of proteins. A protein is exposed to a deuterated solvent and amide hydrogens will exchange with deuterium depending on their solvent accessibility. The uptake in deuterium over time causes a shift in the mass of the protein, detectable by mass spectrometry. Differential HX is used to compare the uptake values between a control protein system versus an altered protein system to make inferences on what the difference in mass uptake indicates about the HOS. One such comparison group is comparing the HOS of originator biologics to biosimilars. Biologics are patented protein therapeutics that are marketed at a high cost due to the cost of clinical trials. The Patient Protection and Affordable Care Act was passed to combat the high prices of drugs. If a company can prove that a biosimilar is structurally and functionally similar to an originator biologic, then the biosimilar does not have to go through all the clinical trials like the originator biologic, reducing the cost of producing the biosimilar, therefore marketing it at an affordable cost.

One area in method validation that is lacking in the HX-MS field is relating the significance of HX data to important changes in the protein HOS. To probe the relationship between HX data significance and important changes in HOS, V_HH-F5 and ricin toxin subunit A (RTA) was selected as a model system. The wild type and seven single point mutations of the V_HH-F5 protein were successfully expressed and ELISA was conducted to see if the point mutation had a significant impact on the relative binding affinity toward RTA, which can be related back to the HOS of the mutants. Based on the calculations by the R code developed, it was determined that four mutants

had significant differences in binding affinity than the wild type– P122A, I176L, D194E, and V225A – implying that the point mutation has an impact on the HOS of the protein. Future work would determine if these significant differences are detectable by HX-MS experiments.

Acknowledgements

I'd like to start by thanking Dr. David Weis for accepting me into his group. I've learned a lot under his mentorship, and he exposed me to experiences that helped me shape my ideal career trajectory in the science community. He was also very supportive when I decided that I needed to change paths and took the time to listen to me and ask questions to make sure that my change in course would not impact my career goals.

I would also like to thank the current and past members of the Weis lab. I would like to thank Dr. Chamalee Gamage for acting as my guide in the lab for my project. I would like to thank Emran Haque, Michael Wrigley, and Sachini Karunaratne for supporting me and providing conversation and laughter throughout my time in the lab. I would also like to thank the undergraduates I mentored – Caitlin McEntee and Julio Soe. Without the chance to mentor you both, I do not think I would be on the path I'm on right now.

I would like to thank my committee – Dr. Heather Desaire and Dr. Meredith Hartley – for taking the time to be on my committee, but also for being people I could talk to throughout my time at KU. I appreciate your insight and candor.

I would like to thank my friends that I've made in Kansas for being another support system and for helping me find time to relax and enjoy life outside of graduate work. I would also like to thank my family for their never-ending love and support from home.

Finally, I would like to thank my new family in Kansas – my fiancé Garrick Valverde. Thank you for your constant support for my graduate work and career aspirations, for making me laugh, and reminding me that I can do anything that I set my mind to.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction and Background to the Thesis

1.1 Introduction

Protein therapeutics on the market have been developed as medicinal remedies for many illnesses, such as cancer, diabetes, and autoimmune diseases. These therapeutics are patented as novel medicines and are classified as biologics. Millions of dollars are invested into the clinical trial process to get biologics on the market, resulting in an expensive drug. To combat the high cost of these biologics, the United States government took action and passed the Patient Protection and Affordable Care Act (ACA). One of the goals of the ACA is to reduce the cost of biologics on the market through the development of biosimilars. Biosimilars are protein therapeutics that can be proved to be structurally and functionally similar to an originator biologic. The originator biologic is defined as the initial biopharmaceutical product that was developed and released into the medicinal market.[1] As the patents on originator biologics expire, biosimilars can be produced without infringing on the patented rights of the originator biologics.[2] With the ACA, if a company can provide results that prove that a biosimilar is highly similar to an approved biologic,[2,3] the company does not have to go through all the stages of clinical trials by piggybacking off the work that has been done by the company who produced the originator biologic.[1,4,5] This greatly reduces the cost of producing the biosimilar, therefore the biosimilar can be listed on the market at a lower cost than the biologic.

There is a push to validate or develop new methods that can be used to confirm structure and function similarity between biologics and biosimilars.[6] Current methods available are X-ray crystallography and NMR,[7] but both have their downfalls. X-ray crystallography can be used to examine higher order structure, but only if the protein of

interest can be crystallized.[6,8,9] The crystallization process can also be lengthy, so even if the protein can be crystallized, using X-ray crystallography for a routine analysis is not practical.[6,8,9] To utilize NMR, proteins would have to have large amounts of active nuclei present that are not solely ^{1}H.[6] NMR also has low sensitivity and has difficulty with analyzing proteins with a molecular weight over 40 kDa.[6–9] Hydrogen exchange mass spectrometry (HX-MS) is a method that can be used to study protein structure and protein binding with ligands, inhibitors, or other binding interactions. Samples can be analyzed at a faster rate than X-ray crystallography, the instrumentation can be automated, and HX-MS can detect small changes in the protein structure.[6–9] HX-MS can have analysis times of hours to days due to the complicated data interpretation,[6] but computer software has been developed to help with the data analysis of the HX-MS data.

While there is a push to develop new methods to compare biosimilars to originator biologics, scientists need to know what the detection limits are for each method to determine significance. Work has been done to develop statistical methods to determine the significance of HX-MS measurements.[8–11] Specifically in the Weis lab, these works detail the statistics involved in determining a significant value in differential HX values[10] and evaluating the limit of detection by spiking different, known amounts of a perturbed protein into a wild type protein to determine when HX-MS detects a significant difference.[9] Now equipped with the tools to determine if an HX measurement is significant and how to evaluate the limit of detection for an HX-MS method, the next steps forward are to investigate (1) if all significant HX differences are important to the protein structure and (2) if all important differences in the protein structure can be

detected by HX-MS. To probe these questions, a model system will have to be selected to generate results that would provide insight on these questions.

The model developed by Hageman and Weis to probe the limit of detection of HX-MS introduced small amounts of a perturbed mutant protein into a wild type protein to determine what percent of the perturbed protein would be detected by HX-MS.[9] This model was a heterogenous mixture, since there were two pools of protein that were combined to create one sample. This model would not be useful in answering the questions proposed in the previous paragraph since two sets of proteins were introduced, with different higher order structures, one of which is known to be significantly different than the other. The model that would make more sense to develop would be a homogenous model, where the sample makeup is just one protein. Manipulating the amino acid sequence of that one protein could alter the structure, and the HX data generated could provide insight into the proposed questions on detectability of important differences. To probe the sensitivity of HX measurements, single-point mutations of a model protein would make more sense than introducing multiple mutations that would increase the chances of substantially perturbing the higher order structure. The HX data generated from the mutant could then be compared to the HX data from the wild type protein to determine if the structure of the mutant has changed compared to the wild type protein. Another aspect of the model selection process that has to be considered is the ability to answer question (1) through an alternative way to look at the higher order structure of the model system. A way to investigate question (1) is by comparing crystal structures between the point mutated protein and the wild type to see if the significant values from HX-MS show a change in the higher order structure

between the crystallized proteins. The model should also be developed to address how the important differences in structure be determined for question (2). One way to test this is to select a model that has a high binding affinity to an antigen, and if that binding is disrupted when compared to the wild type, then the point mutation introduced could have had an impact on the higher order structure of the model. When the binding affinity results from the point mutations are compared to HX-MS data, conclusions can be drawn to determine if the difference in higher order structure, indicated by the change in binding affinity, is detected by HX-MS.

The model protein that was selected was V_HH-F5. This is a camelid nanobody that binds to the A subunit of ricin toxin (RTA),[12] that has been structurally characterized via X-ray crystallography.[13] V_HH-F5 has also been characterized to have strong binding to RTA subunit A. The dissociation constant for the V_HH-F5-RTA construct has a value of 2.2 nM,[13] indicating that a minimal amount of the protein would dissociate from the antigen, therefore, there is a strong binding interaction between the two entities. Protocols were developed for V_HH-F5 protein expression previously[13] and were a starting point for this thesis to express single-point mutants of V_HH-F5. The mutations introduced were in regions distal or proximal to the binding site of V_HH-F5 to RTA, but not in the binding site itself to prevent disruptions in binding due to a change in the paratope when binding affinity assays were conducted. A series of single-point mutated V_HH-F5 proteins were expressed, and relative binding affinity assays were conducted with long-term goal of evaluating HX-MS data to answer the questions of if all important differences in protein structure are detected by HX-MS and are all significant HX data important to the structure of the protein.

1.2 Hydrogen exchange Theory

Hydrogen exchange is a technique where a hydrogen atom of one isotope on a protein is exchanged for another isotopic form of hydrogen. The Weis Lab specifically focuses on hydrogen-deuterium exchanges, where a labile amide hydrogen is replaced by a deuterium from the deuterated solvent the protein is placed in. The extent and location of deuteration of the protein of interest is influenced by a series of factors. The first is identifying which hydrogens in a protein are labile, meaning that they can exchange with the deuterium in the solvent, but the exchanges are also on a measurable time scale. These hydrogens in proteins are the amide hydrogens that reside in the protein backbone, aside from proline.[7] The reason amide hydrogens are labile and detectable is because of the difference in electronegativity between the hydrogen and nitrogen, where the difference is large enough to be a polar bond, but not too large where the hydrogens readily dissociate into the solvent.

Other factors that impact deuteration rates are neighboring side chains, solvent accessibility, the extent of local hydrogen bonding, and the pH of the solvent.[7,14–16] Sidechains can be electron withdrawing or electron donating, impacting the acidity of the amide hydrogens in the backbone. If a hydrogen is buried in the protein structure versus on the surface of the protein structure, the deuterated solvent would have a more difficult time penetrating the protein to reach the buried hydrogens.[7,15] A buried hydrogen could also be exposed if the protein unfolds, but this is governed by the dynamics of the protein.[7,14,15,17] The protein dynamics would impact the rate that the buried hydrogens can readily exchange with the solvent compared to a hydrogen that is on the surface and readily exposed to the deuterated solvent. Amide hydrogens in the

protein can also participate in hydrogen bonding with other atoms in the protein as well as the solvent. When these hydrogen bonds are formed, the bond has to be broken before the hydrogen can exchange with the deuterated solvent, impacting the rate of deuteration.[7,15] Finally, the pH of the solution impacts the rate of deuteration of the protein. HX can be acid or base catalyzed, meaning that the amide either starts HX by taking deuterium from the solvent or the deuterated solvent takes a hydrogen from the amide.[7,15,16] Experiments performed show that the rates for HX that are base catalyzed versus acid catalyzed are magnitudes higher for base catalyzed HX than acid catalyzed.[16] The results from those experiments would also indicate that the pH of the solution would have an effect on the HX rate for the amide hydrogens in a protein.

A basic flow of a hydrogen-exchange experiment is as follows. An aliquot of the concentrated protein of interest is transferred into a deuterated solvent. The percentage of volume of the deuterated solvent to the volume of protein transferred is usually 90% or higher to ensure that the protein is saturated with deuterated solvent and lessen the chances of a phenomenon called back exchange.[18] Back exchange occurs when the protein of interest exchanges deuteriums the protein took up with hydrogens that might be present in the solvent. This phenomenon is especially prevalent in local experiments where peptides are cleaved and generate a primary amine.[7] By limiting the amount of hydrogens available in the solvent for the protein to exchange with, back exchange can be reduced. The protein exchanges protons with deuterons in the solvent for a predetermined period of time called the labeling time.[7] The labeling time points are selected over a range of short and long time scales to try to observe the dynamics of the protein in sites that cover faster and slower rates of exchange.[18] These labeling

times range from seconds to days, usually with at least five time points selected, to capture the protein dynamics of fast-exchanging and slow-exchanging regions. This is important in differential HX experiments, where comparisons are being made between two intact proteins. By measuring HX at multiple time points, a more complete picture of the dynamics can be captured to evaluate differences in HX rates versus selecting less time points that would not be able to show how the difference in rate changes over labeling time. After the labeling time has concluded, the exchange is quenched with a buffer that significantly reduces the pH and temperature to a level where the rate of hydrogen/deuterium exchange is at a minimum.[7] This range resides between a pH of 2-3 and the temperature is reduced to 0 °C.[16,18] At this point in time, the quenched protein solution rests at 0 °C until it is analyzed via liquid chromatography-mass spectrometry (LC-MS), or it is flash frozen and stored at -80 °C until it is thawed and analyzed. When coupled with mass spectrometry, the shift in mass can be tracked over labeling times and interpreted to gather information about the protein of interest.

1.3 HX-MS and Data Analysis

Hydrogen exchange can be coupled with mass spectrometry to track how the mass changes in the protein over a series of labeling times, known as the HX rate. Each exchange of hydrogen for a deuterium increases the mass by 1 Dalton.[7] The mass changes can be tracked globally (over the full protein) or locally (over a series of peptides). Global experiments look at how the dynamics or structure of the intact protein change over a series of labeling time points. After hydrogen exchange, the intact protein sample is directed through a liquid chromatography system prior to mass analysis using mass spectrometry. A local experiment incorporates a digestion to cleave the protein

into peptides after the protein has been quenched. This allows the researcher to look at deuterium uptake in local regions of the protein to make inferences on the protein's structure and dynamics at a local level. The digestion can either occur in the sample vial by adding an enzyme directly into the quenched solution or by adding an additional column in the liquid chromatography work flow that would contain a stationary phase with an enzyme to digest the protein into peptides prior to separation.[7] This enzyme must be able to work in acidic conditions, as the pH of the quench buffer and/or the eluent used in the LC system are generally acidic.[7]

The undeuterated data collected contains mass spectra with a binomial distribution of m/z values around the centroid mass. The centroid mass is the m/z value that is used to determine the extent of deuterium uptake over the labeling times. The deuterated data contains mass spectra with centroid peaks that are of the same or higher m/z value to the undeuterated protein/peptide. The difference in m/z between the centroid peak of the deuterated time point versus the undeuterated time point is the average number of deuterons that were taken up by the protein/peptide. The extent of deuteron uptake can be used to draw inferences on the structure and dynamics of the protein/peptide. A high number of deuterons could indicate that the peptide is on the outer surface of the protein and is not involved in extensive hydrogen bonding. A low number of deuterons could indicate that the peptide is in the hydrophobic core of the protein or is engaged in a hydrogen bonding network.

Differential HX-MS is when the change in deuterium uptake for one protein state, a reference state, is compared to the change in deuterium uptake for another protein state. These experiments are useful to see how a change in the protein state impacts

the deuteration of the protein. Common systems studied are a wild type versus a mutant or a protein without and with a ligand or bound entity. If the deuterium uptake over the labeling times for a protein is insignificant compared to the reference state, then whatever change was made does not have an impact on the HX rate. If the deuterium uptake over labeling times is significantly different from the reference state, then the change does have an impact of the HX rate. For the model selected for this work, a mutant would be compared to a wild type protein in HX-MS experiments. A significant difference in HX rate for the mutant would indicate that the point mutation introduced has impacted the higher order structure, whereas an insignificant difference would mean that the point mutation did not measurably affect the higher order structure. Once the differential HX-MS data deems a difference between the higher order structure of the mutant and wild type as significant or not, the results can be compared to binding affinity assays and solved crystal structures to answer the questions of are all significant HX-MS data important to the higher order structure and are important structural differences detected by HX-MS.

1.4 ELISA Principles

One of the ways to identify mutants that cause an important difference in structure, which is relevant to determining that these differences can be detected by HX-MS, is by developing a functional assay. Binding assays are screening tools that are commonly used to evaluate binding affinity, inhibition, and other binding interactions. One common type of binding assay is an enzyme-linked immunosorbent assay (ELISA). ELISA is a common procedure used to evaluate binding affinity between antigens and antibodies. ELISAs can be easily adaptable to any antigen-antibody pairing, making ELISAs heavily

used in biological studies.[19] There are many different ways to adapt ELISA procedures based on the functional characteristic that a lab is trying to measure, but for the purposes of explaining ELISA principles, this section will focus on a direct ELISA to walk through the steps that are usually taken and the reasoning behind each step.

A direct ELISA[20] is the simplest form of an ELISA since the antigen is bound directly onto the solid support, the antibody conjugated with an enzyme for detection is added, and a substrate is added to spectrophotometrically detect the amount of antibody directly bound to the antigen. There are a series of steps involved in an ELISA, which are coating, washing, blocking, incubation, and detection. The coating step introduces the antigen so the antigen can bind to the solid support through passive adsorption or a binding interaction with the solid support. Washing steps remove any of the previous solution from the wells to prepare the wells for steps to follow. Blocking ensures that the rest of the solid support that does not contain the antigen is coated to prevent non-specific binding of other proteins. Incubation is the process where the secondary antibody is added that interacts with the antigen for a set period of time. Detection involves a reaction in the ELISA solution that generates a change in the properties of the solution that can be detected. Examples of this are a product generated that can absorb light, fluoresce, or luminesce and are detected spectrophotometrically.

For detection of a direct ELISA, the antibody introduced is enzyme conjugated, meaning that it already contains the enzyme that is used for the detection method as a part of the antibody system. Once the solid support has been incubated with the enzyme-conjugated antibody for a set time, the substrate that the enzyme utilizes for detection can be added. After the reaction time, the solutions are quenched to stop the reaction

from developing any further. If the reaction was not quenched, then the enzymes would keep converting the substrate to a product and could result in signals that do not correlate to the binding interactions that are occurring on the plate, an example of which is if there is a complete conversion of the substrate to the product. The conversion of substrate to product can be detected by spectrophotometry at a wavelength that is specific to the enzyme-substrate-quench reaction used. This is commonly measured with a plate reader. The detected values are correlated to concentrations of the antibody added into the wells or a dilution factor that was conducted down a series of wells. Plotting the binding curve can provide insight into the binding affinity between the antigen and the antibody.

Since the model generated for this work consists of a camelid nanobody (V_HH-F5) and RTA, an antigen it binds to, an ELISA can be used to calculate the relative binding affinity between the wild type and mutant V_HH-F5 proteins to see if the data generated will corroborate the same significant or insignificant differences from the differential HX-MS data collected.

1.5 Overview of the Thesis

To summarize, the goal of this work is to develop a model system that can be used to understand if significant differences in HX data are important in the higher order structure of a protein and if important differences in the protein higher order structure are detectable by HX-MS. Chapter 2 discusses the expression of the V_HH-F5 wild type and mutants and the process of confirming the expression of each V_HH-F5 protein. Chapter 3 consists of the ELISA method developed in the Weis lab, the discussion of the single-point mutant impacts on structure, and the analysis of the relative binding

affinity values calculated. Chapter 4 walks through the R code that was programmed to conduct the calculations for the relative binding affinity and generate figures, as well as the explanation of the statistics methods selected.

1.6 References

1. Covic, A.; Kuhlmann, M. K. Biosimilars: Recent Developments. *Int. Urol. Nephrol.* **2007**, *39* (1), 261–266. https://doi.org/10.1007/s11255-006-9167-5.

2. Schellekens, H.; Moors, E. Clinical Comparability and European Biosimilar Regulations. *Nat. Biotechnol.* **2010**, *28* (1), 28–31. https://doi.org/10.1038/nbt0110-28.

3. Nick, C. The US Biosimilars Act: Challenges Facing Regulatory Approval. *Pharm. Med.* **2012**, *26* (3), 145–152. https://doi.org/10.1007/BF03262388.

4. Zuñiga, L.; Calvo, B. Biosimilars Approval Process. *Regul. Toxicol. Pharmacol.* **2010**, *56* (3), 374–377. https://doi.org/10.1016/j.yrtph.2009.11.004.

5. Kozlowski, S.; Woodcock, J.; Midthun, K.; Behrman Sherman, R. Developing the Nation's Biosimilars Program. *N. Engl. J. Med.* **2011**, *365* (5), 385–388. https://doi.org/10.1056/NEJMp1107285.

6. Berkowitz, S. A. Analytical Tools for Characterizing Biopharmaceuticals and the Implications for Biosimilars. *DRUG Discov.* 14.

7. Konermann, L.; Pan, J.; Liu, Y.-H. Hydrogen Exchange Mass Spectrometry for Studying Protein Structure and Dynamics. *Chem Soc Rev* **2011**, *40* (3), 1224–1234. https://doi.org/10.1039/C0CS00113A.

8. Houde, D.; Berkowitz, S. A.; Engen, J. R. The Utility of Hydrogen/Deuterium Exchange Mass Spectrometry in Biopharmaceutical Comparability Studies. *J. Pharm. Sci.* **2011**, *100* (6), 2071–2086. https://doi.org/10.1002/jps.22432.

9. Hageman, T. S.; Weis, D. D. A Structural Variant Approach for Establishing a Detection Limit in Differential Hydrogen Exchange-Mass Spectrometry Measurements. *Anal. Chem.* **2019**, *91* (13), 8017–8024. https://doi.org/10.1021/acs.analchem.9b01326.

10. Hageman, T. S.; Weis, D. D. Reliable Identification of Significant Differences in Differential Hydrogen Exchange-Mass Spectrometry Measurements Using a Hybrid Significance Testing Approach. *Anal. Chem.* **2019**, *91* (13), 8008–8016. https://doi.org/10.1021/acs.analchem.9b01325.

11. Weis, D. D. Comment on Houde, D.; Berkowitz, S. A.; Engen, J. R., The Utility of Hydrogen/Deuterium Exchange Mass Spectrometry in Biopharmaceutical Comparability Studies. J. Pharm. Sci. 2011, 100, 2071-2086. *J. Pharm. Sci.* **2019**, *108* (2), 807–810. https://doi.org/10.1016/j.xphs.2018.10.010.

12. Vance, D. J. Stepwise Engineering of Heterodimeric Single Domain Camelid VHH Antibodies That Passively Protect Mice from Ricin Toxin*. **2013**, *288* (51), 10.

13. Rudolph, M. J.; Vance, D. J.; Cassidy, M. S.; Rong, Y.; Shoemaker, C. B.; Mantis, N. J. Structural Analysis of Nested Neutralizing and Non-Neutralizing B Cell Epitopes on Ricin Toxin’s Enzymatic Subunit: Nested B Cell Epitopes on

RTA. *Proteins Struct. Funct. Bioinforma.* **2016**, *84* (8), 1162–1172. https://doi.org/10.1002/prot.25062.

14. Persson, F.; Halle, B. How Amide Hydrogens Exchange in Native Proteins. *Proc. Natl. Acad. Sci.* **2015**, *112* (33), 10383–10388. https://doi.org/10.1073/pnas.1506079112.

15. Skinner, J. J.; Lim, W. K.; Bédard, S.; Black, B. E.; Englander, S. W. Protein Dynamics Viewed by Hydrogen Exchange: Protein Dynamics from Hydrogen Exchange. *Protein Sci.* **2012**, *21* (7), 996–1005. https://doi.org/10.1002/pro.2081.

16. Jensen, P.; Rand, K. Hydrogen Exchange: A Sensitive Analytical Window into Protein Conformation and Dynamics. In *Hydrogen Exchange Mass Spectrometry of Proteins: Fundamentals, Methods, and Applications*; John Wiley & Sons, Ltd, 2016; pp 1–18.

17. Skinner, J. J.; Lim, W. K.; Bédard, S.; Black, B. E.; Englander, S. W. Protein Hydrogen Exchange: Testing Current Models: Testing HX Models. *Protein Sci.* **2012**, *21* (7), 987–995. https://doi.org/10.1002/pro.2082.

18. Wang, L.; Krishnamurthy, S.; Anand, G. Hydrogen Exchange Mass Spectrometry Experimental Design. In *Hydrogen Exchange Mass Spectrometry of Proteins: Fundamentals, Methods, and Applications*; John Wiley & Sons, Ltd, 2016; pp 19–36.

19. Aydin, S. A Short History, Principles, and Types of ELISA, and Our Laboratory Experience with Peptide/Protein Analyses Using ELISA. *Peptides* **2015**, *72*, 4–15. https://doi.org/10.1016/j.peptides.2015.04.012.

20. Crowther, J. R. *The ELISA Guidebook*; Walker, J. M., Series Ed.; Methods in Molecular Biology; Humana Press: Totowa, NJ, 2009; Vol. 516. https://doi.org/10.1007/978-1-60327-254-4.

Chapter 2

Confirmation of the Expression and Purification of the V_HH F5 Mutants

2.1 Introduction

The protein that was expressed for this project is called F5.[1] This protein is a construct of a series of tags and other small proteins, which can be referred to as a fusion protein. **Figure 2.1** illustrates the components of the fusion protein. The V_HH construct consists of a 6xHis tag, a thrombin cleavage site, a SUMO protein, a TEV cleavage site, and the F5 protein. The 6xHis tag aids in protein purification through nickel affinity chromatography. The thrombin cleavage site is a recognition and cleavage site that is used to cleave the 6xHis affinity tag from the SUMO protein if needed. The SUMO protein is a small protein that is commonly added to other small proteins that are difficult to express in bacteria to aid in protein expression. The TEV site is a cleavage site that is used to separate the F5 protein from the rest of the construct, which is necessary for HX-MS analysis of the F5 mutant. The F5 protein is where the mutations are located and contains two disulfide bridges in the higher order structure. The DNA for this fusion protein was placed in *E. coli* plasmids for expression. This chapter provides a brief overview of the protein expression and purification process, along with SDS-PAGE gel images and deconvoluted mass spectra for each mutant expressed to confirm the molecular weight matched the mutant identity.

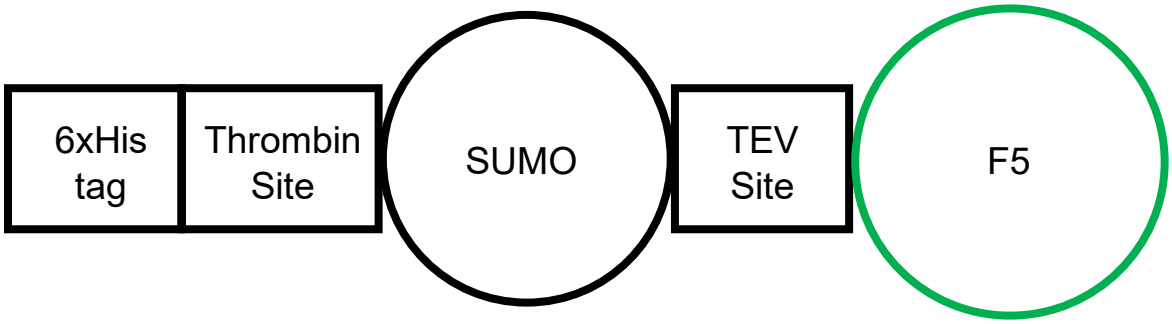

Figure 2.1: The V_HH F5 construct. The V_HH protein consists of a 6xHis tag, a thrombin cleavage site, a SUMO protein, a TEV cleavage site, and the F5 protein of interest (highlighted in green).

2.2 Protein expression and purification

A synopsis of the protein expression and purification procedure performed is listed below, while a more in-depth procedure can be found in Chamalee Gamage's dissertation.[2] Glycerol stocks of Origami2 *E. coli* cells containing the plasmid of interest were streaked onto a LB plate containing 100 µg/mL ampicillin and incubated overnight at 37 °C. Ampicillin is an antibiotic that prevents other bacteria from growing, whereas the *E. coli* strain that contains the F5 plasmid to express the F5 protein has a gene to make the bacteria ampicillin resistant. Bacteria that do not contain the ampicillin resistant F5 plasmid, or have ejected the plasmid, cannot survive in the ampicillin media. A single colony was used to inoculate LB broth containing 100 µg/mL of ampicillin in 50 mL of LB media for an overnight starter culture. The following morning, the culture was transferred to 1 L of LB media with 100 µg/mL ampicillin and continued to grow until an optical density at 600 nm reached 0.6-0.8. At that point, 1 mM of IPTG was added into the flasks to activate the *lac* operon to initiate protein expression of the F5 V_HH protein. Aliquots were taken at 0, 2, and 4 hours for an SDS-PAGE to confirm protein expression. After the 4 hours had passed, the culture was centrifuged to collect the cell pellet. The pellet was washed and resuspended in a 10 mM imidazole buffer before being centrifuged a second time to collect the final pellet of cells. The cell pellets were sonicated to lyse the cells to collect the protein of interest. The protein in the lysate was washed and purified via nickel affinity chromatography through increasing the concentration of imidazole buffer that was added into the column. SDS-PAGE confirmed that the purified protein sample contained only F5 fusion protein and the concentration of each elution fraction was measured using the Nanodrop at 280 nm, a wavelength

where tryptophan and tyrosine absorb light. The purified protein samples were then exchanged into a tris-HCl buffer using a 3 kDa molecular weight cut off filter (MWCO) filter to reduce the concentration of imidazole in the protein sample prior to TEV cleavage. Proteins were aliquoted, flash frozen, and stored at -80 °C until an intact mass analysis was conducted to confirm that the proper F5 mutant was expressed.

If a protein did not produce a high yield upon the first expression, a secondary expression was performed to increase the yield, but instead of placing 1 L of LB media in a 4 L flask for culture growth, the starter culture was split into two 500 mL portions of LB into two 4 L flasks. This increases the amount of oxygen at the interface of the LB broth, which is an abiotic factor for bacteria growth and protein expression. When yields were compared to protein expressed in one L of LB versus two 500 mL portions of LB, the yields were consistently higher when the LB was split for protein expression than using a single flask for culture growth.

2.3 Confirmation of F5 Construct Expression

As stated in the introduction, the V_HH construct consists of a His-Tag, a SUMO tag, a TEV cleavage site, and the F5 protein. The theoretical molecular weight for the wild type F5 V_HH with two intact disulfide bonds is 27953.50 Da, and since the mutants only vary in the amino acid sequence at one point, their molecular weights are approximated to be 28 kDa. V_HH constructs were confirmed via SDS-PAGE, a method that can be used to compare the molecular weight of the protein expressed to a molecular weight protein ladder. Each F5 V_HH protein expression culture was checked using SDS-PAGE to see if a band around 28 kDa increased in intensity over the course of the 4 hour expression process. As shown in **Figure 2.2-Figure** 2.4, each of the F5 proteins had an increase in

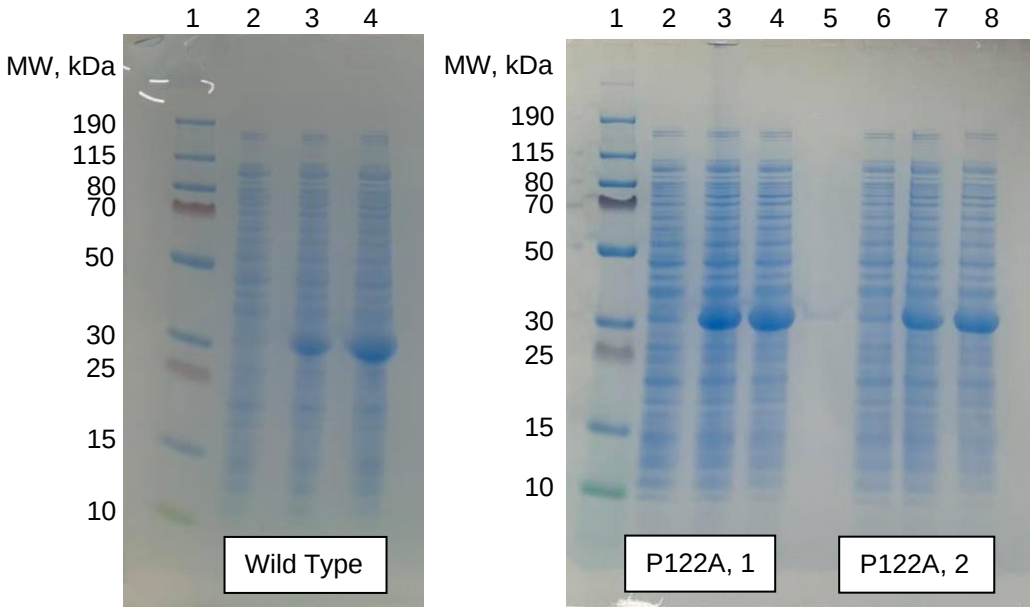

Figure 2.2: SDS-PAGE of wild type and P122A F5 expression. For the wild type, Lane 1 shows bands of a molecular weight ladder. Lanes 2-4 show bands for 0 hour, 2 hour, and 4 hour fractions taken during the expression process. For P122A, there are two sets of bands to monitor the expression of protein in each flask when increasing the oxygenation to increase the yield of protein expressed. Lane 1 shows bands of a molecular weight ladder. Lanes 2-4 (flask 1) and Lanes 6-8 (flask 2) show bands for 0 hour, 2 hour, and 4 hour fractions taken during the expression process.

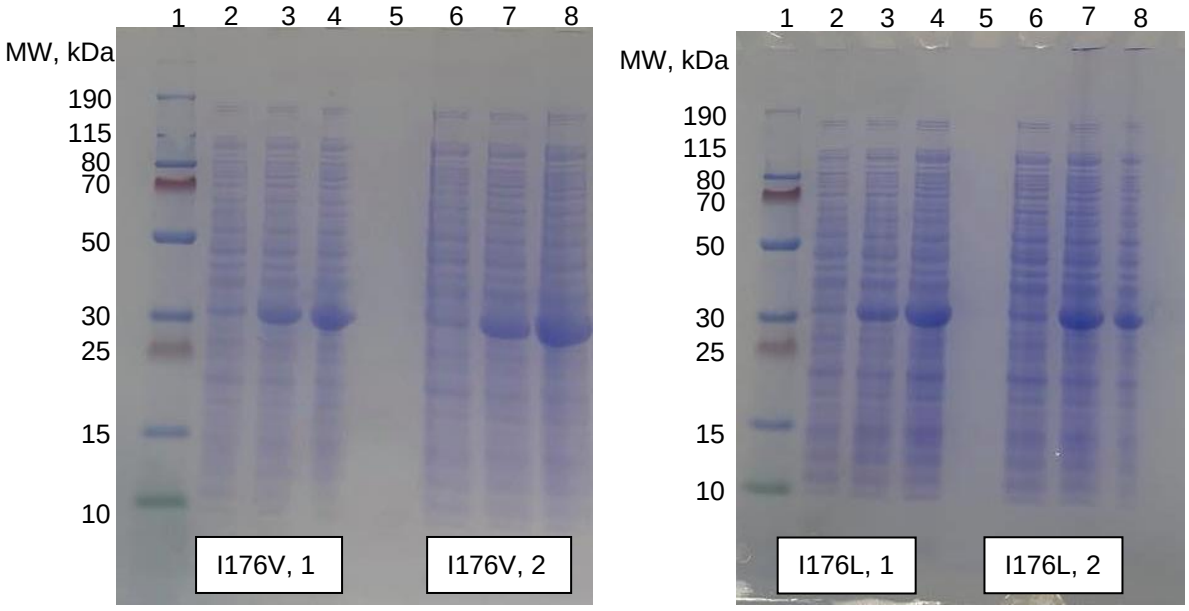

Figure 2.3: SDS-PAGE of I176V and I176L F5 expression. There are two sets of bands to monitor the expression of protein in each flask when increasing the oxygenation of the bacteria culture to increase the amount of protein that was expressed. Lane 1 shows bands of a molecular weight ladder. Lanes 2-4 and Lanes 6-8 show bands for 0 hour, 2 hour, and 4 hour fractions taken from flask 1 (Lanes 2-4) and flask 2 (Lanes 6-8) during the expression process.

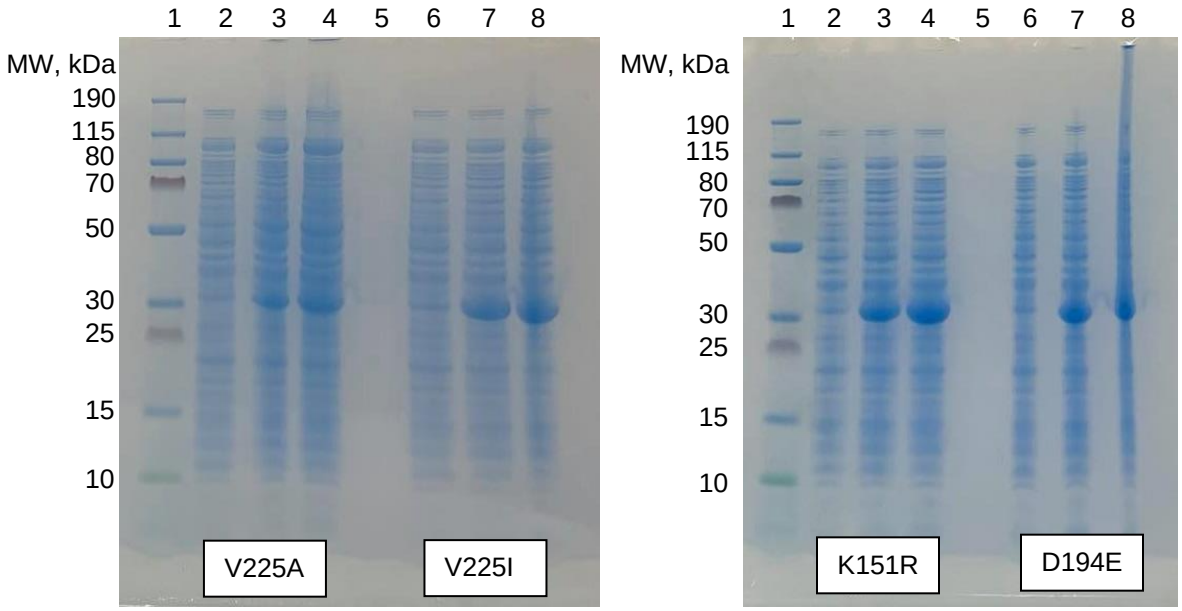

Figure 2.4: SDS-PAGE of V225A, V225I, K151R, and D194E V_HH expression. Lane 1 shows bands of a molecular weight ladder. Lanes 2-4 show bands for 0 hour, 2 hour, and 4 hour fractions taken during the expression process to confirm expression for mutants V225A/K151R. Lanes 6-8 show bands for 0 hour, 2 hour, and 4 hour fractions taken for V255I/D194E.

band intensity around 28 kDa over the 4 hour expression. Each mutant was believed to be expressed, but SDS-PAGE is not precise enough to confirm each mutants' exact molecular weight, which is why once the F5 V_HH proteins were purified, an intact mass check was performed.

2.4 Confirmation of Mutant Purification

The *E. coli* express more proteins than just the over-expressed protein of interest. This means that the F5 mutants have to be purified from the mixture of proteins. The V_HH construct contains a 6xHis tag which means that the F5 constructs can be purified using nickel affinity chromatography. The histidines in the 6xHis tag coordinate to the nickel in the column and a varying concentration of imidazole buffers are used to clean and elute the F5 mutants. Imidazole is used as the buffer because the structure is very similar to

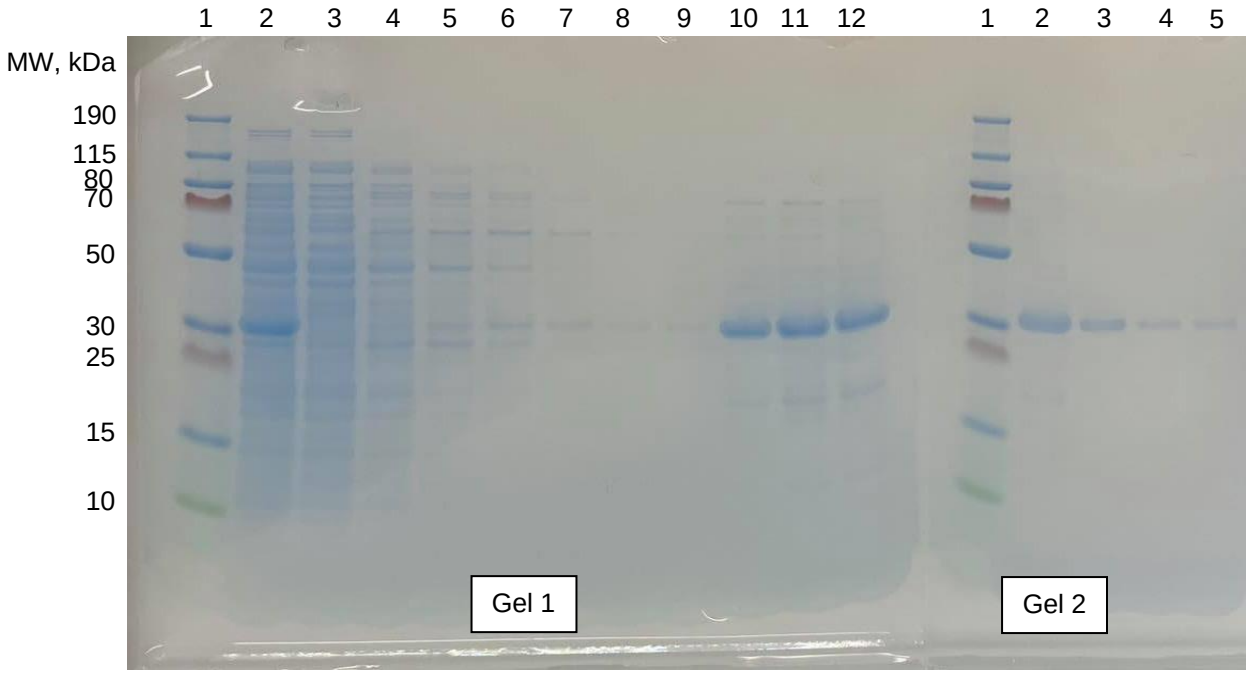

Figure 2.5: SDS-PAGE of wild type F5 purification via nickel affinity chromatography. For Gel 1, Lane 1 is the protein ladder. Lane 2 is the cell lysate. Lane 3 is the flow through from the nickel column. Lanes 4-9 are the column washes. Lanes 10-11 are elutions with 250 mM imidazole. Lane 12 is an elution with 500 mM imidazole. For Gel 2, Lane 1 is the protein ladder, lanes 2-5 contain elutions with 500 mM imidazole.

the histidine side chain. A high concentration of imidazole out-competes the 6xHis tag at the nickel coordination sites and releases the His tagged protein, while low concentrations of imidazole buffer displace any proteins that participate in low-binding or non-specific binding interactions with the nickel column. The wash steps use this low concentration of imidazole buffer and the elution steps increase the concentration of imidazole by an order of magnitude to elute the F5 V_HH mutants. **Figure 2.5** shows the SDS-PAGE of the wild type F5's purification process. The mutant purification SDS-PAGEs are in Appendix A of this chapter. The wash step samples show bands over a wide molecular weight range, indicating that the expressed *E. coli* proteins that are not the F5 protein of interest are washing out of the sample. The elution step samples show a heavy band just above the 30 kDa band in the protein ladder. The mass of the V_HH

construct is around 28 kDa, so it is believed that the band around 30 kDa is the F5 protein, which was verified post-purification using an intact mass analysis. There is also a decrease in the number of bands at other molecular weights, showing a cleaner protein sample. The purified proteins were buffer exchanged into a phosphate saline buffer, which is a more suitable buffer for mass spectrometry and TEV cleavage. The uncleaved F5 V_HH proteins were then subjected to intact mass analysis to confirm each mutant was expressed.

2.5 Intact Mass Analysis of Purified F5 Constructs

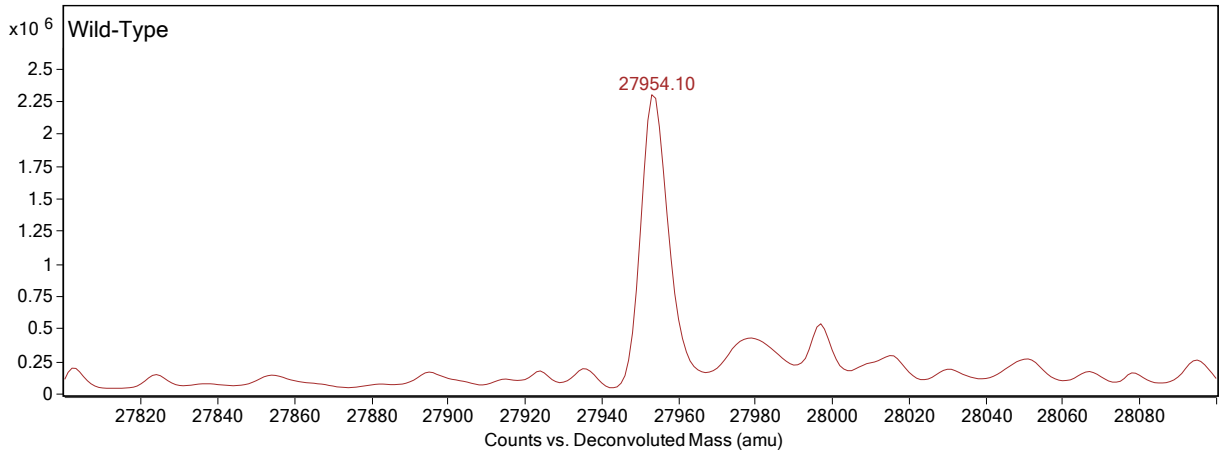

Figure 2.6: Deconvoluted mass spectra of wild-type F5 protein. The theoretical molecular weight of the wild-type protein with both disulfide bonds intact is 27953.5 Da.

The intact mass analysis for each F5 mutant was performed under native conditions to confirm the molecular weight of each protein. It was important to verify that the proteins with the point mutations were successfully expressed and that both disulfide bridges in the mutant were formed. Performing the intact mass under native conditions versus reducing conditions would mean that the disulfide bridges should remain intact if they were properly formed during expression. An example deconvoluted mass spectrum is shown in **Figure 2.6**. The deconvoluted mass spectra shown in Appendix B confirm that

each expected mutation was expressed. A peak is present in each mass spectra that is approximately +23 Da from the main peak in the spectra. This is believed to be a sodium adduct of the F5 mutant, which would shift the molecular weight of the mutant by 22 Da. The reason why the mass would shift by a value of 22 Da instead of 23 Da is because the sodium displaces a hydrogen ion to form the adduct, resulting in a loss of 1 Da to account for the removed hydrogen. Table *2.1* lists each of the F5 proteins expressed along with their theoretical molecular weight, their deconvoluted molecular weight, and the mass difference between the two values. All mutants had deconvoluted molecular weights within 1 Dalton of the theoretical molecular weight of the mutant.

Table 2.1: Intact Mass Table for the Uncleaved F5 V_HH proteins

F5 Mutant	Theoretical Molecular Weight with Disulfide Bonds	Molecular Weight via Intact Mass Analysis	Mass Difference
Wild Type	27953.5 Da	27954.1 Da	0.6 Da
P122A	27927.5 Da	27928.1 Da	0.6 Da
K151R	27981.5 Da	27982.0 Da	0.5 Da
I176L	27953.5 Da	27954.0 Da	0.5 Da
I176V	27939.5 Da	27940.0 Da	0.5 Da
D194E	27967.5 Da	27967.2 Da	-0.3 Da
V225A	27925.5 Da	27926.1 Da	0.6 Da
V225I	27967.5 Da	27968.0 Da	0.5 Da

2.6 References

1. Rudolph, M. J.; Vance, D. J.; Cassidy, M. S.; Rong, Y.; Shoemaker, C. B.; Mantis, N. J. Structural Analysis of Nested Neutralizing and Non-Neutralizing B Cell Epitopes on Ricin Toxin's Enzymatic Subunit: Nested B Cell Epitopes on RTA. *Proteins Struct. Funct. Bioinforma.* **2016**, *84* (8), 1162–1172. https://doi.org/10.1002/prot.25062.

2. Gamage, C. Studies of Protein Stability by Hydrogen Exchange-Mass Spectrometry. Ph.D. Dissertation, University of Kansas, Kansas, U.S.A., 2022.

Chapter 3

Assessing the relative binding affinity of V_HH mutants toward RTA using ELISA

3.1 Introduction

As stated in the first chapter, the goal of this project is to determine if a potential subtle change in a point mutated protein's structure and/or function is detectable by HX-MS. For the purposes of this project, the function of proteins has been evaluated by the binding affinity of the point mutated V_HH construct to RTA. Binding affinity is a value that measures how strongly two objects interact with each other. One of the reasons RTA and V_HH were selected as the model for this project was because they have a very strong binding affinity. The dissociation constant, K_D, for this interaction is 2.2 nM,[1] indicating a high binding affinity. If site directed mutagenesis is performed on the V_HH protein, the small alterations in the protein structure could impact how V_HH binds to RTA, if the mutation has an impact at all. The mutation could also severely alter the structure of V_HH to the point where it is "broken" and the binding interaction with RTA could not take place at all.

ELISA is a common technique that is used to evaluate relative binding affinity. Chapter 1 provided a general outline on the ELISA procedure. The specific ELISA method developed is an indirect ELISA. This ELISA method requires the antigen (RTA) to be directly bound to the plate and the primary antibody (V_HH F5) binds to the antigen. A conjugated secondary antibody (anti-His mAb) that is specific for the primary antibody is added to the plate. The conjugated antibody contains a component that aids in detection. In the case of this ELISA, the secondary antibody contains horseradish peroxidase (HRP) that partially oxidizes substrate TMB to produce a blue color.[2] The oxidation reaction can then be quenched with phosphoric acid to stop the reaction, producing a yellow color that can be used for colorimetric detection. Without this

quench, the HRP would continue to oxidize the TMB, continuing to produce a blue color until all of the TMB substrate has been converted into the TMB product. This chapter discusses the ELISA method developed, how the mutations could impact their relative binding affinity, and what the results concluded about each mutant's binding affinity relative to the wild-type V_HH F5.

3.2 ELISA Method

The ELISA procedure was developed by group member Chamalee Gamage.[3] The following is a synopsis of that method, with each F5 protein measured in triplicate. 100 µL of 2 µg/mL of RTA was added into Immunol 4HBX microplates (Thermo Fisher) and then incubated overnight at 4 °C. The next day, the plate was washed three times with wash buffer (0.1% TWEEN, 20 mM sodium phosphate, 150 mM sodium chloride, pH 7.40). The plate was patted dry, followed by the addition of 200 µL of 2% goat serum solution to block the plate. After 2 hours at room temperature, the blocking buffer was discarded from the plate and a 1:2 dilution series of the mutants was performed down the plate in blocking buffer. The first row of wells started with 200 µL of 1 µg/mL of the F5 protein. 100 µL of blocking buffer was added in rows 2-8. 100 µL of solution was taken from the first row, added into the subsequent row, and mixed 4 times prior to removing 100 µL and adding to the next row. After mixing the last row of the plate, 100 µL of the solution was removed to ensure that the volume in all wells was consistent. The plate was incubated for 1 hour at room temperature, followed by three wash steps with the wash buffer. The plate was then patted dry and 100 µL of 2 µg/mL of anti-His mAb was added into the wells and incubated for 50-55 minutes. The anti-His mAb was tagged with the enzyme horseradish peroxidase (HRP), which was subsequently used

for the colorimetric detection. The plate was washed three times and dried. 100 µL of 3,3',5,5'-tetramethylbenzidine (TMB) was added into each of the wells rapidly and reacted with HRP for 6 minutes. HRP reacts with the TMB substrate to produce a blue product. This blue product developed until it was quickly quenched with 100 µL of 1 M phosphoric acid at the 6-minute mark. The absorbance of the plate was measured at 450 nm using a plate reader. The collected data was exported as a .csv file to be worked up in the R-code described in the Chapter 4. The concentration at which 50% (C_{50}) of the mutant binds to RTA was compared to the wild-type F5 protein to draw conclusions about how the point mutation may have affected the binding affinity.

3.3 Binding Affinity Predictions

As stated in Chapter One, seven mutants were selected for protein expression and binding affinity testing. The mutants selected were P122A, K151R, I176L, I176V, D194E, V225A, and V225I. The locations of the original amino acids are indicated in **Figure 3.1**. Each mutant was selected for a potential effect that it would have on the F5

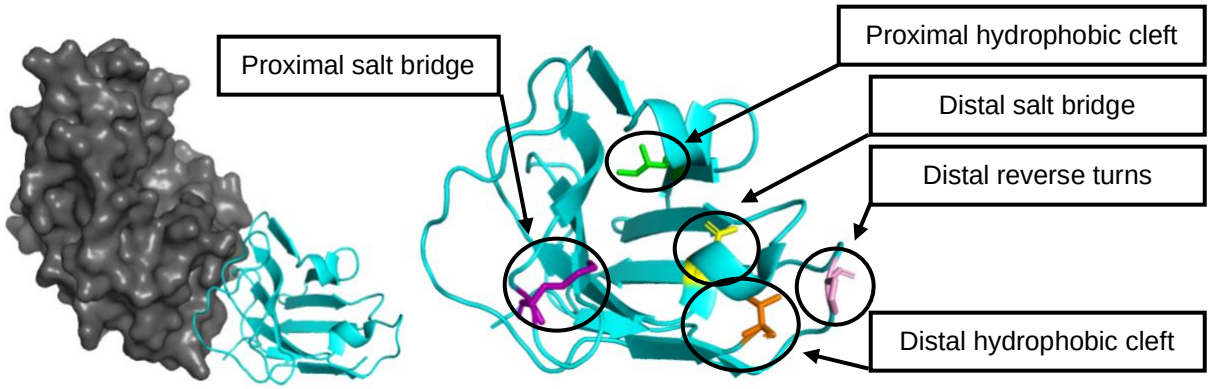

Figure 3.1: PyMOL images of the RTA-F5 complex from PDB 4Z9K[1] (left) and the locations of the amino acids that were mutated on the F5 structure (right). The amino acids highlighted are P122, K151, I176, D194, and V225 in pink, purple, green, yellow, and orange, respectively. The different regions the project focused on are circled and labeled in black.

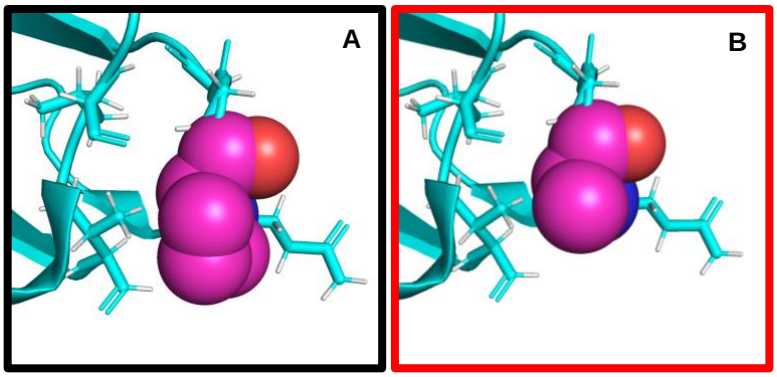

Figure 3.2: PyMOL images of the P122A mutation. 2A shows P122 as spheres and the surrounding amino acids as sticks. 2B shows P122A as spheres with the surrounding amino acids as sticks.

structure to see if a significant difference in binding affinity was measured using ELISA, indicating a change in the protein HOS, and if that difference in structure could then be detected in future work via HXMS. The first mutation selected was located in the distal turn regions of the F5 protein. The amino acid was P122, **shown in Figure 3.2**, which is restricted to certain dihedral angles in a protein structure due to the cyclical structure the side chain forms with the protein backbone. By converting the proline into an amino acid that does not have the same angle restrictions, such as alanine, it could alter the structure of the turn region and affect the affinity of the F5 to RTA.

Two of the mutations targeted polar side chains in distal and proximal salt bridges. K151, shown in **Figure 3.3A** and **Figure 3.3B**, forms a salt bridge in a region proximal to the paratope with E154 and has electrostatic interactions with the carbonyl carbon from local arginine residue R146 and local glutamine residue E152. If the lysine was mutated to an arginine, the change in side chain from a primary amine to a guanidino group could alter the salt bridge and electrostatic interactions, which could affect the binding of the F5 mutant to RTA. D194, shown in **Figure 3.3C** and **Figure 3.3D** forms a salt bridge distal from the paratope with an arginine residue (R173). If aspartic acid is

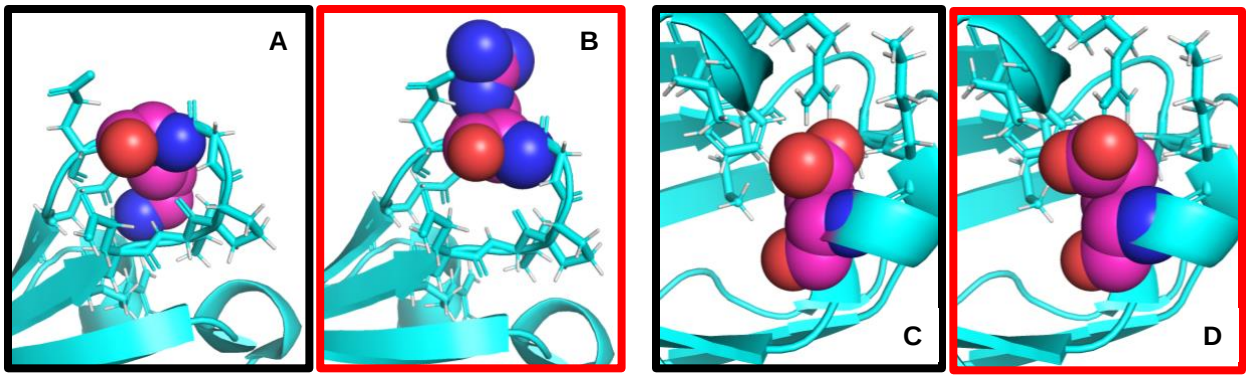

Figure 3.3: PyMOL images of the K151R mutation in 3A and 3B and of the D194E mutation in 3C and 3D. 3A and 3C show the original amino acid as spheres with surrounding residues as sticks, while the 3B and 3D show the mutation as spheres with the surrounding amino acids as sticks.

substituted for glutamic acid, the sidechain grows in length which could also affect the binding of the F5 mutant to RTA. The D194E mutation is hypothesized to cause the most disruptions in the F5 structure since the length of the sidechain is changing versus in K151R where the mutation does not alter the length of the sidechain but changes the type of headgroup on the end of the sidechain.

The final set of mutations focused on hydrophobic side chains. I176 and V225 were each compared to two separate mutants to create a total of four mutants for analysis. I176, shown in **Figure 3.4**, is located in a hydrophobic pocket proximal to the paratope, meaning it is surrounded by and is interacting with other hydrophobic side chains. If the size of the sidechain is reduced by mutating the amino acid to valine, shown in **Figure 3.4C**, that hydrophobic pocket might collapse toward the site of the mutation, disrupting the structure and changing the binding affinity. If the sidechain is mutated so that the structure of the sidechain is similar in size, but the orientation is changed through mutating the isoleucine to leucine (**Figure 3.4A**), then there is not much of a change that is occurring with the I176L mutation, but the side chain packing could be disrupted

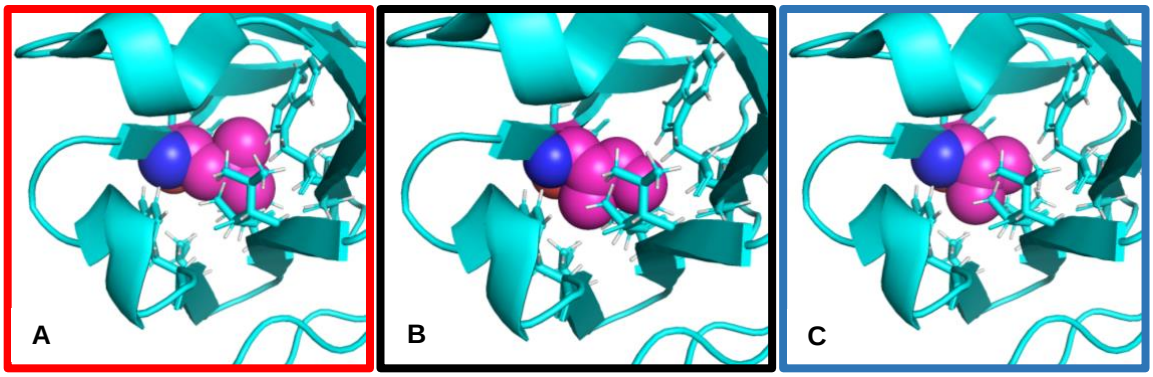

Figure 3.4: PyMOL images of the I176L/I176V mutation. The black box shows I176 as spheres and the surrounding amino acids as sticks. The red box uses the same depiction for I176L and the blue box for I176V.

and alter the binding affinity. The change in sidechain packing could strengthen the hydrophobic interactions in the pocket if the sidechain is oriented closer to the other hydrophobic residues. The alternative is that the disruption in sidechain packing could cause gaps within the hydrophobic pocket, which could cause a small collapse in the hydrophobic pocket and disrupt the F5 structure. The mutation that would have the greatest affect would be the I176V mutation since the potential collapse of the hydrophobic pocket could lead to more disruptions than the alteration in side chain packing.

For V225, shown in **Figure 3.5**, the residue is in a hydrophobic pocket similar to I176, but it is distal from the paratope and also rests in the last few residues of the F5 protein. This positioning is important since the end of the protein could potentially have more movement than a centrally located amino acid. Valine can be mutated so the size of the side chain is decreased, by mutating to alanine as shown in **Figure 3.5A** or increasing the side chain size by mutating the valine to isoleucine, as shown in **Figure 3.5C**. Like the decrease in the side chain size for I176, the V225A mutation could result in a collapse of the hydrophobic pocket, which would alter the tertiary structure and

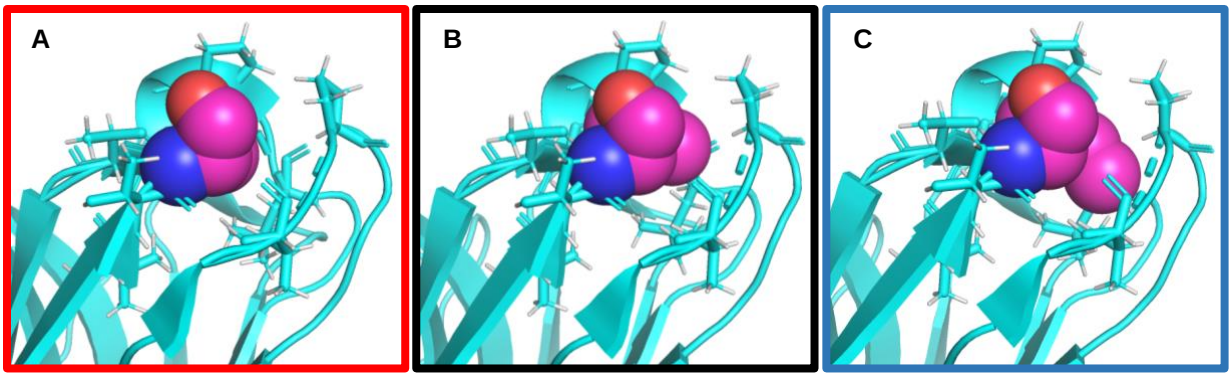

Figure 3.5: PyMOL images of the V225A/V225I mutation. 5B shows V225 as spheres and the surrounding amino acids as sticks. The same depiction is used in 5A for V225A and the 5C for V225I.

potentially change the affinity of the F5 to RTA. Increasing the size of the side chain, like the V225I mutation, can result in steric crowding with the neighboring hydrophobic residues or it could strengthen the hydrophobic interactions. Overall, the V225A mutation is hypothesized to cause the most disruption in the structure of F5, which could in turn cause the most disruptions in the binding affinity of the F5 to RTA.

Based on the hypotheses for each mutation, the mutations that would be expected to have the greatest impact on the HOS, reflected in the binding affinity, would be the D194E, the V225A, and the I176V mutations. The mutation that would be expected to have the least impact on the binding affinity of F5 to RTA would be the I176L mutation.

3.4 Data Analysis

The value that was used to compare the binding affinity of the mutants to the F5 protein was a value called C_{50}. This value is the concentration of F5 at which 50% of the F5 protein is bound to RTA on the ELISA plate. A mutant that has a lower relative affinity for RTA than the wild type would have a higher value for C_{50}. The amount of F5 that stays in complex with the RTA is dependent upon the affinity, and a lower relative affinity means that the F5 dissociates from the RTA at a higher F5 concentration than a

mutant with a stronger relative affinity that can stay in complex with the RTA at lower F5 concentrations. **Figure 3.6** provides a visual indication of determining C_{50} on a normalized ELISA plot. Each mutant's normalized absorbance values from ELISA were plotted against the concentration of the F5 mutant in the well. The horizontal dashed black line across the normalized ELISA plot is placed at 50% normalized absorbance. The intersection between the dashed line with each mutant's normalized ELISA plot is the concentration of the F5 mutant that binds to RTA at a normalized 50% absorbance. Once the C_{50} values are calculated from the ELISA plot, which is covered in more detail in Chapter 4, the C_{50} values for each mutant were normalized to the C_{50} of the wild type protein (100%) and displayed as a percentage to account for experimental variability between other assays conducted at the same time or between lab members' ELISA experiments.

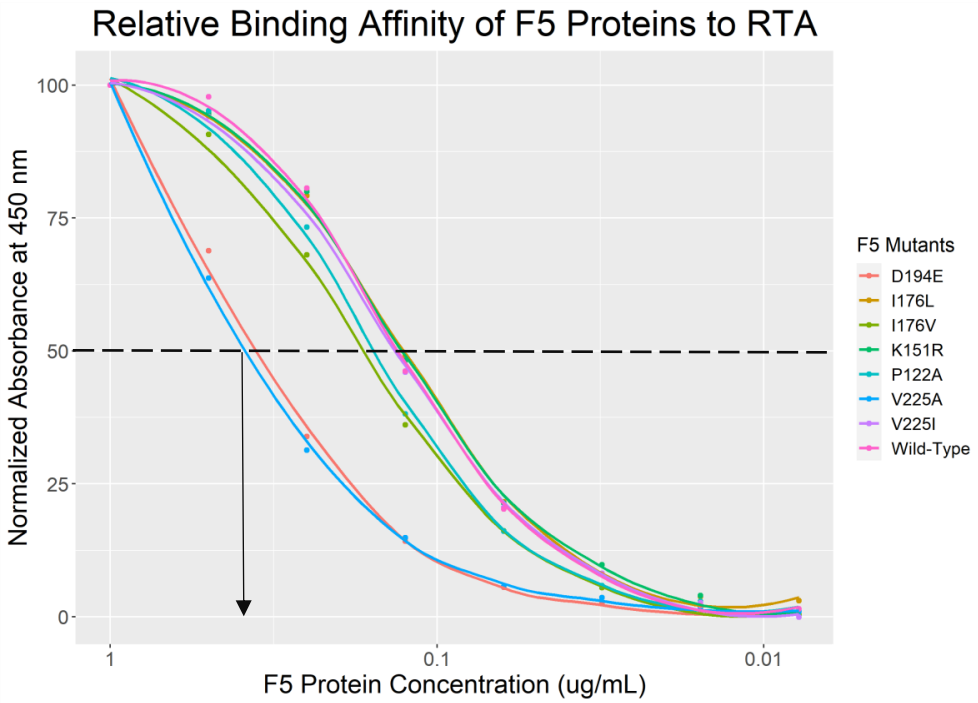

Figure 3.6: ELISA plot of all F5 mutants and the wild type. Black dashed line is the C_{50} line, where 50% absorbance indicates how much F5 is bound to the plate. Created by R code discussed in Chapter 4.

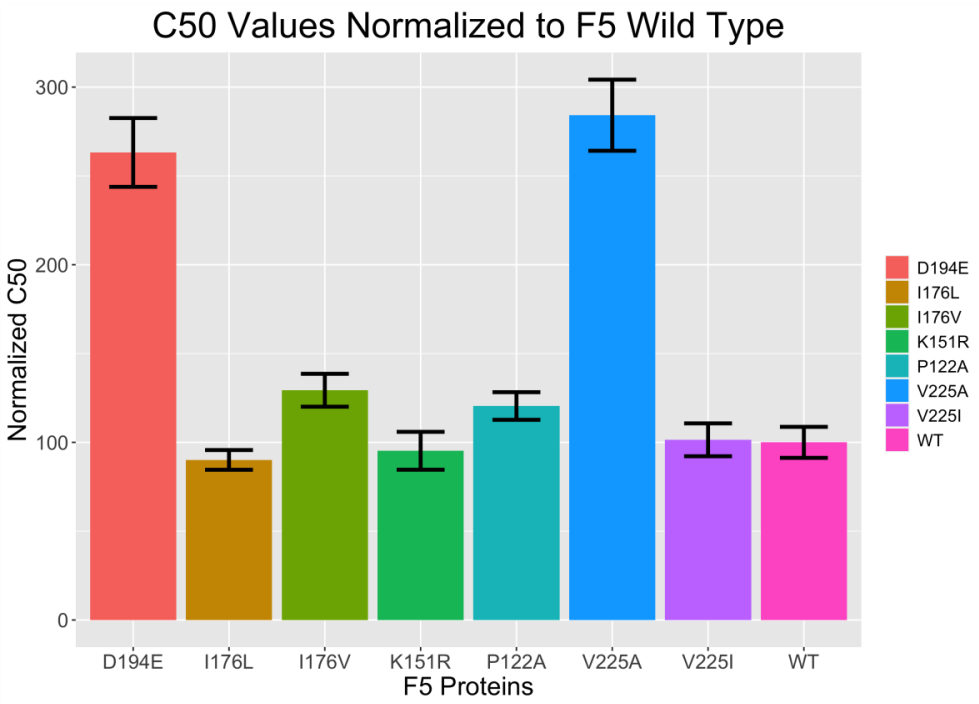

Figure 3.7: Normalized C_{50} bar plot of the F5 mutants compared to the wild-type F5. Error bars indicate a 95% confidence interval. Created by R code discussed in Chapter 4.

When referring to the error bars from the plot in **Figure 3.7**, which were calculated using a 95% confidence interval covered in Chapter 4, there are four mutants that have significantly different C_{50} values from the wild type protein. These are D194E, I176V, P122A, and V225A. The confidence intervals for these four mutants do not overlap with the wild type protein, with two mutants having a substantial difference (D194E and V225A) and the other two mutants having a smaller difference than the wild type (I176V and P122A). These mutants all had higher C_{50} confidence intervals than the wild type, indicating that they had a lower affinity for RTA than the wild type.

Three of these mutants were hypothesized to have stronger effects on relative binding affinity – D194E, I176V, and V225A – but P122A was not a part of that hypothesis. This would suggest that mutating a proline in the turn region has a significant effect on the

structure of the F5 protein. When looking at the overlapping error bars in **Figure 3.7**, the other three mutants, I176L, K151R, and V225I, did not show a significant difference in affinity when compared to the wild type F5 protein based on the statistical method used. A more complex statistical analysis would need to be conducted to verify that the differences in binding affinity between the three mutants and the wild type are insignificant.

The mutants that should move forward for HXMS analysis are D194E, V225A, and P122A. D194E had a large decrease in the binding affinity to RTA, due to the disruption of the salt bridge, that it would be interesting to confirm this effect is visible by HX-MS. V225A had a decrease in binding affinity due to the collapse of the hydrophobic pocket, just like I176V, but the magnitude of difference could be because V225A is located towards the C-terminus end of the protein that could be more dynamic. This would be another great mutant to confirm if the mutated amino acid has structural differences detectable by HX-MS. P122A had a decrease in binding affinity since the proline side chain could not restrict the dihedral angles in the turns of the protein. The difference in binding affinity between P122A and the wild type is small, so corroborating the difference with HXMS, and confirming if HX-MS can detect such a small difference, would be useful.

3.5 References

1. Rudolph, M. J.; Vance, D. J.; Cassidy, M. S.; Rong, Y.; Shoemaker, C. B.; Mantis, N. J. Structural Analysis of Nested Neutralizing and Non-Neutralizing B Cell Epitopes on Ricin Toxin’s Enzymatic Subunit: Nested B Cell Epitopes on

RTA. *Proteins Struct. Funct. Bioinforma.* **2016**, *84* (8), 1162–1172. https://doi.org/10.1002/prot.25062.

2. Kireyko, A. V.; Veselova, I. A.; Shekhovtsova, T. N. Mechanisms of Peroxidase Oxidation of O-Dianisidine, 3,3′,5,5′-Tetramethylbenzidine, and o-Phenylenediamine in the Presence of Sodium Dodecyl Sulfate. *Russ. J. Bioorganic Chem.* **2006**, *32* (1), 71–77. https://doi.org/10.1134/S1068162006010079.

3. Gamage, C. Studies of Protein Stability by Hydrogen Exchange-Mass Spectrometry. Ph.D. Dissertation, University of Kansas, Kansas, U.S.A., 2022.

Chapter 4

Developing an R Script to Calculate C_{50} and the Uncertainty in C_{50}

4.1 Introduction

Chapter 3 walked through the ELISA method that was developed for a relative binding affinity analysis of wild-type F5 versus the mutant F5 proteins. The data that was obtained for the ELISA assay requires inverse linear regression to calculate the concentration of F5 protein that binds to the RTA on the plate. This concentration value is interpolated by using data points above and below the 50% normalized ELISA response for each F5 protein. Due to the different variables associated with this calculation, a special way to calculate the uncertainty associated with this measurement is required. The uncertainty analysis for an inverse linear regression calculation was taken from Draper and Smith.[1] This uncertainty calculation accounts for the multiple variables involved in calculating the concentration of F5 protein at 50% normalized ELISA response, but the calculation is complex. An idea to develop an R code to simplify the calculations for the user was developed. A code for these calculations would ensure consistency between different plates and different users and aid in working up the raw data for the inverse linear regression and uncertainty calculations for the ELISA response. The code also develops figures that can be used for journal articles or PowerPoint presentations. Chapter 4 outlines the steps the code takes in working up the raw data, how the inverse linear regression calculation and uncertainty for the 50% normalized ELISA response is calculated, and exceptions to the code that were developed based on the raw data collected from three different lab personnel.

4.2 Code Packages and Initial Lines

The R code required a series of packages that needed to be installed for different purposes. The four packages installed were PEIP, reshape2, ggplot2, and scales. PEIP

CONCENTRATION	WILD-TYPE	MUTANT 1	MUTANT 2
1	1.0	1.0	1.0
0.5	0.8	0.6	0.8
0.1	0.6	0.5	0.4

PROTEIN	CONCENTRATION	ABSORBANCE
WILD-TYPE	1	1.0
WILD-TYPE	0.5	0.8
WILD-TYPE	0.1	0.6
MUTANT 1	1	1.0
MUTANT 1	0.5	0.6
MUTANT 1	0.1	0.5
MUTANT 2	1	1.0
MUTANT 2	0.5	0.8
MUTANT 2	0.1	0.4

Figure 4.1: An example of how the melt function reorganizes a data table in R. The first table in the figure is the original data table and the second table is what the melt function generates if the column is selected as the identifier for the data.

contains a statistical function called tinv that determines the t test value when a p value is provided for the statistical analysis. The reshape2 package contains a function called melt that organizes the data by the row or column header, making the data “tidy”. A simple example of how the melt function, selecting for the column header, works is shown in **Figure 4.1**. Converting the data tables was essential for graphing the data. ggplot2 is a package that is used to generate graphics in R. The package was used to create both figures that the R code produces. The final package, scales, was a package that was used to help generate the axis of the ELISA plot. The axis needed to be logarithmic and in reverse order, so the scales package has a series of functions to assist with that.

The first part of the code converts the raw data into a usable format for the calculations to follow. The raw data contains the absorbance values for a 12 column by 8 row well plate. The proteins are varied in the columns, with 3 columns of one protein, a blank column, and the pattern repeated two more times with different mutants. The first lines of code consist of user inputs such as the user’s initials, the raw data file name and

ELISA_VHH_R_Code.R × vhh ×

Filter

	Wild-Type	Wild-Type	Wild-Type	Blank_Wild-Type	V225A	V225A	V225A	Blank_V225A	D194E	D194E	D194E	Blank_D194E
4	3.4895	3.5005	3.5288	0.1518	3.0768	2.9501	3.0034	0.1186	3.1562	3.1110	3.1017	0.1130
5	3.4924	3.4116	3.3972	0.1521	2.0599	1.8765	1.9799	0.1165	2.1815	2.1581	2.2441	0.1478
6	2.9041	2.8514	2.8201	0.1503	1.0428	1.0804	1.0175	0.2073	1.2419	1.1019	1.1216	0.1472
7	1.7672	1.7005	1.6481	0.1787	0.579	0.5563	0.5930	0.1622	0.5870	0.5641	0.5500	0.1654
8	0.8455	0.8408	0.8219	0.1827	0.2904	0.3047	0.3485	0.1477	0.3187	0.3058	0.3015	0.1475
9	0.4054	0.436	0.4337	0.1339	0.258	0.2276	0.2745	0.1452	0.2253	0.2297	0.2002	0.1337
10	0.2039	0.2248	0.2577	0.1474	0.2249	0.1911	0.1748	0.1383	0.1767	0.1827	0.1600	0.1254
11	0.1807	0.2489	0.1911	0.1585	0.2065	0.1823	0.1441	0.1682	0.1437	0.1607	0.1605	0.1745

Figure 4.2: vhh data frame that identifies each mutant on the plate and the blank column associated with each F5 mutant.

ELISA_VHH_R_Code.R × vhhcorr ×

Filter

	Wild-Type	Wild-Type.1	Wild-Type.2	V225A	V225A.1	V225A.2	D194E	D194E.1	D194E.2
4	3.332575	3.343575	3.371875	2.9263	2.7996	2.8529	3.0118875	2.9666875	2.9573875
5	3.335475	3.254675	3.240275	1.9094	1.7260	1.8294	2.0371875	2.0137875	2.0997875
6	2.747175	2.694475	2.663175	0.8923	0.9299	0.8670	1.0975875	0.9575875	0.9772875
7	1.610275	1.543575	1.491175	0.4285	0.4058	0.4425	0.4426875	0.4197875	0.4056875
8	0.688575	0.683875	0.664975	0.1399	0.1542	0.1980	0.1743875	0.1614875	0.1571875
9	0.248475	0.279075	0.276775	0.1075	0.0771	0.1240	0.0809875	0.0853875	0.0558875
10	0.046975	0.067875	0.100775	0.0744	0.0406	0.0243	0.0323875	0.0383875	0.0156875
11	0.023775	0.091975	0.034175	0.0560	0.0318	-0.0064	-0.0006125	0.0163875	0.0161875

Figure 4.3: vhhcorr data frame that has the average blank absorbance value subtracted from the absorbance of each mutant.

extension, and the order of the mutants on the plate. The code then identifies which column of the plate is a blank for each mutant, labels the column as a blank for that specific mutant, and the average for each blank column is calculated. **Figure 4.2** shows what this data frame looks like with the blank labeled columns. The blank columns are removed from the data frame and the mutants are blank corrected with the average blank absorbance stored previously, shown in **Figure 4.3**.

4.3 Working up Absorbance Data

ELISA_VHH_R_Code.R × normvhh ×

Filter

	Wild-Type	Wild-Type.1	Wild-Type.2	V225A	V225A.1	V225A.2	D194E	D194E.1	D194E.2
4	99.913056	100.000000	100.000000	100.000000	100.000000	100.0000000	100.00000000	100.0000000	100.0000000
5	100.000000	97.341169	96.097127	65.249633	61.651665	64.1242245	67.63823350	67.8800008	71.0014329
6	82.362332	80.586648	78.982020	30.492431	33.215459	30.3901293	36.44184917	32.2780037	33.0456357
7	48.277232	46.165407	44.223911	14.643065	14.494928	15.5105331	14.69800914	14.1500411	13.7177661
8	20.643986	20.453407	19.721223	4.780781	5.507930	6.9403064	5.78997390	5.4433607	5.3150796
9	7.449464	8.346605	8.208341	3.673581	2.753965	4.3464545	2.68892845	2.8782101	1.8897591
10	1.408345	2.030013	2.988693	2.542460	1.450207	0.8517649	1.07532237	1.2939516	0.5304513
11	0.712792	2.750798	1.013531	1.913679	1.135877	-0.2243331	-0.02033608	0.5523838	0.5473581

Figure 4.4: normvhh data frame that shows the normalized absorbance values for each column on the plate.

Once blank corrected, the absorbance values are normalized. The way this works is the script finds the maximum absorbance value in each column and divides the absorbance values in that column by the maximum value, then multiplies the new value by 100%. This was done for every column in the blank corrected data frame and stored as normvhh, the data shown in **Figure 4.4**.

Then, the average of each normalized absorbance for each set of concentration replicates for each protein is calculated. An empty data frame is created to store these average values called avvhh. The if loops assign the protein name to each row in the new data frame. The for loops took the average of the absorbances for each protein concentration and added that value into the avvhh data frame. The data frame is transposed so that the protein names are now column headers versus row headers and the row names are assigned as the concentration of the mutant. This data frame, shown in **Figure 4.5**, is used to aid in graphing the ELISA plot later in the R script.

ELISA_VHH_R_Code.R × avvhh ×

Filter

	Wild-Type	V225A	D194E
1	99.971019	100.0000000	100.0000000
0.5	97.812765	63.6751739	68.8398891
0.25	80.643667	31.3660065	33.9218295
0.125	46.222183	14.8828419	14.1886054
0.0625	20.272872	5.7430058	5.5161381
0.03125	8.001470	3.5913334	2.4856326
0.015625	2.142350	1.6148106	0.9665751
0.0078125	1.492374	0.9417409	0.3598019

Figure 4.5: avvhh data frame that will be used for plotting the ELISA plots.

ELISA_VHH_R_Code.R × dmM.g501 × dmM.l501 × normvhh1 ×

Filter

	1	0.5	0.25	0.125	0.0625	0.03125	0.015625	0.0078125
Wild-Type	99.91306	100.00000	82.36233	48.27723	20.64399	7.449464	1.408345	0.712792
Wild-Type.1	100.00000	97.34117	80.58665	46.16541	20.45341	8.346605	2.030013	2.750798
Wild-Type.2	100.00000	96.09713	78.98202	44.22391	19.72122	8.208341	2.988693	1.013531

Figure 4.6: normvhh1 matrix. This was developed for the wild-type protein. One matrix is developed for each F5 mutant.

When calculating the avvhh data frame, three new matrices (normvhh1, 2, and 3) are created to separate the normalized absorbance values for each mutant. These matrices, an example of which is shown in **Figure 4.6**, are used to determine the data points that are above and below a 50% normalized absorbance value for each mutant. To determine the points around 50% normalized absorbance, the code first identifies if there are any lines that contain a normalized absorbance that is 50-59%. This is because if the error associated with the three data points for that mutant concentration overlaps at 50%, then that concentration value will be taken as the C_{50} value. If this is identified in a data frame and the error bars overlap over 50%, then that concentration value is taken for the C_{50} value for that mutant. If the error bars do not overlap, then the code moves on to a different set of code. If the data frame does not have any normalized absorbance values around 50-59%, then the code will move on as well.

ELISA_VHH_R_Code.R × dmM.g501 ×

Filter

	Var1	Var2	value
1	Wild-Type	1.00	99.91306
2	Wild-Type.1	1.00	100.00000
3	Wild-Type.2	1.00	100.00000
4	Wild-Type	0.50	100.00000
5	Wild-Type.1	0.50	97.34117
6	Wild-Type.2	0.50	96.09713
7	Wild-Type	0.25	82.36233
8	Wild-Type.1	0.25	80.58665
9	Wild-Type.2	0.25	78.98202

ELISA_VHH_R_Code.R × dmM.l501 ×

Filter

	Var1	Var2	value
10	Wild-Type	0.1250000	48.277232
11	Wild-Type.1	0.1250000	46.165407
12	Wild-Type.2	0.1250000	44.223911
13	Wild-Type	0.0625000	20.643986
14	Wild-Type.1	0.0625000	20.453407
15	Wild-Type.2	0.0625000	19.721223
16	Wild-Type	0.0312500	7.449464
17	Wild-Type.1	0.0312500	8.346605
18	Wild-Type.2	0.0312500	8.208341
19	Wild-Type	0.0156250	1.408345
20	Wild-Type.1	0.0156250	2.030013
21	Wild-Type.2	0.0156250	2.988693
22	Wild-Type	0.0078125	0.712792
23	Wild-Type.1	0.0078125	2.750798
24	Wild-Type.2	0.0078125	1.013531

Figure 4.7: The two melted data frames. The left data frame shows the lines of data that had normalized absorbances over 50% and the right data frame shows the data that had normalized absorbances below 50%.

If the data does not fit the criteria listed above, the code will separate the data into normalized absorbances above and below 50%. The melt function is used to help make this identification easier, by sorting the data frame from high to low concentration for all three columns for a mutant. **Figure 4.7** shows that the melted data frame is split between greater than or less than 50% normalized absorbance. The last three data points are taken from the greater than 50% data frame and the first three data points are taken from the less than 50% data frame and are stored in two vectors shown in **Figure 4.8**: a concentration vector and an absorbance vector.

This process is done for each mutant on a plate until there is an avvhh data frame, a concentration vector, and an absorbance vector for each mutant. Then, inverse linear regression and uncertainty calculations are conducted for each mutant.

```
> p1con
[1] 0.250 0.250 0.250 0.125 0.125 0.125
> p1abs
[1] 82.36233 80.58665 78.98202 48.27723 46.16541 44.22391
```

Figure 4.8: The concentration and absorbance vectors developed for the wild-type F5 protein. The p1abs vector shows the stored absorbances just above and below 50% normalized absorbance, while the p1con vector stores the concentration of F5 protein associated with those respective absorbance values.

4.4 Inverse linear regression and uncertainty in 50% normalized ELISA response

An inverse linear regression analysis is performed with the points above and below 50% normalized absorbance each mutant. These values are used to calculate the 50% normalized ELISA response for each mutant, and the concentration associated with that response is the value referred to as C_{50}. The uncertainty analysis for this interpolation generates the uncertainty involved in the ELISA response at a normalized 50% absorbance. The approach taken for the calculations was outlined by Draper and Smith.[1] The lm function in R obtains values that are calculated for a traditional linear regression analysis. The slope (m) and its standard deviation (σ) are pulled from the values the lm function calculated. The averages are taken for all concentration and normalized absorbance data points. The qt function calculates the t value for a single tail t test (t). Since the ideal t test value was a two tailed test at a 95% confidence interval, the code calculated a single tail t test at a 97.5% confidence interval, which would provide the same t value as the two tailed test at 95% confidence. Another reason why this is an acceptable way to approach the confidence intervals is the code is generating a concentration at 50% normalized absorbance, where 50% is a theoretical value that has no uncertainty associated with that value. Based on the writings of Draper and Smith,[1] the equation to calculate the uncertainty for this type of analysis is

shown in Equation 1, and utilizes the values discussed along with the average y value, number of variables (n), and sum square of x values (SS_x).

Equation 1: $$u = \frac{m(50-\bar{y}) \pm t\sigma\sqrt{\frac{(50-\bar{y})^2}{SS_x} + \left(\frac{m^2}{n}\right) - \left(\frac{t^2\sigma^2}{(n)(SS_x)}\right)}}{m^2 - \left(\frac{t^2\sigma^2}{SS_x}\right)}$$

To calculate the upper and lower values for the C_{50} range, the higher or lower value of uncertainty was added to the average concentration for the mutant, shown in Equations 2 and 3. The value for C_{50} was determined by using Equation 4. The error associated with determining the value of C_{50} was calculated using Equation 5.

Equation 2: $$C_{50}\ \text{upper limit} = \bar{x} + u_{high}$$

Equation 3: $$C_{50}\ \text{lower limit} = \bar{x} + u_{low}$$

Equation 4: $$C_{50} = \frac{(\text{upper limit} - \text{lower limit})}{2} + \text{lower limit}$$

Equation 5: $$C_{50}\ \text{error} = C_{50} - \text{lower limit}$$

The values generated from Equations 4 and 5 are stored in separate matrices and each mutant's value is added into the matrices. Once each mutant has their data stored in the matrices, the matrices are combined into one data frame. After all plates and mutants are added into a single data frame, the data frame is converted into a .csv file and is exported into the working directory. This file was labeled as "… C50 File.csv" and organizes the data as shown in **Figure 4.9**. The avvhh data, containing the ELISA plot points, also combines multiple plates into one data frame and exports it as a .csv file

2022-08-04 HB C50 File

	WT	V225A	D194E	K151R	V225I	P122A	I176L	I176V
C50	0.13871672719568	0.394188447275491	0.365111441889492	0.132132610954037	0.140703147788406	0.167109594636542	0.125	0.179410496526985
error	0.00855893892651424	0.0134216809312996	0.0146303775688932	0.0123497189281332	0.00946088113916654	0.00318669670395386	0	0.00651360109249652

Figure 4.9: C50 data file that contains the C_{50} values and error associated with each value.

2022-08-04 HB ELISA File

	WT	V225A	D194E	K151R	V225I	P122A	I176L	I176V
1	99.9710186205363	100	100	100	100	100	100	100
0.5	97.8127654089575	63.675173879583	68.8398890649744	94.6421091929117	94.0604459172204	95.1460463741614	94.5535643977394	90.740113093913
0.25	80.6436667054293	31.366006469321	33.9218295185643	79.9947707371327	77.8911571370239	73.303509833389	79.2321384943463	68.0736258665919
0.125	46.2221832316311	14.882841869801	14.1886054408811	48.1836077148061	45.991966965621	38.1613808134083	49.0703896190033	36.0686670447914
0.0625	20.2728721670257	5.74300575014866	5.51613805187478	21.3354059417254	20.5730076297918	16.077106985504	21.6312324952136	16.1203216825675
0.03125	8.00146994401985	3.59133343653034	2.48563256647711	9.77211338990501	8.18558288143805	5.88379511511819	8.04441392230897	5.44389848176028
0.015625	2.14235038614698	1.61481060158664	0.966575080332057	4.02080234338313	2.64204976629433	2.9132501895487	3.81201935253846	1.90342583124914
0.0078125	1.49237375033354	0.941740936481595	0.359801925139236	-0.0223466693794009	-0.0591045911582032	1.24133277594156	3.050484285973	0.39674324026877

Figure 4.10: ELISA .csv file that contains the average normalized absorbance values for each F5 mutant.

into the working directory, shown in **Figure 4.10.**

4.5 C_{50} Bar plot

To create the C_{50} bar plot, the code sources the C_{50} .csv file from the working directory. The wild type C_{50} concentration and C_{50} error are saved as values that are used to normalize the mutant C_{50} concentrations and C_{50} errors to the wild-type F5 protein. The mutant C_{50} concentrations are divided by the wild type and multiplied by 100. The mutant C_{50} error propagation calculations used the equation provided in Equation 6 to normalize the error to match that the C_{50} concentration was normalized. Since the C_{50} values were manipulated through normalizing each mutants' C_{50} value to the wild type, the error that was generated for each F5 protein in Equation 5 would now have to reflect the change in the error associated with the measurement of the C_{50} values because the C_{50} values were normalized.

Equation 6: $$\text{Mutant } C_{50} \text{ error} = \sqrt{\left(\frac{\text{mutant } C_{50}\ u}{\text{mutant } C_{50}}\right)^2 + \left(\frac{\text{WT } C_{50}\ u}{\text{WT } C_{50}}\right)^2}$$

2022-08-04 HB Normalized C50 File

	C50	error
D194E	263.206499512095	19.3643408572568
I176L	90.1116992355719	5.5599677552492
I176V	129.335877621954	9.25911986451371
K151R	95.2535527800082	10.6678117402604
P122A	120.468236250131	7.77989787907766
V225A	284.167926424209	20.0259192126091
V225I	101.431997880057	9.25658708006536
WT	100	8.72581681683189

Figure 4.11: The Normalized C50.csv file that contains the normalized C_{50} values and uncertainty associated with the normalized C_{50} value.

The data frame was updated with the normalized C_{50} values, and the new error associated with them. Then, the data frame was organized alphabetically by mutant name to make the bar plot and ELISA plot consistent with the order of mutants and the coloring associated with the graphs. The data frame was exported as a .csv file to the working directory in case a member of the lab wanted to refer back to the values calculated by the R code. An example of this .csv file is shown in **Figure 4.11**.

ggplot was used to generate the bar plot and required manipulation of the theme and the addition of error bars in the plot. The code generated .png images of the plots in two sizes: one for a PowerPoint and one for a journal article. This would allow for consistency between graphs across lab members and across various science communication platforms. An example of a figure that is generated from the R code is shown in **Figure 4.12**.

4.6 ELISA plot

The ELISA .csv file that was generated earlier in the code was sourced from the working directory. The data frame was arranged alphabetically by mutant name prior to using the melt function. The melt function converted the data frame to create three

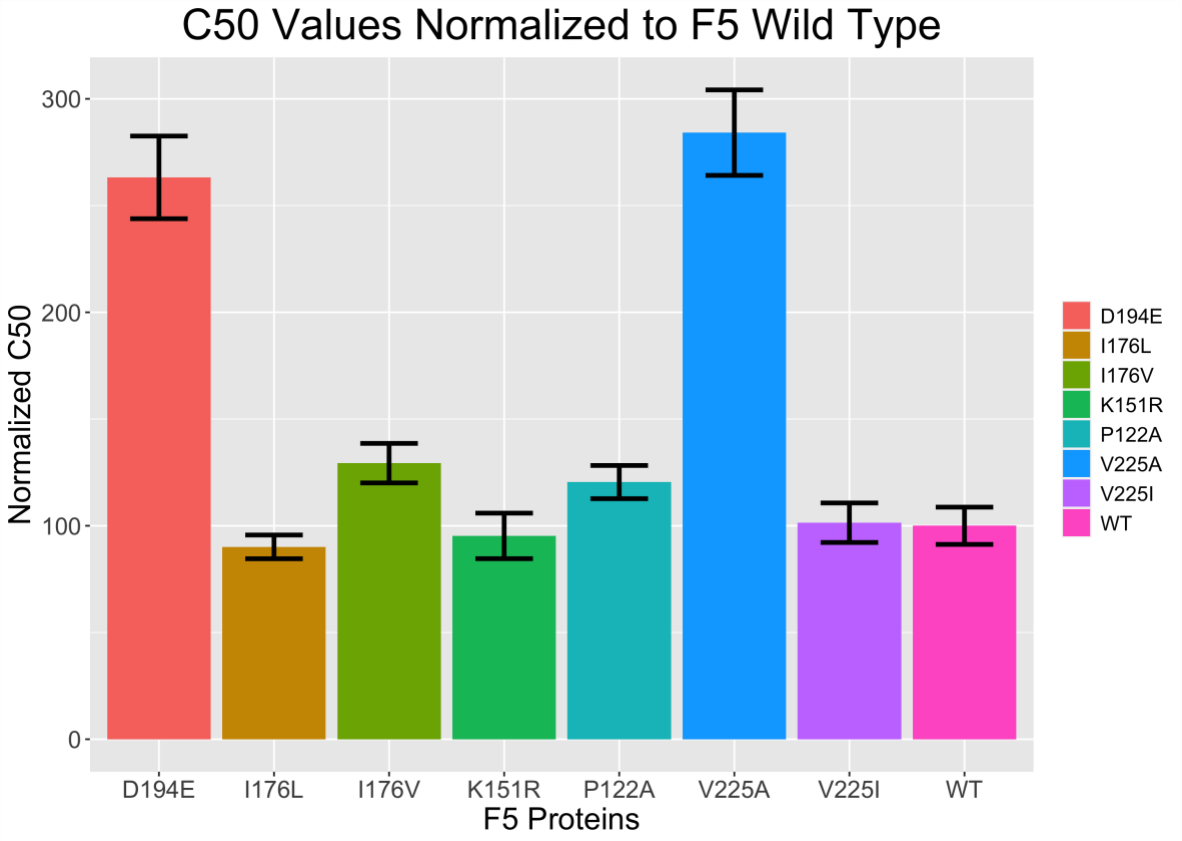

Figure 4.12: Normalized C50 bar plot of the F5 wild-type and mutants. The error bars represent a 95% confidence interval.

columns: mutant name, concentration of mutant, and average normalized absorbance. ggplot was used to plot the data to create an ELISA graph. The x axis was converted into a logarithmic axis, then using functions in the scale package, the x axis was reversed so the values were in descending numerical value. The scale function also ensured that each plot of data would be a color that was differentiated by the mutant name. Since both the bar plot and ELISA plot data was listed alphabetically by mutant, the colors from both ggplots should be consistent in the generated figures. The type of line generated for these curves was called a loess curve, which has its own function in R to generate the smooth lines that are normally observed in ELISA plots. Just as with the bar plots, .png images were generated in a size that was suitable for PowerPoints

and for journal articles. **Figure 4.13** is an example of what the plot looks like when generated.

4.7 Exceptions to Code

One of the first exceptions that required an adjustment to the code was instances when there were less than three proteins present on a plate. The way this was worked around was when the code recognizes that a protein name is “NA” instead of an actual variant when developing the normvhh data frame (refer to #normalize absorbance values in page 63 of Appendix C), the code will remove columns containing “NA” in the protein name. The code after this creates scenarios depending on the number of columns that are present in the normvhh data frame or based on the number of normvhh variants that

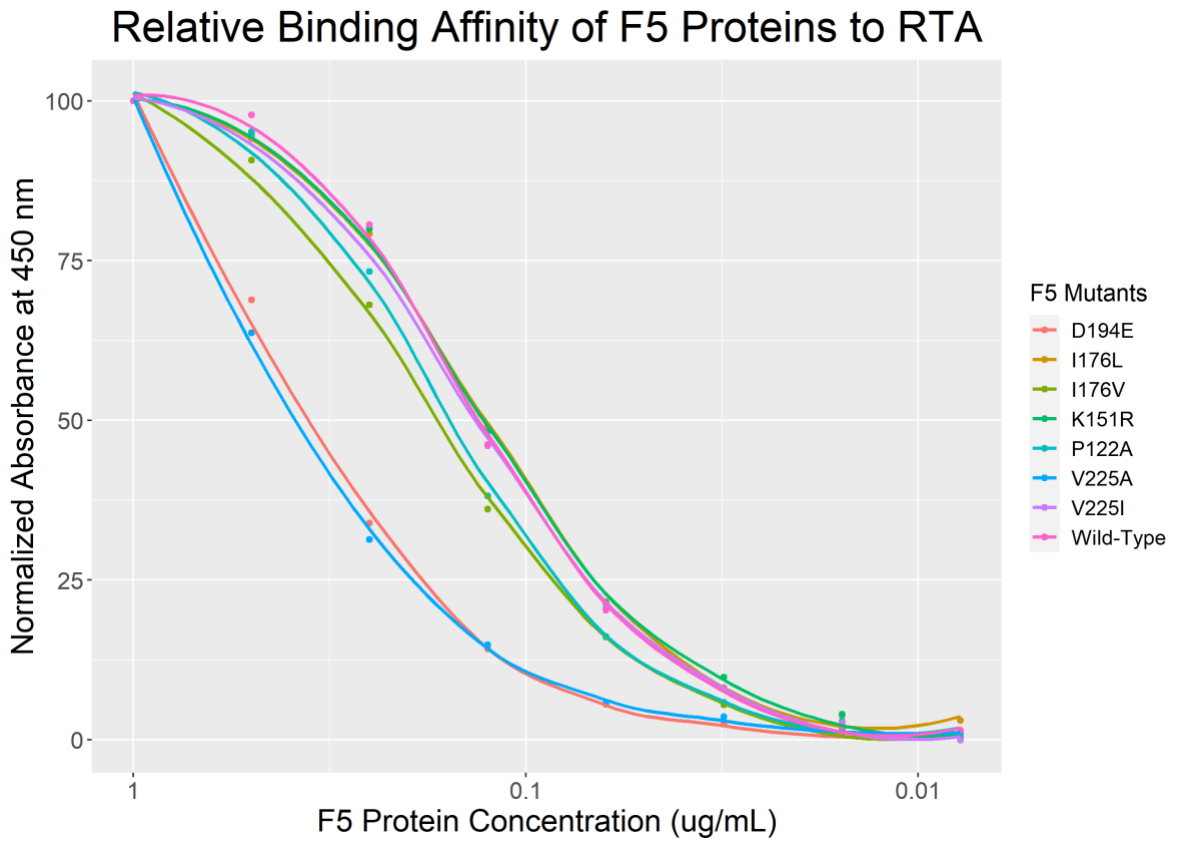

Figure 4.13: ELISA plot of the normalized absorbance values for each F5 protein at 450 nm.

have developed (normvhh1, normvhh2, normvhh3). An example of this is shown in the section of code under #average absorbance values on page 63 of Appendix C when the code generates if loops based on the number of columns present in the normvhh data frame. One protein present will have 3 columns of data in the normvhh data frame, two proteins will have 6 columns, and three proteins will have 9 columns.

Another exception that was introduced when programming was instances where there might not be data for a well. To account for this, blank wells were assigned 0 values under section #blank correct absorbance values on page 63 of Appendix C, which would be changed to NA's when appropriate, an example of this under section #normalize absorbance values on page 63 of Appendix C. There are other instances where functions can use na.rm, which tells the function that if a NA value is present, to skip over that value when performing the function or calculation. This was conducted under the section #average absorbance values on pages 63-64 of Appendix C when the average normalized absorbance values for each concentration replicate of each F5 protein was performed. If this NA well was one of the wells essential for the C_{50} calculations, then the code accounts for this by adjusting the number of x values present for the sum squares of x and for the n value when calculating the error associated with the measurement. This is under section #linear regression calculation for C50 calculations on pages 67-68 of Appendix C, where the first four lines account for the number of concentration points used in the uncertainty calculation.

As stated in 4.3, there are instances when the normalized absorbance for a set of data points is already around 50%. If calculations can prove that the lower and upper limit of the C_{50} interval contains 50%, then the concentration for that point can be used instead

of interpolating the concentration. The lines of code for this calculation are in section #extracting values per mutant above and below 50% on page 64 of Appendix C. This would mean that the mutant that fits this description would not have to go under inverse linear regression analysis since the concentration for C_{50} was verified in previous lines of code.

There are also instances when more than one plate of data needs to be analyzed. Lines will ask for the file names of a second or third plate. These occur on pages 70 and 77 in Appendix C under the variables “y” and “z”. If there are no additional plates, the code breaks the loop and proceeds onto the plot generation. If there are additional plates, then the code repeats the steps in 4.1-4.4 and puts the calculated values into the same exported .csv files as the first plate to ensure all the data is in one place to generate the bar plot and ELISA plot.

4.8 References

1. Draper, N. R.; Smith, H. *Applied Regression Analysis*, 3rd ed.; Wiley series in probability and statistics; Wiley: New York, 1998.

Appendix

Appendix A: Protein Purification Gel Images

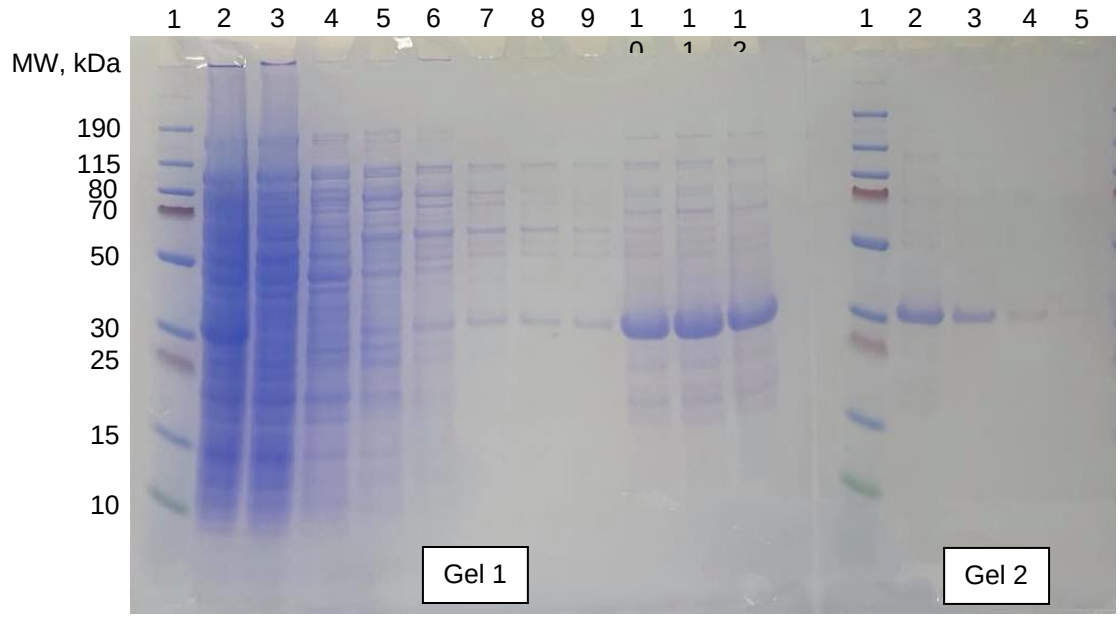

Figure A.1: SDS-PAGE of I176V F5 purification via nickel affinity chromatography. For Gel 1, Lane 1 is the protein ladder. Lane 2 is the cell lysate. Lane 3 is the flow through from the nickel column. Lanes 4-9 are the column washes. Lanes 10-11 are elutions with 250 mM imidazole. Lane 12 is an elution with 500 mM imidazole. For Gel 2, Lane 1 is the protein ladder, lane 2-5 contain elutions with 500 mM imidazole.

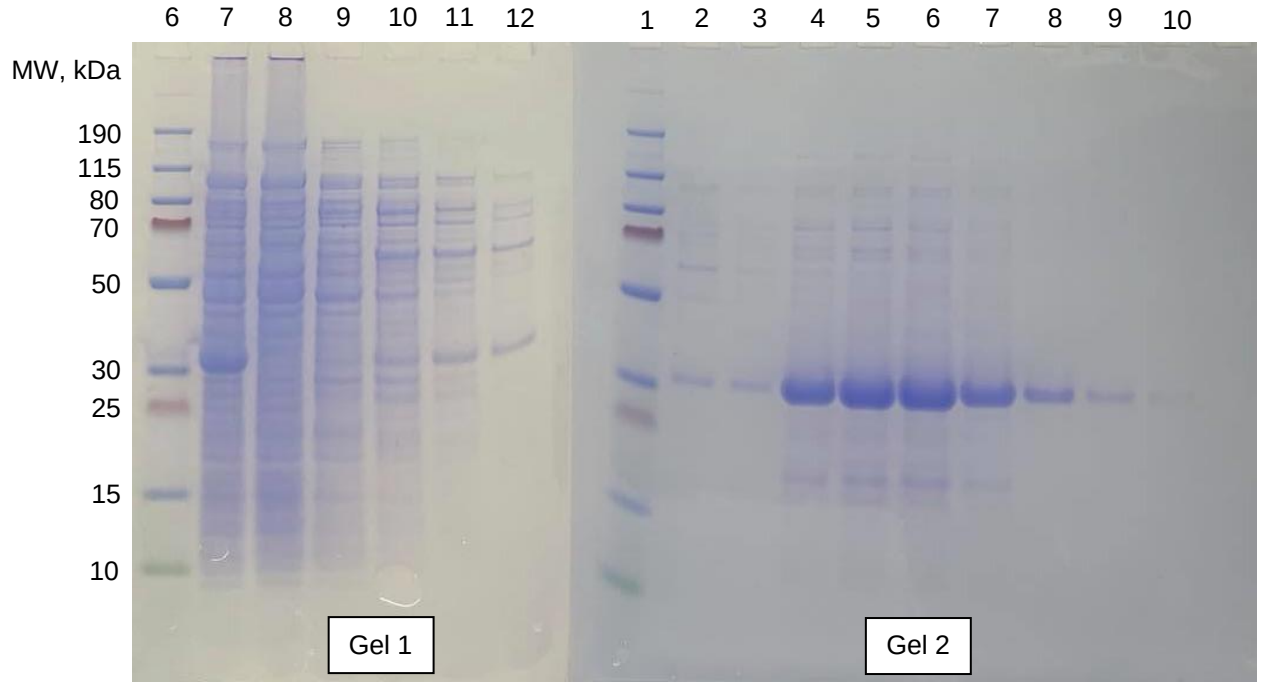

Figure A.2: SDS-PAGE of I176L F5 purification via nickel affinity chromatography. For Gel 1, Lane 6 is the protein ladder. Lane 7 is the cell lysate. Lane 8 is the flow through from the nickel column. Lanes 9-12 are column washes. For Gel 2, Lane 1 is the molecular weight ladder, lanes 2-3 are column washes. Lanes 4-5 are elutions with 250 mM imidazole. Lanes 6-10 are elutions with 500 mM imidazole.

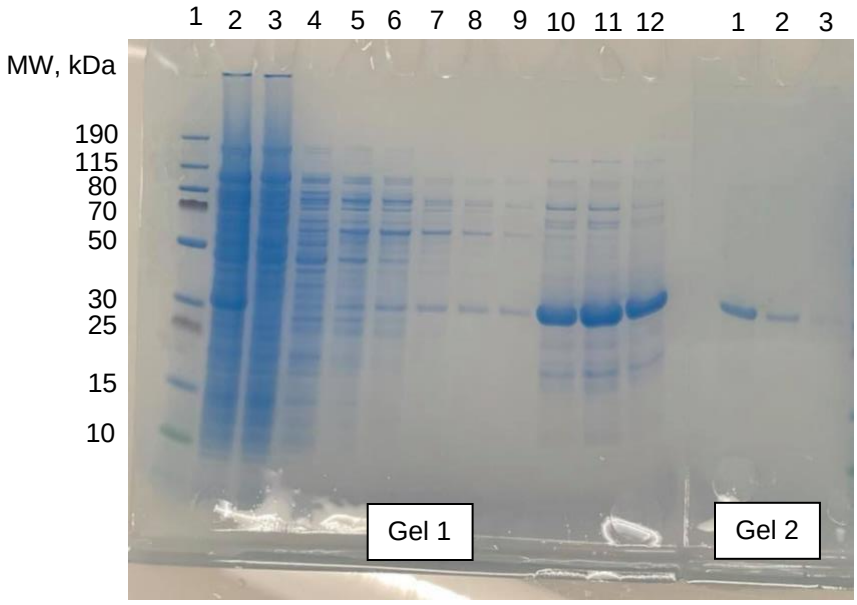

Figure A.3: SDS-PAGE of V225I F5 purification via nickel affinity chromatography. For Gel 1, Lane 1 is the protein ladder. Lane 2 is the cell lysate. Lane 3 is the flow through from the nickel column. Lanes 4-9 are the column washes. Lanes 10-11 are elutions with 250 mM imidazole. Lane 12 is an elution with 500 mM imidazole. For Gel 2, Lanes 1-3 contain elutions with 500 mM imidazole.

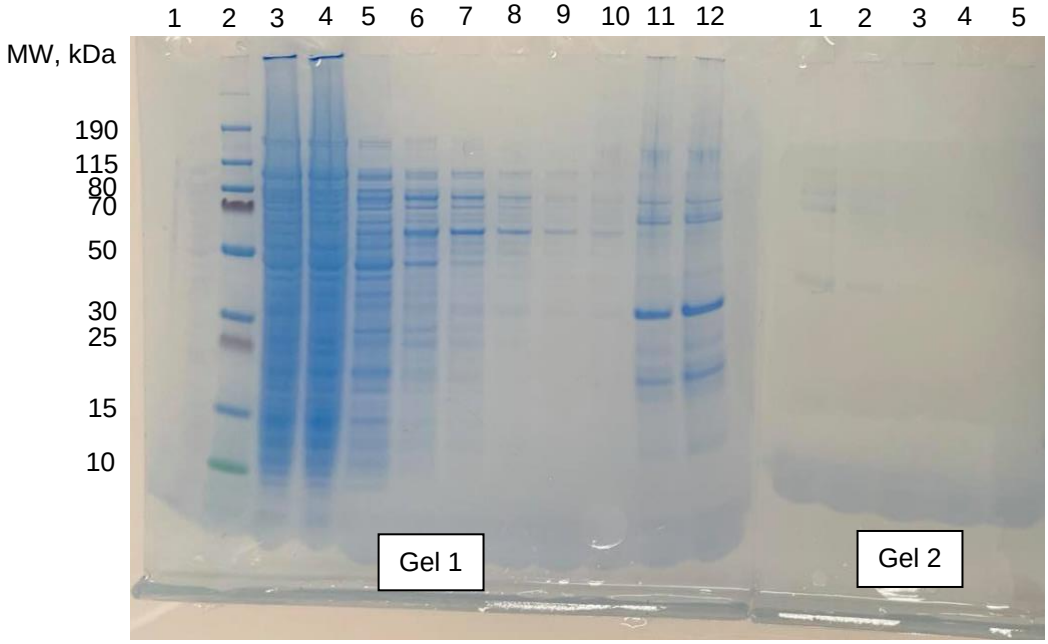

Figure A.4: SDS-PAGE of V225A F5 purification via nickel affinity chromatography. For Gel 1, lane 1 is empty. Lane 2 is the protein ladder. Lane 3 is the cell lysate. Lane 4 is the flow through from the nickel column. Lanes 5-10 are column washes. Lanes 11-12 are elutions with 250 mM imidazole. For Gel 2, Lanes 1-5 are elutions with 500 mM imidazole.

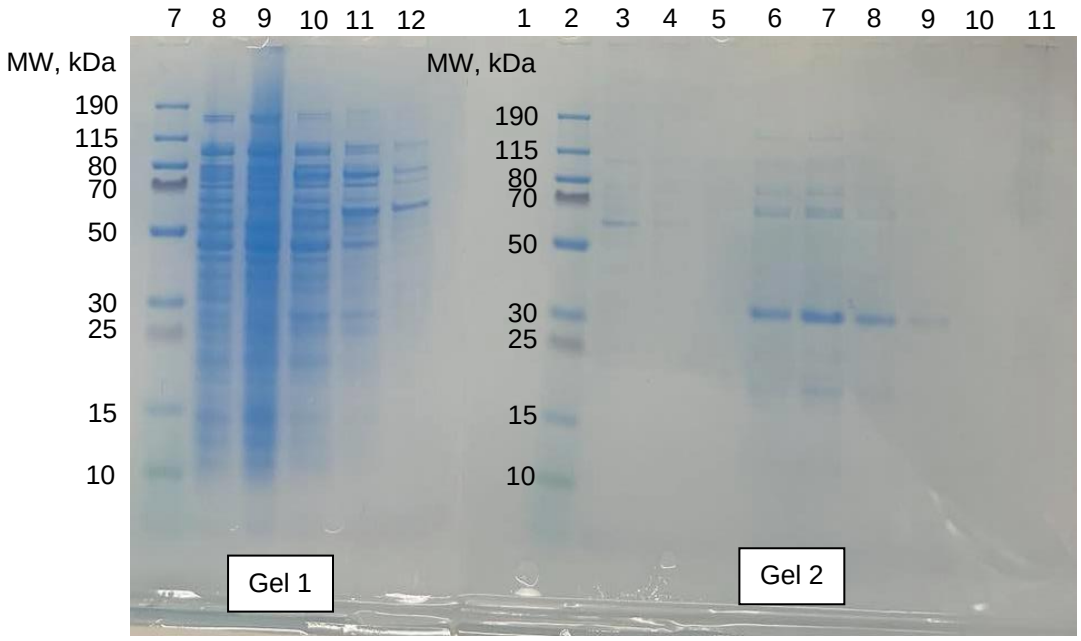

Figure A.5: SDS-PAGE of D194E F5 purification via nickel affinity chromatography. For Gel 1, Lane 7 is the protein ladder. Lane 8 is the cell lysate. Lane 9 is the flow through from the nickel column. Lanes 10-12 are column washes. For Gel 2, lane 1 is empty. Lane 2 is the molecular weight ladder. Lanes 3-5 are column washes. Lanes 6-7 are elutions with 250 mM imidazole. Lanes 8-11 are elutions with 500 mM imidazole.

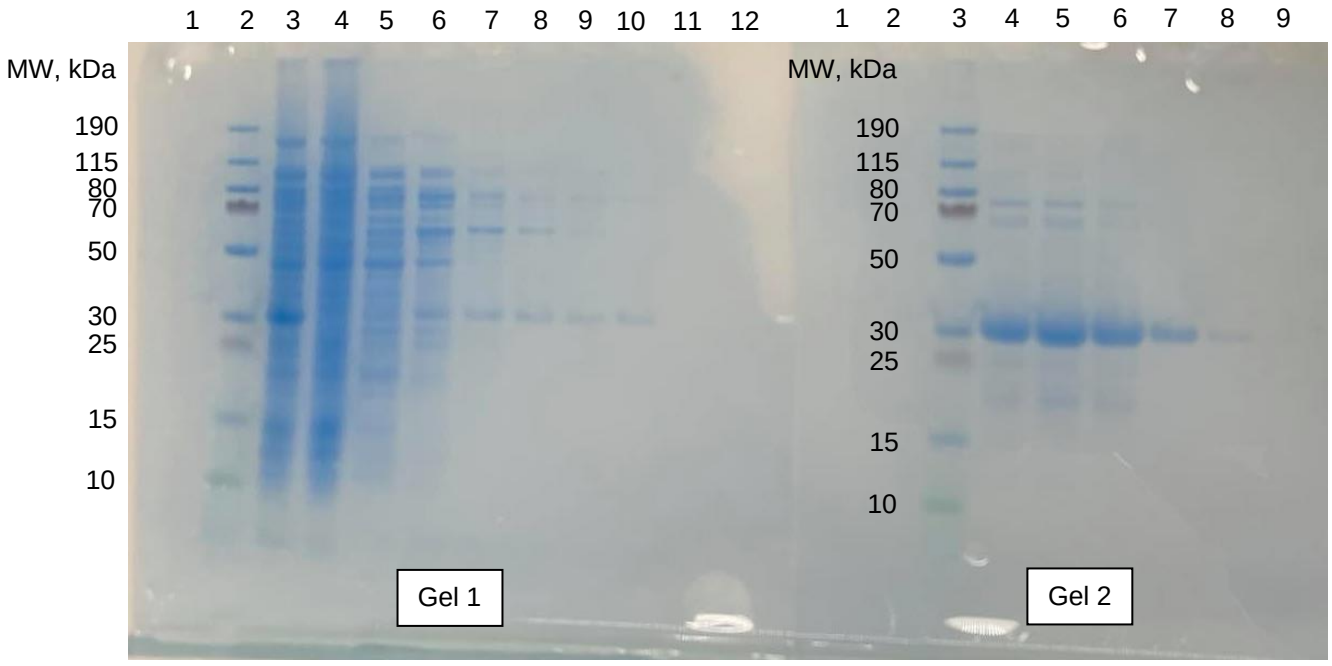

Figure A.6: SDS-PAGE of K151R F5 purification via nickel affinity chromatography. For Gel 1, Lane 1 is empty. Lane 2 is the protein ladder. Lane 3 is the cell lysate. Lane 4 is the flow through from the nickel column. Lanes 5-10 are the column washes. Lanes 11 and 12 are empty. For Gel 2, lanes 1-2 are empty, lane 3 is the protein ladder. Lanes 4-5 are elutions with 250 mM imidazole. Lanes 6-9 are elutions with 500 mM imidazole.

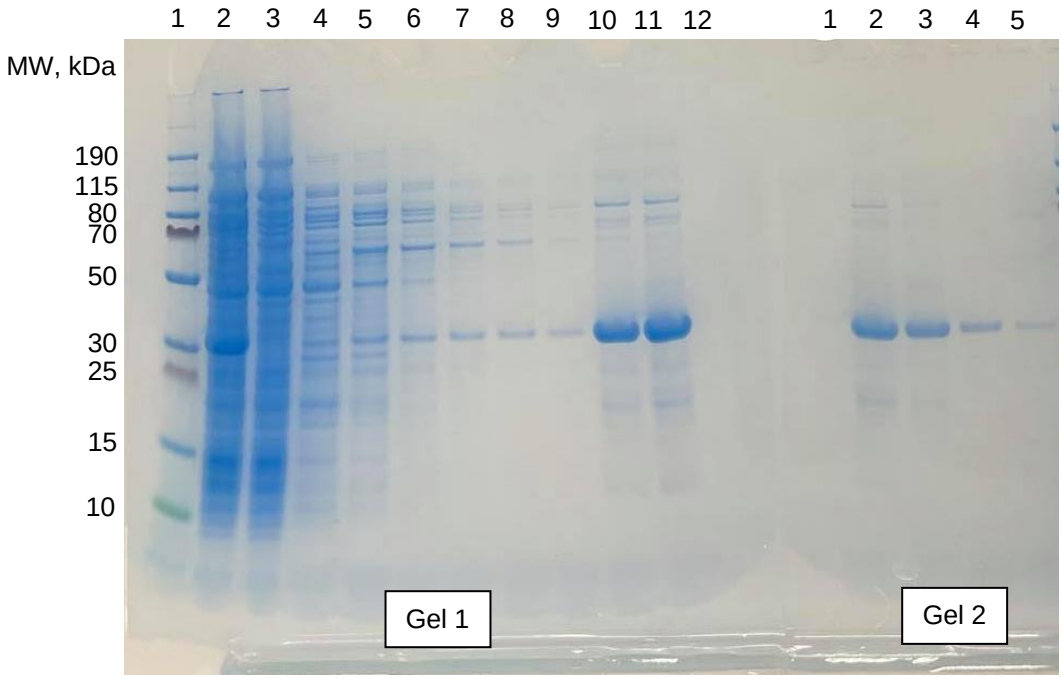

Figure A.7: SDS-PAGE of P122A F5 purification via nickel affinity chromatography. For Gel 1, Lane 1 is the protein ladder. Lane 2 is the cell lysate. Lane 3 is the flow through from the nickel column. Lanes 4-9 are the column washes. Lanes 10-11 are elutions with 250 mM imidazole. Lane 12 is empty. For Gel 2, lane 1 is empty. Lanes 2-5 contain elutions with 500 mM imidazole.

Appendix B: Deconvoluted Mass Spectra

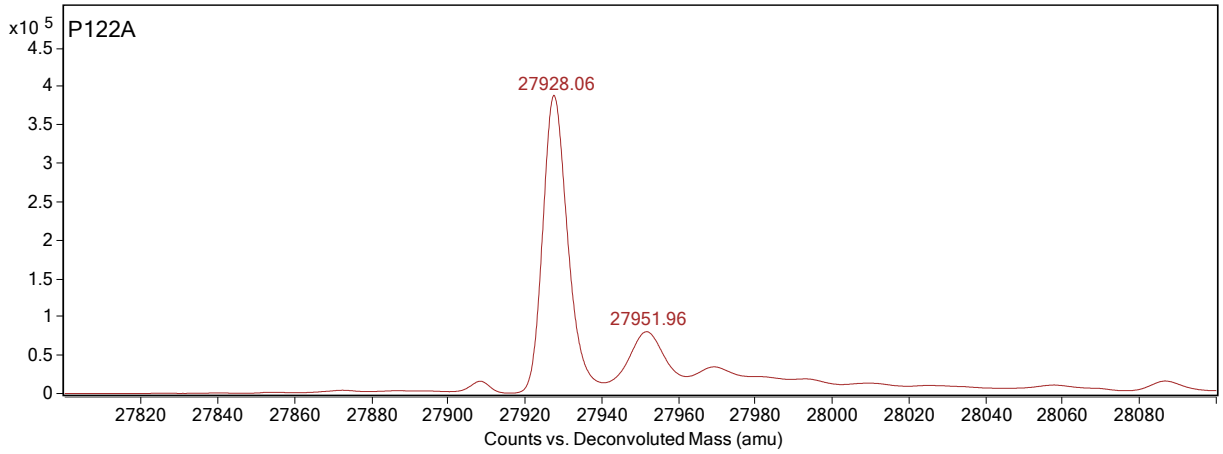

Figure B.1: Deconvoluted mass spectra of P122A F5 protein. The molecular weight of the P122A protein with both disulfide bonds intact is 27927.5 Da.

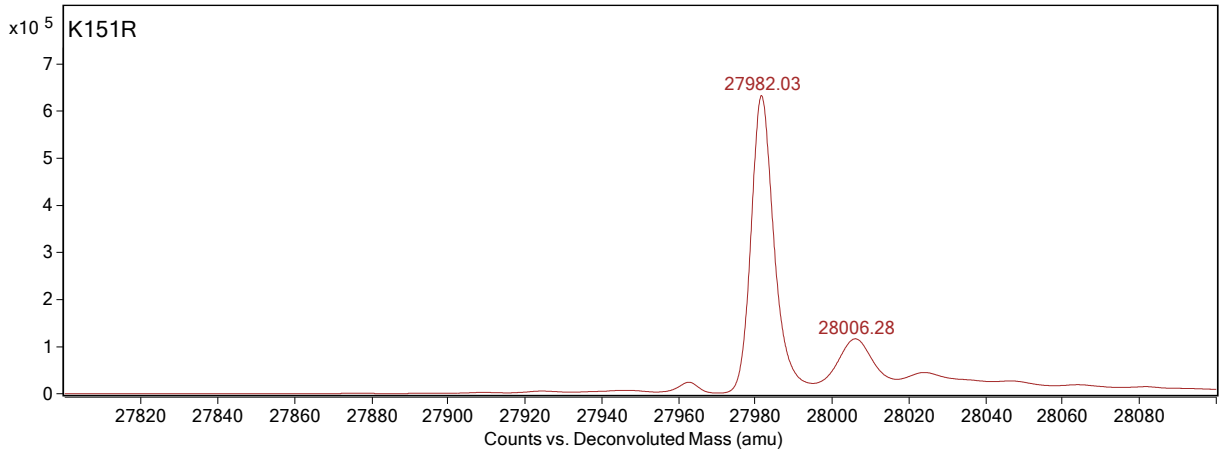

Figure B.2: Deconvoluted mass spectra of K151R F5 protein. The molecular weight of the K151R protein with both disulfide bonds intact is 27981.5 Da.

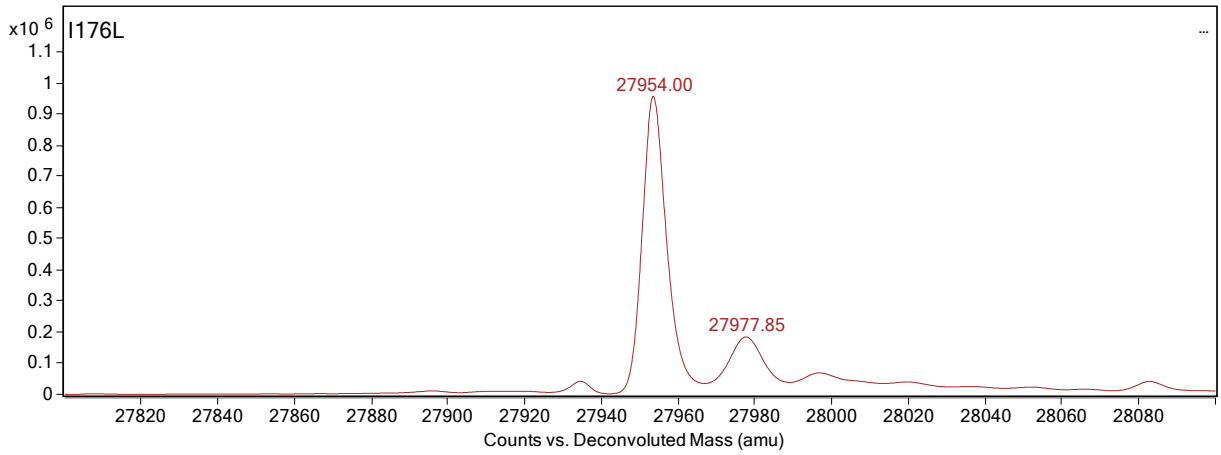

Figure B.3: Deconvoluted mass spectra of I176L F5 protein. The molecular weight of the I176L protein with both disulfide bonds intact is 27953.5 Da.

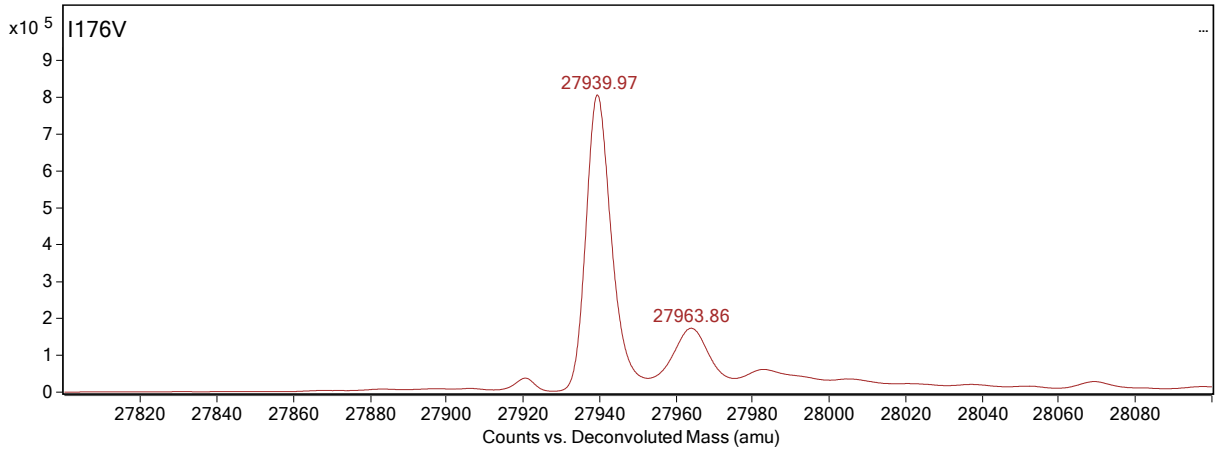

Figure B.4: Deconvoluted mass spectra of I176V F5 protein. The molecular weight of the I176V protein with both disulfide bonds intact is 27939.5 Da.

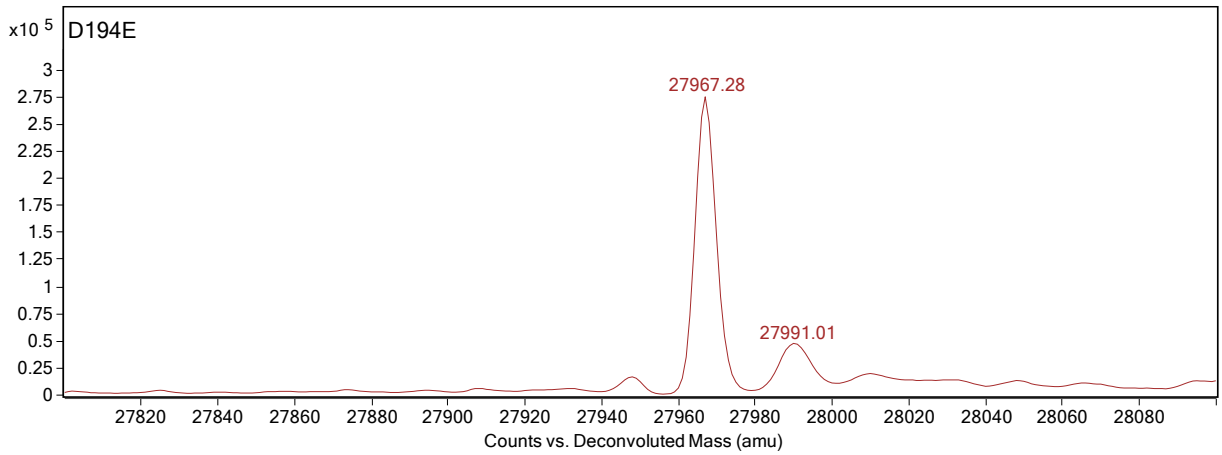

Figure B.5: Deconvoluted mass spectra of D194E F5 protein. The molecular weight of the D194E protein with both disulfide bonds intact is 27967.5 Da.

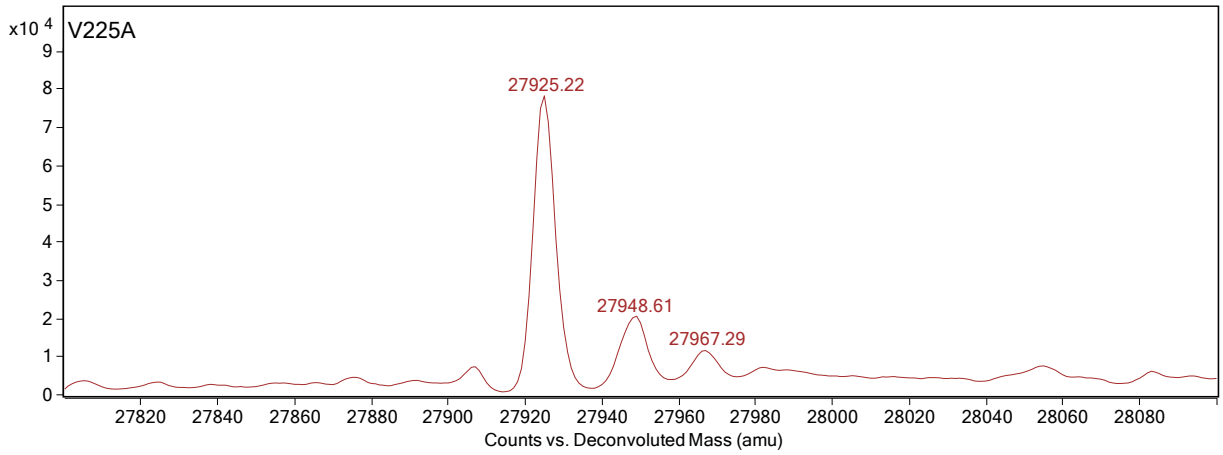

Figure B.6: Deconvoluted mass spectra of V255A F5 protein. The molecular weight of the V255A protein with both disulfide bonds intact is 27925.5 Da.

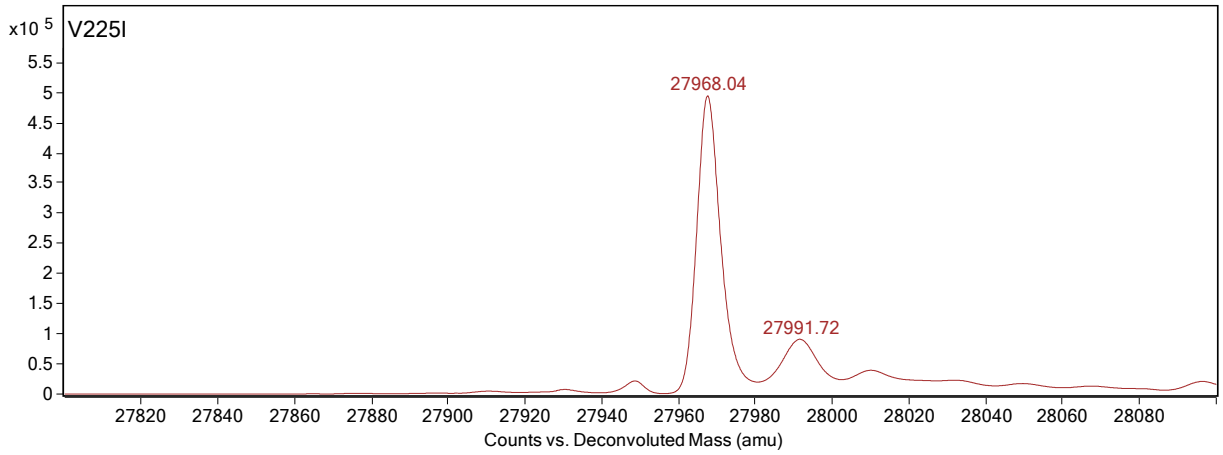

Figure B.7: Deconvoluted mass spectra of V255I F5 protein. The molecular weight of the V255I protein with both disulfide bonds intact is 27967.5 Da.

Appendix C: ELISA_VHH_R_Code.R Script

```
#install necessary packages
install.packages("PEIP") #for statistical analysis (tinv function)
install.packages("ggplot2") #for plot development
install.packages("reshape2") #for melt function (ELISA)
install.packages("scales") #for ELISA plots (to create reverse/logarithmic x axis)

vhhread <- function(...){
  library("PEIP")
  library("ggplot2")
  library("reshape2")
  library("scales")

  #user inputs for raw data files and protein names
  a <- readline("What is the user's initials?")
  x <- readline("What is the first raw data file name? (include csv extension)")
  vhh <- read.csv(file=x, header=FALSE, sep=",", dec = ".")
  vhh <- vhh[4:11, 3:14]
  p1 <- readline("What is the first protein?")
  p2 <- readline("What is the second protein? (Type 'NA' if no second protein)")
  p3 <- readline("What is the third protein? (Type 'NA' if no third protein)")
  pb1 <- paste("Blank_",p1)
  pb2 <- paste("Blank_",p2)
  pb3 <- paste("Blank_",p3)

  #Assign min value per protein as BLANK
  for (i in 1:3){
    if (i==1){
      minbs1 <- min(as.numeric(vhh[1,1:4]))
      bs1loc <- which(vhh[1,]==minbs1, arr.ind=TRUE)
      columnbs1 <- as.numeric(bs1loc[2])
      colnames(vhh)[1:4] <- p1
      colnames(vhh)[columnbs1] <- pb1
    }
    if (i==2){
      minbs2 <- min(as.numeric(vhh[1,5:8]))
      bs2loc <- which(vhh[1,]==minbs2, arr.ind=TRUE)
      columnbs2 <- as.numeric(bs2loc[2])
      colnames(vhh)[5:8] <- p2
      colnames(vhh)[columnbs2] <- pb2
    }
    if (i==3){
      minbs3 <- min(as.numeric(vhh[1,9:12]))
      bs3loc <- which(vhh[1,]==minbs3, arr.ind=TRUE)
      columnbs3 <- as.numeric(bs3loc[2])
      colnames(vhh)[9:12] <- p3
      colnames(vhh)[columnbs3] <- pb3
    }
  }
```

```
#find average value of blanks
blanks <- vector(mode="numeric", length = 3)
blank <- vhh[grepl(pattern = "\\Blank.*", x=colnames(vhh))==TRUE]
blanks[1] <- mean(as.numeric(blank[,1]))
blanks[2] <- mean(as.numeric(blank[,2]))
blanks[3] <- mean(as.numeric(blank[,3]))

#blank correct absorbance values
vhhcorr <- vhh[grepl(pattern = "\\Blank.*", x=colnames(vhh))==FALSE]
for(i in 1:9){
  if(i<=3){
    vhhcorr[,i] <- as.numeric(vhhcorr[,i])-blanks[1]
  }
  if(i>3&i<=6){
    vhhcorr[,i] <- as.numeric(vhhcorr[,i])-blanks[2]
  }
  if(i>6&i<=9){
    vhhcorr[,i] <- as.numeric(vhhcorr[,i])-blanks[3]
  }
}
vhhcorr[is.na(vhhcorr)] <- 0

#normalize absorbance values
normvhh <- vhhcorr
for(i in 1:9){
  maxab <- max(normvhh[,i])
  normvhh[,i] <- normvhh[,i]/maxab*100
}
normvhh <- normvhh[grepl(pattern = "\\.*[n|N][a|A].*", x=colnames(normvhh))==FALSE]
normvhh[normvhh==0] <- NA

#average absorbance values
avvhh <- as.data.frame(matrix(nrow=3, ncol=8))
proteinnames <- colnames(normvhh)
if (ncol(normvhh)==3){
  avvhh <- as.data.frame(matrix(nrow=1, ncol=8))
  rownames(avvhh)[1] <- c(proteinnames[1])
}
if(ncol(normvhh)==6){
  avvhh <- as.data.frame(matrix(nrow=2, ncol=8))
  rownames(avvhh)[1] <- c(proteinnames[1])
  rownames(avvhh)[2] <- c(proteinnames[4])
}
if(ncol(normvhh)==9){
  rownames(avvhh)[1] <- c(proteinnames[1])
  rownames(avvhh)[2] <- c(proteinnames[4])
  rownames(avvhh)[3] <- c(proteinnames[7])
}
for (i in 1:nrow(avvhh)){
  if (i==1){
```

```
    normvhh1 <- t(normvhh[,1:3])
    for(r in 1:8){
      avvhh[1, r] <- mean(normvhh1[,r], na.rm = TRUE)
    }
  }
  if (i==2){
    normvhh2 <- t(normvhh[,4:6])
    for(r in 1:8){
      avvhh[2, r] <- mean(normvhh2[,r], na.rm = TRUE)
    }
  }
  if (i==3){
    normvhh3 <- t(normvhh[,7:9])
    for(r in 1:8){
      avvhh[3, r] <- mean(normvhh3[,r], na.rm = TRUE)
    }
  }
}
avvhh <- t(avvhh)
rownames(avvhh) <- c(1, .5, .25, .125, .0625, .03125, .015625, .0078125)

#extracting values per mutant above and below 50%
colnames(normvhh1) <- c(1, .5, .25, .125, .0625, .03125, .015625, .0078125)
p1abs <- c()
p1con <- c()
columnnumber1 <- 0
for (i in 1:ncol(normvhh1)){
  fiftyISH <- grepl(pattern = "[5]\\d\\D", x=normvhh1[,i])
  fiftyloc <- which(fiftyISH, arr.ind = TRUE)
  if(length(fiftyloc)>0){
    columnnumber1 <- i
  }
}
if (columnnumber1 >0 ){
  av50 <- mean(normvhh1[,columnnumber1])
  std50 <- sd(normvhh1[,columnnumber1])
  tval50 <- -tinv(0.025, 2)
  conf.int.50 <- tval50 * std50 / sqrt(2)
  upper50.1 <- av50 + conf.int.50
  lower50.1 <- av50 - conf.int.50
  if (upper50.1>50 && lower50.1<50){
    c50_1.1 <- colnames(normvhh1)[columnnumber1]
    p1con[1:6] <- c50_1.1
    p1con <- as.numeric(p1con)
    p1abs[1:6] <- normvhh1[,c50_1.1]
  }
  else{break()}
  }
if(columnnumber1==0) {
  dmM1 <- melt(normvhh1, id.var=1)
  nv1 <- dmM1$value>50
```

```
  dmM.g501 <- dmM1[nv1==TRUE,]
  if (nrow(dmM.g501)%%3==0){
    p1con[1:3] <- tail(dmM.g501$Var2, n=3)
    p1abs[1:3] <- tail(dmM.g501$value, n=3)
  }
  if (nrow(dmM.g501)%%3==1){
    p1con[1:3] <- tail(dmM.g501$Var2, n=4)[1:3]
    p1abs[1:3] <- tail(dmM.g501$value, n=4)[1:3]
  }
  if (nrow(dmM.g501)%%3==2){
    p1con[1:3] <- tail(dmM.g501$Var2, n=5)[1:3]
    p1abs[1:3] <- tail(dmM.g501$value, n=5)[1:3]
  }
  dmM.l501 <- dmM1[nv1==FALSE,]
  if (nrow(dmM.l501)%%3==0){
    p1con[4:6] <- head(dmM.l501$Var2, n=3)
    p1abs[4:6] <- head(dmM.l501$value, n=3)
  }
  if (nrow(dmM.l501)%%3==1){
    p1con[4:6] <- head(dmM.l501$Var2, n=4)[2:4]
    p1abs[4:6] <- head(dmM.l501$value, n=4)[2:4]
  }
  if (nrow(dmM.l501)%%3==2){
    p1con[4:6] <- head(dmM.l501$Var2, n=5)[3:5]
    p1abs[4:6] <- head(dmM.l501$value, n=5)[3:5]
  }
}

if(ncol(avvhh)>=2){
  colnames(normvhh2) <- c(1, .5, .25, .125, .0625, .03125, .015625, .0078125)
  p2abs <- c()
  p2con <- c()
  columnnumber2 <- 0
  for (i in 1:ncol(normvhh2)){
    fiftyISH2 <- grepl(pattern = "[5]\\d\\D", x=normvhh2[,i])
    fiftyloc2 <- which(fiftyISH2, arr.ind = TRUE)
    if(length(fiftyloc2)>0){
      columnnumber2 <- i
    }
  }
  if (columnnumber2 >0 ){
    av50 <- mean(normvhh2[,columnnumber2])
    std50 <- sd(normvhh2[,columnnumber2])
    tval50 <- -tinv(0.025, 2)
    conf.int.50 <- tval50 * std50 / sqrt(2)
    upper50.2 <- av50 + conf.int.50
    lower50.2 <- av50 - conf.int.50
    if (upper50.2>50 && lower50.2<50){
      c50_2.1 <- colnames(normvhh2)[columnnumber2]
      p2con[1:6] <- c50_2.1
      p2con <- as.numeric(p2con)
```

```
      p2abs[1:6] <- normvhh2[,c50_2.1]
    }
    else{
      break()
    }
  }
  else{
    dmM2 <- melt(normvhh2, id.var=1)
    nv2 <- dmM2$value>50
    dmM.g502 <- dmM2[nv2==TRUE,]
    if (nrow(dmM.g502)%%3==0){
      p2con[1:3] <- tail(dmM.g502$Var2, n=3)
      p2abs[1:3] <- tail(dmM.g502$value, n=3)

    }
    if (nrow(dmM.g502)%%3==1){
      p2con[1:3] <- tail(dmM.g502$Var2, n=4)[1:3]
      p2abs[1:3] <- tail(dmM.g502$value, n=4)[1:3]
    }
    if (nrow(dmM.g502)%%3==2){
      p2con[1:3] <- tail(dmM.g502$Var2, n=5)[1:3]
      p2abs[1:3] <- tail(dmM.g502$value, n=5)[1:3]
    }
    dmM.l502 <- dmM2[nv2==FALSE,]
    if (nrow(dmM.l502)%%3==0){
      p2con[4:6] <- head(dmM.l502$Var2, n=3)
      p2abs[4:6] <- head(dmM.l502$value, n=3)
    }
    if (nrow(dmM.l502)%%3==1){
      p2con[4:6] <- head(dmM.l502$Var2, n=4)[2:4]
      p2abs[4:6] <- head(dmM.l502$value, n=4)[2:4]
    }
    if (nrow(dmM.l502)%%3==2){
      p2con[4:6] <- head(dmM.l502$Var2, n=5)[3:5]
      p2abs[4:6] <- head(dmM.l502$value, n=5)[3:5]
    }
  }
}
if(ncol(avvhh)>=3){
  colnames(normvhh3) <- c(1, .5, .25, .125, .0625, .03125, .015625, .0078125)
  p3abs <- c()
  p3con <- c()
  columnnumber3 <- 0
  for (i in 1:ncol(normvhh3)){
    fiftyISH3 <- grepl(pattern = "[5]\\d\\D", x=normvhh3[,i])
    fiftyloc3 <- which(fiftyISH3, arr.ind = TRUE)
    if(length(fiftyloc3)>0){
      columnnumber3 <- i
    }
  }
  if (columnnumber3 >0 ){
```

```
    av50 <- mean(normvhh3[,columnnumber3])
    std50 <- sd(normvhh3[,columnnumber3])
    tval50 <- -tinv(0.025, 2)
    conf.int.50 <- tval50 * std50 / sqrt(2)
    upper50.3 <- av50 + conf.int.50
    lower50.3 <- av50 - conf.int.50
    if (upper50.3>50 && lower50.3<50){
      c50_3.1 <- colnames(normvhh3)[columnnumber3]
      p3con[1:6] <- c50_3.1
      p3con <- as.numeric(p3con)
      p3abs[1:6] <- normvhh3[,c50_3.1]
    }
    else{
      break()
    }
  }
  else{
    dmM3 <- melt(normvhh3, id.var=1)
    nv3 <- dmM3$value>50
    dmM.g503 <- dmM3[nv3==TRUE,]
    if (nrow(dmM.g503)%%3==0){
      p3con[1:3] <- tail(dmM.g503$Var2, n=3)
      p3abs[1:3] <- tail(dmM.g503$value, n=3)
      }
    if (nrow(dmM.g503)%%3==1){
      p3con[1:3] <- tail(dmM.g503$Var2, n=4)[1:3]
      p3abs[1:3] <- tail(dmM.g503$value, n=4)[1:3]
      }
    if (nrow(dmM.g503)%%3==2){
      p3con[1:3] <- tail(dmM.g503$Var2, n=5)[1:3]
      p3abs[1:3] <- tail(dmM.g503$value, n=5)[1:3]
      }
    dmM.l503 <- dmM3[nv3==FALSE,]
    if (nrow(dmM.l503)%%3==0){
      p3con[4:6] <- head(dmM.l503$Var2, n=3)
      p3abs[4:6] <- head(dmM.l503$value, n=3)
      }
    if (nrow(dmM.l503)%%3==1){
      p3con[4:6] <- head(dmM.l503$Var2, n=4)[2:4]
      p3abs[4:6] <- head(dmM.l503$value, n=4)[2:4]
      }
    if (nrow(dmM.l503)%%3==2){
      p3con[4:6] <- head(dmM.l503$Var2, n=5)[3:5]
      p3abs[4:6] <- head(dmM.l503$value, n=5)[3:5]
    }
  }
}

#linear regression calculation for C50 calculations
p1con <- na.omit(p1con)
p1abs <- na.omit(p1abs)
```

```
  p1con.count1.1 <- sum(p1con==p1con[1])
  p1con.count1.2 <- sum(p1con==p1con[4])
  p1count <- p1con.count1.1+p1con.count1.2
  p1reg <- lm(p1abs~p1con)
  p1data <- summary(p1reg)
  p1xav <- mean(p1con)
  p1yav <- mean(p1abs)
  p1slope <- as.numeric(p1reg$coefficients[2])
  p1sigma <- as.numeric(p1data$sigma)
  p1t <- qt(0.975, p1count)
  sqrt <- (((50-p1yav)^2)/((p1con.count1.1*(p1con[1]-p1xav)^2)+(p1con.count1.2*(p1con[4]-p1xa
v)^2)))+((p1slope^2)/p1count)-(((p1t^2)*(p1sigma^2))/(p1count*((p1con.count1.1*(p1con[1]-p1xa
v)^2)+(p1con.count1.2*(p1con[4]-p1xav)^2))))
  p1upper <- p1xav + ((p1slope*(50-p1yav))+(p1t*p1sigma*(sqrt)^(1/2)))/((p1slope^2)-((p1t^2)*(p
1sigma^2)/p1count))
  p1lower <- p1xav + ((p1slope*(50-p1yav))-(p1t*p1sigma*(sqrt)^(1/2)))/((p1slope^2)-((p1t^2)*(p1
sigma^2)/p1count))
  p1c50 <- (p1upper-p1lower)/2+p1lower
  p1c50error <- p1c50-p1lower
  if(is.numeric(p1c50error)!=TRUE){
    p1c50<-as.numeric(c50_1.1)
    p1c50error<-0
  }
  c50stdmatrix <- p1c50error
  c50valmatrix <- p1c50

  if(ncol(avvhh)>=2){
    p2con <- na.omit(p2con)
    p2abs <- na.omit(p2abs)
    p2con.count1.1 <- sum(p2con==p2con[1])
    p2con.count1.2 <- sum(p2con==p2con[4])
    p2count <- p2con.count1.1+p2con.count1.2
    p2reg <- lm(p2abs~p2con)
    p2data <- summary(p2reg)
    p2xav <- mean(p2con)
    p2yav <- mean(p2abs)
    p2slope <- as.numeric(p2reg$coefficients[2])
    p2sigma <- as.numeric(p2data$sigma)
    p2t <- qt(0.975, p2count)
    sqrt <- (((50-p2yav)^2)/((p2con.count1.1*(p2con[1]-p2xav)^2)+(p2con.count1.2*(p2con[4]-p2x
av)^2)))+((p2slope^2)/p2count)-(((p2t^2)*(p2sigma^2))/(p2count*((p2con.count1.1*(p2con[1]-p2x
av)^2)+(p2con.count1.2*(p2con[4]-p2xav)^2))))
    p2upper <- p2xav + ((p2slope*(50-p2yav))+(p2t*p2sigma*(sqrt)^(1/2)))/((p2slope^2)-((p2t^2)*(
p2sigma^2)/p2count))
    p2lower <- p2xav + ((p2slope*(50-p2yav))-(p2t*p2sigma*(sqrt)^(1/2)))/((p2slope^2)-((p2t^2)*(p
2sigma^2)/p2count))
    p2c50 <- (p2upper-p2lower)/2+p2lower
    p2c50error <- p2c50-p2lower
    if(is.numeric(p2c50error)!=TRUE){
      p2c50<-as.numeric(c50_2.1)
      p2c50error<-0
```

```
  }
  c50stdmatrix <- cbind(c50stdmatrix, p2c50error)
  c50valmatrix <- cbind(c50valmatrix, p2c50)
 }
 if(ncol(avvhh)>=3){
  p3con <- na.omit(p3con)
  p3abs <- na.omit(p3abs)
  p3con.count1.1 <- sum(p3con==p3con[1])
  p3con.count1.2 <- sum(p3con==p3con[4])
  p3count <- p3con.count1.1+p3con.count1.2
  p3reg <- lm(p3abs~p3con)
  p3data <- summary(p3reg)
  p3xav <- mean(p3con)
  p3yav <- mean(p3abs)
  p3slope <- as.numeric(p3reg$coefficients[2])
  p3sigma <- as.numeric(p3data$sigma)
  p3t <- qt(0.975, p3count)
  sqrt <- (((50-p3yav)^2)/((p3con.count1.1*(p3con[1]-p3xav)^2)+(p3con.count1.2*(p3con[4]-p3x
av)^2)))+((p3slope^2)/p3count)-(((p3t^2)*(p3sigma^2))/(p3count*((p3con.count1.1*(p3con[1]-p3x
av)^2)+(p3con.count1.2*(p3con[4]-p3xav)^2))))
  p3upper <- p3xav + ((p3slope*(50-p3yav))+(p3t*p3sigma*(sqrt)^(1/2)))/((p3slope^2)-((p3t^2)*(
p3sigma^2)/p3count))
  p3lower <- p3xav + ((p3slope*(50-p3yav))-(p3t*p3sigma*(sqrt)^(1/2)))/((p3slope^2)-((p3t^2)*(p
3sigma^2)/p3count))
  p3c50 <- (p3upper-p3lower)/2+p3lower
  p3c50error <- p3c50-p3lower
  if(is.numeric(p3c50error)!=TRUE){
   p3c50<-as.numeric(c50_3.1)
   p3c50error<-0
  }
  c50stdmatrix <- cbind(c50stdmatrix, p3c50error)
  c50valmatrix <- cbind(c50valmatrix, p3c50)
 }

 c50matrix <- rbind(c50valmatrix, c50stdmatrix)
 c50df <- as.data.frame(t(c50matrix))
 for (i in 1:ncol(avvhh)){
  if (i==1){
   rownames(c50df)[1] <- c(p1)
  }
  if (i==2){
   rownames(c50df)[1] <- c(p1)
   rownames(c50df)[2] <- c(p2)
  }
  if (i==3){
   rownames(c50df)[1] <- c(p1)
   rownames(c50df)[2] <- c(p2)
   rownames(c50df)[3] <- c(p3)
  }
 }
 colnames(c50df) <- c("C50", "error")
```

```
 c50df <- t(c50df)

 y <- readline("What is the second raw data file name? (include csv extension) (Type 'NA' if no f
ile) ")
 if(y!="NA"){
  vhhy <- read.csv(file=y, header=FALSE, sep=",", dec = ".")
  vhhy <- vhhy[4:11, 3:14]
  p1y <- readline("What is the first protein?")
  p2y <- readline("What is the second protein? (Type 'NA' if no second protein)")
  p3y <- readline("What is the third protein? (Type 'NA' if no third protein)")
  pb1y <- paste("Blank_",p1y)
  pb2y <- paste("Blank_",p2y)
  pb3y <- paste("Blank_",p3y)

  #See concentration assignment below to assign 1:4 or 5:8 or 9:12 with min value as BLANK
  for (i in 1:3){
   if (i==1){
    minbs1y <- min(as.numeric(vhhy[1,1:4]))
    bs1locy <- which(vhhy==minbs1y, arr.ind=TRUE)
    columnbs1y <- as.numeric(bs1locy[2])
    colnames(vhhy)[1:4] <- p1y
    colnames(vhhy)[columnbs1y] <- pb1y
   }
   if (i==2){
    minbs2y <- min(as.numeric(vhhy[1,5:8]))
    bs2locy <- which(vhhy==minbs2y, arr.ind=TRUE)
    columnbs2y <- as.numeric(bs2locy[2])
    colnames(vhhy)[5:8] <- p2y
    colnames(vhhy)[columnbs2y] <- pb2y
   }
   if (i==3){
    minbs3y <- min(as.numeric(vhhy[1,9:12]))
    bs3locy <- which(vhhy==minbs3y, arr.ind=TRUE)
    columnbs3y <- as.numeric(bs3locy[2])
    colnames(vhhy)[9:12] <- p3y
    colnames(vhhy)[columnbs3y] <- pb3y
   }
  }

  #find average value of blanks
  blanksy <- vector(mode="numeric", length = 3)
  blanky <- vhhy[grepl(pattern = "\\Blank.*", x=colnames(vhhy))==TRUE]
  blanksy[1] <- mean(as.numeric(blanky[,1]))
  blanksy[2] <- mean(as.numeric(blanky[,2]))
  blanksy[3] <- mean(as.numeric(blanky[,3]))

  #blank correct absorbance values
  vhhcorry <- vhhy[grepl(pattern = "\\Blank.*", x=colnames(vhhy))==FALSE]
  for(i in 1:9){
   if(i<=3){
    vhhcorry[,i] <- as.numeric(vhhcorry[,i])-blanksy[1]
```

```
  }
  if(i>3&i<=6){
    vhhcorry[,i] <- as.numeric(vhhcorry[,i])-blanksy[2]
  }
  if(i>6&i<=9){
    vhhcorry[,i] <- as.numeric(vhhcorry[,i])-blanksy[3]
  }
}
vhhcorry[is.na(vhhcorry)] <- 0

#normalize absorbance values
normvhhy <- vhhcorry
for(i in 1:9){
  maxaby <- max(normvhhy[,i])
  normvhhy[,i] <- normvhhy[,i]/maxaby*100
}
normvhhy <- normvhhy[grepl(pattern = "\\.*[n|N][a|A].*", x=colnames(normvhhy))==FALSE]
normvhhy[normvhhy==0] <- NA

#average absorbance values
avvhhy <- as.data.frame(matrix(nrow=3, ncol=8))
proteinnamesy <- colnames(normvhhy)
if (ncol(normvhhy)==3){
  avvhhy <- as.data.frame(matrix(nrow=1, ncol=8))
  rownames(avvhhy)[1] <- c(proteinnamesy[1])
}
if(ncol(normvhhy)==6){
  avvhhy <- as.data.frame(matrix(nrow=2, ncol=8))
  rownames(avvhhy)[1] <- c(proteinnamesy[1])
  rownames(avvhhy)[2] <- c(proteinnamesy[4])
}
if(ncol(normvhhy)==9){
  rownames(avvhhy)[1] <- c(proteinnamesy[1])
  rownames(avvhhy)[2] <- c(proteinnamesy[4])
  rownames(avvhhy)[3] <- c(proteinnamesy[7])
}
for (i in 1:nrow(avvhhy)){
  if (i==1){
    normvhh1y <- t(normvhhy[,1:3])
    for(r in 1:8){
      avvhhy[1, r] <- mean(normvhh1y[,r], na.rm = TRUE)
    }
  }
  if (i==2){
    normvhh2y <- t(normvhhy[,4:6])
    for(r in 1:8){
      avvhhy[2, r] <- mean(normvhh2y[,r], na.rm = TRUE)
    }
  }
  if (i==3){
    normvhh3y <- t(normvhhy[,7:9])
```

```
    for(r in 1:8){
      avvhhy[3, r] <- mean(normvhh3y[,r], na.rm = TRUE)
    }
  }
}
avvhhy <- t(avvhhy)
rownames(avvhhy) <- c(1, .5, .25, .125, .0625, .03125, .015625, .0078125)

#extracting values per mutant above and below 50%
colnames(normvhh1y) <- c(1, .5, .25, .125, .0625, .03125, .015625, .0078125)
p1absy <- c()
p1cony <- c()
columnnumber1y <- 0
for (i in 1:ncol(normvhh1y)){
  fiftyISHy <- grepl(pattern = "[5]\\d\\D", x=normvhh1y[,i])
  fiftylocy <- which(fiftyISHy, arr.ind = TRUE)
  if(length(fiftylocy)>0){
    columnnumber1y <- i
  }
}
if (columnnumber1y >0 ){
  av50y <- mean(normvhh1y[,columnnumber1y])
  std50y <- sd(normvhh1y[,columnnumber1y])
  tval50y <- -tinv(0.025, 2)
  conf.int.50y <- tval50y * std50y / sqrt(2)
  upper50.1y <- av50y + conf.int.50y
  lower50.1y <- av50y - conf.int.50y
  if (upper50.1y>50 && lower50.1y<50){
    c50_1.1y <- colnames(normvhh1y)[columnnumber1y]
    p1cony[1:6] <- c50_1.1y
    p1cony <- as.numeric(p1cony)
    p1absy[1:6] <- normvhh1y[,c50_1.1y]
  }
  else{
    break()
    }
}
else {
  dmM1y <- melt(normvhh1y, id.var=1)
  nv1y <- dmM1y$value>50
  dmM.g501y <- dmM1y[nv1y==TRUE,]
  if (nrow(dmM.g501y)%%3==0){
    p1cony[1:3] <- tail(dmM.g501y$Var2, n=3)
    p1absy[1:3] <- tail(dmM.g501y$value, n=3)
  }
  if (nrow(dmM.g501y)%%3==1){
    p1cony[1:3] <- tail(dmM.g501y$Var2, n=4)[1:3]
    p1absy[1:3] <- tail(dmM.g501y$value, n=4)[1:3]
  }
  if (nrow(dmM.g501y)%%3==2){
    p1cony[1:3] <- tail(dmM.g501y$Var2, n=5)[1:3]
```

```
    p1absy[1:3] <- tail(dmM.g501y$value, n=5)[1:3]
  }
  dmM.l501y <- dmM1y[nv1y==FALSE,]
  if (nrow(dmM.l501y)%%3==0){
    p1cony[4:6] <- head(dmM.l501y$Var2, n=3)
    p1absy[4:6] <- head(dmM.l501y$value, n=3)
  }
  if (nrow(dmM.l501y)%%3==1){
    p1cony[4:6] <- head(dmM.l501y$Var2, n=4)[2:4]
    p1absy[4:6] <- head(dmM.l501y$value, n=4)[2:4]
  }
  if (nrow(dmM.l501y)%%3==2){
    p1cony[4:6] <- head(dmM.l501y$Var2, n=5)[3:5]
    p1absy[4:6] <- head(dmM.l501y$value, n=5)[3:5]
  }
}

if(ncol(avvhhy)>=2){
  colnames(normvhh2y) <- c(1, .5, .25, .125, .0625, .03125, .015625, .0078125)
  p2absy <- c()
  p2cony <- c()
  columnnumber2y <- 0
  for (i in 1:ncol(normvhh2y)){
    fiftyISHy <- grepl(pattern = "[5]\\d\\D", x=normvhh2y[,i])
    fiftylocy <- which(fiftyISHy, arr.ind = TRUE)
    if(length(fiftylocy)>0){
      columnnumber2y <- i
    }
  }
  if (columnnumber2y >0 ){
    av50y <- mean(normvhh2y[,columnnumber2y])
    std50y <- sd(normvhh2y[,columnnumber2y])
    tval50y <- -tinv(0.025, 2)
    conf.int.50y <- tval50y * std50y / sqrt(2)
    upper50.2y <- av50y + conf.int.50y
    lower50.2y <- av50y - conf.int.50y
    if (upper50.2y>50 && lower50.2y<50){
      c50_2.1y <- colnames(normvhh2y)[columnnumber2y]
      p2cony[1:6] <- c50_2.1y
      p2cony <- as.numeric(p2cony)
      p2absy[1:6] <- normvhh2y[,c50_2.1y]
    }
    else {
      break()
    }
  }
  else {
    dmM2y <- melt(normvhh2y, id.var=1)
    nv2y <- dmM2y$value>50
    dmM.g502y <- dmM2y[nv2y==TRUE,]
    if (nrow(dmM.g502y)%%3==0){
```

```
      p2cony[1:3] <- tail(dmM.g502y$Var2, n=3)
      p2absy[1:3] <- tail(dmM.g502y$value, n=3)
    }
    if (nrow(dmM.g502y)%%3==1){
      p2cony[1:3] <- tail(dmM.g502y$Var2, n=4)[1:3]
      p2absy[1:3] <- tail(dmM.g502y$value, n=4)[1:3]
    }
    if (nrow(dmM.g502y)%%3==2){
      p2cony[1:3] <- tail(dmM.g502y$Var2, n=5)[1:3]
      p2absy[1:3] <- tail(dmM.g502y$value, n=5)[1:3]
    }
    dmM.l502y <- dmM2y[nv2y==FALSE,]
    if (nrow(dmM.l502y)%%3==0){
      p2cony[4:6] <- head(dmM.l502y$Var2, n=3)
      p2absy[4:6] <- head(dmM.l502y$value, n=3)
    }
    if (nrow(dmM.l502y)%%3==1){
      p2cony[4:6] <- head(dmM.l502y$Var2, n=4)[2:4]
      p2absy[4:6] <- head(dmM.l502y$value, n=4)[2:4]
    }
    if (nrow(dmM.l502y)%%3==2){
      p2cony[4:6] <- head(dmM.l502y$Var2, n=5)[3:5]
      p2absy[4:6] <- head(dmM.l502y$value, n=5)[3:5]
    }
  }
}
if(ncol(avvhhy)>=3){
  colnames(normvhh3y) <- c(1, .5, .25, .125, .0625, .03125, .015625, .0078125)
  p3absy <- c()
  p3cony <- c()
  columnnumber3y <- 0
  for (i in 1:ncol(normvhh3y)){
    fiftyISHy <- grepl(pattern = "[5]\\d\\D", x=normvhh3y[,i])
    fiftylocy <- which(fiftyISHy, arr.ind = TRUE)
    if(length(fiftylocy)>0){
      columnnumber3y <- i
    }
  }
  if (columnnumber3y >0 ){
    av50y <- mean(normvhh3y[,columnnumber3y])
    std50y <- sd(normvhh3y[,columnnumber3y])
    tval50y <- -tinv(0.025, 2)
    conf.int.50y <- tval50y * std50y / sqrt(2)
    upper50.3y <- av50y + conf.int.50y
    lower50.3y <- av50y - conf.int.50y
    if (upper50.3y>50 && lower50.3y<50){
      c50_3.1y <- colnames(normvhh3y)[columnnumber3y]
      p3cony[1:6] <- c50_3.1y
      p3cony <- as.numeric(p3cony)
      p3absy[1:6] <- normvhh2y[,c50_3.1y]
    }
```

```
      else{
      break()
      }
    }
    else{
      dmM3y <- melt(normvhh3y, id.var=1)
      nv3y <- dmM3y$value>50
      dmM.g503y <- dmM3y[nv3y==TRUE,]
      if (nrow(dmM.g503y)%%3==0){
        p3cony[1:3] <- tail(dmM.g503y$Var2, n=3)
        p3absy[1:3] <- tail(dmM.g503y$value, n=3)
      }
      if (nrow(dmM.g503y)%%3==1){
        p3cony[1:3] <- tail(dmM.g503y$Var2, n=4)[1:3]
        p3absy[1:3] <- tail(dmM.g503y$value, n=4)[1:3]
      }
      if (nrow(dmM.g503y)%%3==2){
        p3cony[1:3] <- tail(dmM.g503y$Var2, n=5)[1:3]
        p3absy[1:3] <- tail(dmM.g503y$value, n=5)[1:3]
      }
      dmM.l503y <- dmM3y[nv3y==FALSE,]
      if (nrow(dmM.l503y)%%3==0){
        p3cony[4:6] <- head(dmM.l503y$Var2, n=3)
        p3absy[4:6] <- head(dmM.l503y$value, n=3)
      }
      if (nrow(dmM.l503y)%%3==1){
        p3cony[4:6] <- head(dmM.l503y$Var2, n=4)[2:4]
        p3absy[4:6] <- head(dmM.l503y$value, n=4)[2:4]
      }
      if (nrow(dmM.l503y)%%3==2){
        p3cony[4:6] <- head(dmM.l503y$Var2, n=5)[3:5]
        p3absy[4:6] <- head(dmM.l503y$value, n=5)[3:5]
      }
    }
  }

  #linear regression calculation for C50
  p1cony <- na.omit(p1cony)
  p1absy <- na.omit(p1absy)
  p1con.count1.1y <- sum(p1cony==p1cony[1])
  p1con.count1.2y <- sum(p1cony==p1cony[4])
  p1county <- p1con.count1.1y+p1con.count1.2y
  p1regy <- lm(p1absy~p1cony)
  p1datay <- summary(p1regy)
  p1xavy <- mean(p1cony)
  p1yavy <- mean(p1absy)
  p1slopey <- as.numeric(p1regy$coefficients[2])
  p1sigmay <- as.numeric(p1datay$sigma)
  p1ty <- qt(0.975, p1county)
  sqrty <- (((50-p1yavy)^2)/((p1con.count1.1y*(p1cony[1]-p1xavy)^2)+(p1con.count1.2y*(p1con
y[4]-p1xavy)^2)))+((p1slopey^2)/p1county)-(((p1ty^2)*(p1sigmay^2))/(p1county*((p1con.count1.
```

```
1y*(p1cony[1]-p1xavy)^2)+(p1con.count1.2y*(p1cony[4]-p1xavy)^2))))
   p1uppery <- p1xavy + ((p1slopey*(50-p1yavy))+(p1ty*p1sigmay*(sqrty)^(1/2)))/((p1slopey^2)-
((p1ty^2)*(p1sigmay^2)/p1county))
   p1lowery <- p1xavy + ((p1slopey*(50-p1yavy))-(p1ty*p1sigmay*(sqrty)^(1/2)))/((p1slopey^2)-((
p1ty^2)*(p1sigmay^2)/p1county))
   p1c50y <- (p1uppery-p1lowery)/2+p1lowery
   p1c50errory <- p1c50y-p1lowery
   if(is.numeric(p1c50errory)!=TRUE){
     p1c50y<-as.numeric(c50_1.1y)
     p1c50errory<-0
   }
   c50stdmatrixy <- p1c50errory
   c50valmatrixy <- p1c50y

   if(ncol(avvhhy)>=2){
     p2cony <- na.omit(p2cony)
     p2absy <- na.omit(p2absy)
     p2con.count1.1y <- sum(p2cony==p2cony[1])
     p2con.count1.2y <- sum(p2cony==p2cony[4])
     p2county <- p1con.count1.1y+p1con.count1.2y
     p2regy <- lm(p2absy~p2cony)
     p2datay <- summary(p2regy)
     p2xavy <- mean(p2cony)
     p2yavy <- mean(p2absy)
     p2slopey <- as.numeric(p2regy$coefficients[2])
     p2sigmay <- as.numeric(p2datay$sigma)
     p2ty <- qt(0.975, p2county)
     sqrty <- (((50-p2yavy)^2)/((p2con.count1.1y*(p2cony[1]-p2xavy)^2)+(p2con.count1.2y*(p2co
ny[4]-p2xavy)^2)))+((p2slopey^2)/p2county)-(((p2ty^2)*(p2sigmay^2))/(p2county*((p2con.count1
.1y*(p2cony[1]-p2xavy)^2)+(p2con.count1.2y*(p2cony[4]-p2xavy)^2))))
     p2uppery <- p2xavy + ((p2slopey*(50-p2yavy))+(p2ty*p2sigmay*(sqrty)^(1/2)))/((p2slopey^2)
-((p2ty^2)*(p2sigmay^2)/p2county))
     p2lowery <- p2xavy + ((p2slopey*(50-p2yavy))-(p2ty*p2sigmay*(sqrty)^(1/2)))/((p2slopey^2)-
((p2ty^2)*(p2sigmay^2)/p2county))
     p2c50y <- (p2uppery-p2lowery)/2+p2lowery
     p2c50errory <- p2c50y-p2lowery
     if(is.numeric(p2c50errory)!=TRUE){
       p2c50y<-as.numeric(c50_2.1y)
       p2c50errory<-0
     }
     c50stdmatrixy <- cbind(c50stdmatrixy, p2c50errory)
     c50valmatrixy <- cbind(c50valmatrixy, p2c50y)
   }
   if(ncol(avvhhy)>=3){
     p3cony <- na.omit(p3cony)
     p3absy <- na.omit(p3absy)
     p3con.count1.1y <- sum(p3cony==p3cony[1])
     p3con.count1.2y <- sum(p3cony==p3cony[4])
     p3county <- p3con.count1.1y+p3con.count1.2y
     p3regy <- lm(p3absy~p3cony)
     p3datay <- summary(p3regy)
```

```
    p3xavy <- mean(p3cony)
    p3yavy <- mean(p3absy)
    p3slopey <- as.numeric(p3regy$coefficients[2])
    p3sigmay <- as.numeric(p3datay$sigma)
    p3ty <- qt(0.975, p3county)
    sqrty <- (((50-p3yavy)^2)/((p3con.count1.1y*(p3cony[1]-p3xavy)^2)+(p3con.count1.2y*(p3co
ny[4]-p3xavy)^2)))+((p3slopey^2)/p3county)-(((p3ty^2)*(p3sigmay^2))/(p3county*((p3con.count1
.1y*(p3cony[1]-p3xavy)^2)+(p3con.count1.2y*(p3cony[4]-p3xavy)^2))))
    p3uppery <- p3xavy + ((p3slopey*(50-p3yavy))+(p3ty*p3sigmay*(sqrty)^(1/2)))/((p3slopey^2)
-((p3ty^2)*(p3sigmay^2)/p3county))
    p3lowery <- p3xavy + ((p3slopey*(50-p3yavy))-(p3ty*p3sigmay*(sqrty)^(1/2)))/((p3slopey^2)-
((p3ty^2)*(p3sigmay^2)/p3county))
    p3c50y <- (p3uppery-p3lowery)/2+p3lowery
    p3c50errory <- p3c50y-p3lowery
    if(is.numeric(p3c50errory)!=TRUE){
      p3c50y<-as.numeric(c50_3.1y)
      p3c50errory<-0
    }
    c50stdmatrixy <- cbind(c50stdmatrixy, p3c50errory)
    c50valmatrixy <- cbind(c50valmatrixy, p3c50y)
  }
  c50matrixy <- rbind(c50valmatrixy, c50stdmatrixy)
  c50dfy <- as.data.frame(t(c50matrixy))
  for (i in 1:ncol(avvhhy)){
    if (i==1){
      rownames(c50dfy)[1] <- c(p1y)
    }
    if (i==2){
      rownames(c50dfy)[1] <- c(p1y)
      rownames(c50dfy)[2] <- c(p2y)
    }
    if (i==3){
      rownames(c50dfy)[1] <- c(p1y)
      rownames(c50dfy)[2] <- c(p2y)
      rownames(c50dfy)[3] <- c(p3y)
    }
  }
  colnames(c50dfy) <- c("C50", "error")
  c50dfy <- t(c50dfy)
}

 z <- readline("What is the third raw data file name? (include csv extension) (Type 'NA' if no file)
")
if(z!="NA"){
  vhhz <- read.csv(file=z, header=FALSE, sep=",", dec = ".")
  vhhz <- vhhz[4:11, 3:14]
  p1z <- readline("What is the first protein?")
  p2z <- readline("What is the second protein? (Type 'NA' if no second protein)")
  p3z <- readline("What is the third protein? (Type 'NA' if no third protein)")
  pb1z <- paste("Blank_",p1z)
  pb2z <- paste("Blank_",p2z)
```

```
pb3z <- paste("Blank_",p3z)

#See concentration assignment below to assign 1:4 or 5:8 or 9:12 with min value as BLANK
for (i in 1:3){
  if (i==1){
    minbs1z <- min(as.numeric(vhhz[1,1:4]))
    bs1locz <- which(vhhz==minbs1z, arr.ind=TRUE)
    columnbs1z <- as.numeric(bs1locz[2])
    colnames(vhhz)[1:4] <- p1z
    colnames(vhhz)[columnbs1z] <- pb1z
  }
  if (i==2){
    minbs2z <- min(as.numeric(vhhz[1,5:8]))
    bs2locz <- which(vhhz==minbs2z, arr.ind=TRUE)
    columnbs2z <- as.numeric(bs2locz[2])
    colnames(vhhz)[5:8] <- p2z
    colnames(vhhz)[columnbs2z] <- pb2z
  }
  if (i==3){
    minbs3z <- min(as.numeric(vhhz[1,9:12]))
    bs3locz <- which(vhhz==minbs3z, arr.ind=TRUE)
    columnbs3z <- as.numeric(bs3locz[2])
    colnames(vhhz)[9:12] <- p3z
    colnames(vhhz)[columnbs3z] <- pb3z
  }
}

#find average value of blanks
blanksz <- vector(mode="numeric", length = 3)
blankz <- vhhz[grepl(pattern = "\\Blank.*", x=colnames(vhhz))==TRUE]
blanksz[1] <- mean(as.numeric(blankz[,1]))
blanksz[2] <- mean(as.numeric(blankz[,2]))
blanksz[3] <- mean(as.numeric(blankz[,3]))

#blank correct absorbance values
vhhcorrz <- vhhz[grepl(pattern = "\\Blank.*", x=colnames(vhhz))==FALSE]
for(i in 1:9){
  if(i<=3){
    vhhcorrz[,i] <- as.numeric(vhhcorrz[,i])-blanksz[1]
  }
  if(i>3&i<=6){
    vhhcorrz[,i] <- as.numeric(vhhcorrz[,i])-blanksz[2]
  }
  if(i>6&i<=9){
    vhhcorrz[,i] <- as.numeric(vhhcorrz[,i])-blanksz[3]
  }
}
vhhcorrz[is.na(vhhcorrz)] <- 0

#normalize absorbance values
normvhhz <- vhhcorrz
```

```
for(i in 1:9){
  maxabz <- max(normvhhz[,i])
  normvhhz[,i] <- normvhhz[,i]/maxabz*100
}
normvhhz <- normvhhz[grepl(pattern = "\\.*[n|N][a|A].*", x=colnames(normvhhz))==FALSE]
normvhhz[normvhhz==0] <- NA

#average absorbance values
avvhhz <- as.data.frame(matrix(nrow=3, ncol=8))
proteinnamesz <- colnames(normvhhz)
if (ncol(normvhhz)==3){
  avvhhz <- as.data.frame(matrix(nrow=1, ncol=8))
  rownames(avvhhz)[1] <- c(proteinnamesz[1])
}
if(ncol(normvhhz)==6){
  avvhhz <- as.data.frame(matrix(nrow=2, ncol=8))
  rownames(avvhhz)[1] <- c(proteinnamesz[1])
  rownames(avvhhz)[2] <- c(proteinnamesz[4])
}
if(ncol(normvhhz)==9){
  rownames(avvhhz)[1] <- c(proteinnamesz[1])
  rownames(avvhhz)[2] <- c(proteinnamesz[4])
  rownames(avvhhz)[3] <- c(proteinnamesz[7])
}
for (i in 1:nrow(avvhhz)){
  if (i==1){
    normvhh1z <- t(normvhhz[,1:3])
    for(r in 1:8){
      avvhhz[1, r] <- mean(normvhh1z[,r], na.rm = TRUE)
    }
  }
  if (i==2){
    normvhh2z <- t(normvhhz[,4:6])
    for(r in 1:8){
      avvhhz[2, r] <- mean(normvhh2z[,r], na.rm = TRUE)
    }
  }
  if (i==3){
    normvhh3z <- t(normvhhz[,7:9])
    for(r in 1:8){
      avvhhz[3, r] <- mean(normvhh3z[,r], na.rm = TRUE)
    }
  }
}
avvhhz <- t(avvhhz)
rownames(avvhhz) <- c(1, .5, .25, .125, .0625, .03125, .015625, .0078125)

#extracting values above and below 50%
colnames(normvhh1z) <- c(1, .5, .25, .125, .0625, .03125, .015625, .0078125)
p1absz <- c()
p1conz <- c()
```

```
columnnumber1z <- 0
for (i in 1:ncol(normvhh1z)){
  fiftyISHz <- grepl(pattern = "[5]\\d\\D", x=normvhh1z[,i])
  fiftylocz <- which(fiftyISHz, arr.ind = TRUE)
  if(length(fiftylocz)>0){
    columnnumber1z <- i
  }
}
if (columnnumber1z >0 ){
  av50z <- mean(normvhh1z[,columnnumber1z])
  std50z <- sd(normvhh1z[,columnnumber1z])
  tval50z <- -tinv(0.025, 2)
  conf.int.50z <- tval50z * std50z / sqrt(2)
  upper50.1z <- av50z + conf.int.50z
  lower50.1z <- av50z - conf.int.50z
  if (upper50.1z>50 && lower50.1z<50){
    c50_1.1z <- colnames(normvhh1z)[columnnumber1z]
    p1conz[1:6] <- c50_1.1z
    p1conz <- as.numeric(p1conz)
    p1absz[1:6] <- normvhh1z[,c50_1.1z]
    }
  else{
    break()
  }
}
if(columnnumber1z ==0) {
  dmM1z <- melt(normvhh1z, id.var=1)
  nv1z <- dmM1z$value>50
  dmM.g501z <- dmM1z[nv1z==TRUE,]
  if (nrow(dmM.g501z)%%3==0){
    p1conz[1:3] <- tail(dmM.g501z$Var2, n=3)
    p1absz[1:3] <- tail(dmM.g501z$value, n=3)
  }
  if (nrow(dmM.g501z)%%3==1){
    p1conz[1:3] <- tail(dmM.g501z$Var2, n=4)[1:3]
    p1absz[1:3] <- tail(dmM.g501z$value, n=4)[1:3]
  }
  if (nrow(dmM.g501z)%%3==2){
    p1conz[1:3] <- tail(dmM.g501z$Var2, n=5)[1:3]
    p1absz[1:3] <- tail(dmM.g501z$value, n=5)[1:3]
  }
  dmM.l501z <- dmM1z[nv1z==FALSE,]
  if (nrow(dmM.l501z)%%3==0){
    p1conz[4:6] <- head(dmM.l501z$Var2, n=3)
    p1absz[4:6] <- head(dmM.l501z$value, n=3)
  }
  if (nrow(dmM.l501z)%%3==1){
    p1conz[4:6] <- head(dmM.l501z$Var2, n=4)[2:4]
    p1absz[4:6] <- head(dmM.l501z$value, n=4)[2:4]
  }
  if (nrow(dmM.l501z)%%3==2){
```

```
    p1conz[4:6] <- head(dmM.l501z$Var2, n=5)[3:5]
    p1absz[4:6] <- head(dmM.l501z$value, n=5)[3:5]
  }
}

if(ncol(avvhhz)>=2){
  colnames(normvhh2z) <- c(1, .5, .25, .125, .0625, .03125, .015625, .0078125)
  p2absz <- c()
  p2conz <- c()
  columnnumber2z <- 0
  for (i in 1:ncol(normvhh2z)){
    fiftyISHz <- grepl(pattern = "[5]\\d\\D", x=normvhh2z[,i])
    fiftylocz <- which(fiftyISHz, arr.ind = TRUE)
    if(length(fiftylocz)>0){
      columnnumber2z <- i
    }
  }
  if (columnnumber2z >0 ){
    av50z <- mean(normvhh2z[,columnnumber2z])
    std50z <- sd(normvhh2z[,columnnumber2z])
    tval50z <- -tinv(0.025, 2)
    conf.int.50z <- tval50z * std50z / sqrt(2)
    upper50.2z <- av50z + conf.int.50z
    lower50.2z <- av50z - conf.int.50z
    if (upper50.2z>50 && lower50.2z<50){
      c50_2.1z <- colnames(normvhh2z)[columnnumber2z]
      p2conz[1:6] <- c50_2.1z
      p2conz <- as.numeric(p2conz)
      p2absz[1:6] <- normvhh2z[,c50_2.1z]
    }
    else{
      break()
    }

  }
  else{
    dmM2z <- melt(normvhh2z, id.var=1)
    nv2z <- dmM2z$value>50
    dmM.g502z <- dmM2z[nv2z==TRUE,]
    if (nrow(dmM.g502z)%%3==0){
      p2conz[1:3] <- tail(dmM.g502z$Var2, n=3)
      p2absz[1:3] <- tail(dmM.g502z$value, n=3)
    }
    if (nrow(dmM.g502z)%%3==1){
      p2conz[1:3] <- tail(dmM.g502z$Var2, n=4)[1:3]
      p2absz[1:3] <- tail(dmM.g502z$value, n=4)[1:3]
    }
    if (nrow(dmM.g502z)%%3==2){
      p2conz[1:3] <- tail(dmM.g502z$Var2, n=5)[1:3]
      p2absz[1:3] <- tail(dmM.g502z$value, n=5)[1:3]
    }
```

```
    dmM.l502z <- dmM2z[nv2z==FALSE,]
    if (nrow(dmM.l502z)%%3==0){
      p2conz[4:6] <- head(dmM.l502z$Var2, n=3)
      p2absz[4:6] <- head(dmM.l502z$value, n=3)
    }
    if (nrow(dmM.l502z)%%3==1){
      p2conz[4:6] <- head(dmM.l502z$Var2, n=4)[2:4]
      p2absz[4:6] <- head(dmM.l502z$value, n=4)[2:4]
    }
    if (nrow(dmM.l502z)%%3==2){
      p2conz[4:6] <- head(dmM.l502z$Var2, n=5)[3:5]
      p2absz[4:6] <- head(dmM.l502z$value, n=5)[3:5]
    }
  }
}
if(ncol(avvhhz)>=3){
  colnames(normvhh3z) <- c(1, .5, .25, .125, .0625, .03125, .015625, .0078125)
  p3absz <- c()
  p3conz <- c()
  columnnumber3z <- 0
  for (i in 1:ncol(normvhh3z)){
    fiftyISHz <- grepl(pattern = "[5]\\d\\D", x=normvhh3z[,i])
    fiftylocz <- which(fiftyISHz, arr.ind = TRUE)
    if(length(fiftylocz)>0){
      columnnumber3z <- i
    }
  }
  if (columnnumber3z >0 ){
    av50z <- mean(normvhh3z[,columnnumber3z])
    std50z <- sd(normvhh3z[,columnnumber3z])
    tval50z <- -tinv(0.025, 2)
    conf.int.50z <- tval50z * std50z / sqrt(2)
    upper50.3z <- av50z + conf.int.50z
    lower50.3z <- av50z - conf.int.50z
    if (upper50.3z>50 && lower50.3z<50){
      c50_3.1z <- colnames(normvhh3z)[columnnumber3z]
      p3conz[1:6] <- c50_3.1z
      p3conz <- as.numeric(p3conz)
      p3absz[1:6] <- normvhh3z[,c50_3.1z]
    }
    else{
      break()
    }
  }
  else{
    dmM3z <- melt(normvhh3z, id.var=1)
    nv3z <- dmM3z$value>50
    dmM.g503z <- dmM3z[nv3z==TRUE,]
    if (nrow(dmM.g503z)%%3==0){
      p3conz[1:3] <- tail(dmM.g503z$Var2, n=3)
      p3absz[1:3] <- tail(dmM.g503z$value, n=3)
```

```
      }
      if (nrow(dmM.g503z)%%3==1){
        p3conz[1:3] <- tail(dmM.g503z$Var2, n=4)[1:3]
        p3absz[1:3] <- tail(dmM.g503z$value, n=4)[1:3]
      }
      if (nrow(dmM.g503z)%%3==2){
        p3conz[1:3] <- tail(dmM.g503z$Var2, n=5)[1:3]
        p3absz[1:3] <- tail(dmM.g503z$value, n=5)[1:3]
      }
      dmM.l503z <- dmM3z[nv3z==FALSE,]
      if (nrow(dmM.l503z)%%3==0){
        p3conz[4:6] <- head(dmM.l503z$Var2, n=3)
        p3absz[4:6] <- head(dmM.l503z$value, n=3)
      }
      if (nrow(dmM.l503z)%%3==1){
        p3conz[4:6] <- head(dmM.l503z$Var2, n=4)[2:4]
        p3absz[4:6] <- head(dmM.l503z$value, n=4)[2:4]
      }
      if (nrow(dmM.l503z)%%3==2){
        p3conz[4:6] <- head(dmM.l503z$Var2, n=5)[3:5]
        p3absz[4:6] <- head(dmM.l503z$value, n=5)[3:5]
      }
    }
  }

  #linear regression calculation for C50
  p1conz <- na.omit(p1conz)
  p1absz <- na.omit(p1absz)
  p1con.count1.1z <- sum(p1conz==p1conz[1])
  p1con.count1.2z <- sum(p1conz==p1conz[4])
  p1countz <- p1con.count1.1z+p1con.count1.2z
  p1regz <- lm(p1absz~p1conz)
  p1dataz <- summary(p1regz)
  p1xavz <- mean(p1conz)
  p1yavz <- mean(p1absz)
  p1slopez <- as.numeric(p1regz$coefficients[2])
  p1sigmaz <- as.numeric(p1dataz$sigma)
  p1tz <- qt(0.975, p1countz)
  sqrtz <- (((50-p1yavz)^2)/((p1con.count1.1z*(p1conz[1]-p1xavz)^2)+(p1con.count1.2z*(p1con
z[4]-p1xavz)^2)))+((p1slopez^2)/p1countz)-(((p1tz^2)*(p1sigmaz^2))/(p1countz*((p1con.count1.
1z*(p1conz[1]-p1xavz)^2)+(p1con.count1.2z*(p1conz[4]-p1xavz)^2))))
  p1upperz <- p1xavz + ((p1slopez*(50-p1yavz))+(p1tz*p1sigmaz*(sqrtz)^(1/2)))/((p1slopez^2)-
((p1tz^2)*(p1sigmaz^2)/p1countz))
  p1lowerz <- p1xavz + ((p1slopez*(50-p1yavz))-(p1tz*p1sigmaz*(sqrtz)^(1/2)))/((p1slopez^2)-((
p1tz^2)*(p1sigmaz^2)/p1countz))
  p1c50z <- (p1upperz-p1lowerz)/2+p1lowerz
  p1c50errorz <- p1c50z-p1lowerz
  if(is.na(p1c50errorz)==TRUE){
    p1c50z<-as.numeric(c50_1.1z)
    p1c50errorz<-0
  }
```

```
  c50stdmatrixz <- p1c50errorz
  c50valmatrixz <- p1c50z

  if(ncol(avvhhz)>=2){
    p2conz <- na.omit(p2conz)
    p2absz <- na.omit(p2absz)
    p2con.count1.1z <- sum(p2conz==p2conz[1])
    p2con.count1.2z <- sum(p2conz==p2conz[4])
    p2countz <- p2con.count1.1z+p2con.count1.2z
    p2regz <- lm(p2absz~p2conz)
    p2dataz <- summary(p2regz)
    p2xavz <- mean(p2conz)
    p2yavz <- mean(p2absz)
    p2slopez <- as.numeric(p2regz$coefficients[2])
    p2sigmaz <- as.numeric(p2dataz$sigma)
    p2tz <- qt(0.975, p2countz)
    sqrtz <- (((50-p2yavz)^2)/((p2con.count1.1z*(p2conz[1]-p2xavz)^2)+(p2con.count1.2z*(p2co
nz[4]-p2xavz)^2)))+((p2slopez^2)/p2countz)-(((p2tz^2)*(p2sigmaz^2))/(p2countz*((p2con.count1
.1z*(p2conz[1]-p2xavz)^2)+(p2con.count1.2z*(p2conz[4]-p2xavz)^2))))
    p2upperz <- p2xavz + ((p2slopez*(50-p2yavz))+(p2tz*p2sigmaz*(sqrtz)^(1/2)))/((p2slopez^2)
-((p2tz^2)*(p2sigmaz^2)/p2countz))
    p2lowerz <- p2xavz + ((p2slopez*(50-p2yavz))-(p2tz*p2sigmaz*(sqrtz)^(1/2)))/((p2slopez^2)-
((p2tz^2)*(p2sigmaz^2)/p2countz))
    p2c50z <- (p2upperz-p2lowerz)/2+p2lowerz
    p2c50errorz <- p2c50z-p2lowerz
    if(is.na(p2c50errorz)==TRUE){
      p2c50z<-as.numeric(c50_2.1z)
      p2c50errorz<-0
    }
    c50stdmatrixz <- cbind(c50stdmatrixz, p2c50errorz)
    c50valmatrixz <- cbind(c50valmatrixz, p2c50z)
  }
  if(ncol(avvhhz)>=3){
    p3conz <- na.omit(p3conz)
    p3absz <- na.omit(p3absz)
    p3con.count1.1z <- sum(p3conz==p3conz[1])
    p3con.count1.2z <- sum(p3conz==p3conz[4])
    p3countz <- p3con.count1.1z+p3con.count1.2z
    p3regz <- lm(p3absz~p3conz)
    p3dataz <- summary(p3regz)
    p3xavz <- mean(p3conz)
    p3yavz <- mean(p3absz)
    p3slopez <- as.numeric(p3regz$coefficients[2])
    p3sigmaz <- as.numeric(p3dataz$sigma)
    p3tz <- -tinv(.025, 6)
    sqrtz <- (((50-p3yavz)^2)/((p3con.count1.1z*(p3conz[1]-p3xavz)^2)+(p3con.count1.2z*(p3co
nz[4]-p3xavz)^2)))+((p3slopez^2)/p3countz)-(((p3tz^2)*(p3sigmaz^2))/(p3countz*((p3con.count1
.1z*(p3conz[1]-p3xavz)^2)+(p3con.count1.2z*(p3conz[4]-p3xavz)^2))))
    p3upperz <- p3xavz + ((p3slopez*(50-p3yavz))+(p3tz*p3sigmaz*(sqrtz)^(1/2)))/((p3slopez^2)
-((p3tz^2)*(p3sigmaz^2)/p3countz))
    p3lowerz <- p3xavz + ((p3slopez*(50-p3yavz))-(p3tz*p3sigmaz*(sqrtz)^(1/2)))/((p3slopez^2)-
```

```
((p3tz^2)*(p3sigmaz^2)/p3countz))
    p3c50z <- (p3upperz-p3lowerz)/2+p3lowerz
    p3c50errorz <- p3c50z-p3lowerz
    if(is.na(p3c50errorz)==TRUE){
      p3c50z<-as.numeric(c50_3.1z)
      p3c50errorz<-0
    }

    c50stdmatrixz <- cbind(c50stdmatrixz, p3c50errorz)
    c50valmatrixz <- cbind(c50valmatrixz, p3c50z)
  }
  c50matrixz <- rbind(c50valmatrixz, c50stdmatrixz)
  c50dfz <- as.data.frame(t(c50matrixz))
  for (i in 1:ncol(avvhhz)){
    if (i==1){
      rownames(c50dfz)[1] <- c(p1z)
    }
    if (i==2){
      rownames(c50dfz)[1] <- c(p1z)
      rownames(c50dfz)[2] <- c(p2z)
    }
    if (i==3){
      rownames(c50dfz)[1] <- c(p1z)
      rownames(c50dfz)[2] <- c(p2z)
      rownames(c50dfz)[3] <- c(p3z)
    }
  }
  colnames(c50dfz) <- c("C50", "error")
  c50dfz <- t(c50dfz)
}

#Create Excel Files for ELISA and C50 Worked Up Data
if(y=="NA" & z=="NA"){
  avvhh <- avvhh
  c50df <- c50df
}
if(y!="NA" & z=="NA"){
  avvhh <- cbind(avvhh, avvhhy)
  c50df <- cbind(c50df, c50dfy)
}
if(y!="NA" & z!="NA"){
  avvhh <- cbind(avvhh, avvhhy, avvhhz)
  c50df <- cbind(c50df, c50dfy, c50dfz)
}
write.csv(avvhh, file=paste(Sys.Date(), a, "ELISA File.csv"))
write.csv(c50df, file=paste(Sys.Date(), a, "C50 File.csv"))

#Work Up C50 Excel Data
myfiles <- list.files()
c50file <- grepl(pattern = "\\s?[C][5][0]\\s?[F][i][l][e].[c][s][v]", myfiles)
allc50 <- read.csv(file=myfiles[c50file==TRUE], header=FALSE, sep=",", dec=".")
```

```
colnames(allc50) <- allc50[1,]
rownames(allc50) <- allc50[,1]
allc50 <- allc50[-1,-1]
wtc50val <- allc50[grepl(pattern = "\\.*[w|W].*[t|T].*", x=colnames(allc50))==TRUE]
wtc50 <- as.numeric(wtc50val[1,])
wterror <- as.numeric(wtc50val[2,])
allc50 <- as.data.frame(t(allc50))
allc50$error <- as.numeric(allc50$error)
allc50$C50<- as.numeric(allc50$C50)
allc50$error <- (((allc50$error/allc50$C50)^2)+((wterror/wtc50)^2))^(1/2)
allc50$C50<- allc50$C50/wtc50*100
allc50$error <- allc50$error*allc50$C50
allc50 <- allc50[order(rownames(allc50), decreasing = FALSE),]
write.csv(allc50, file = paste(Sys.Date(), a, "Normalized C50 File.csv"))

#Create C50 Bar Plots
ggplot(allc50)+
  geom_col(aes(x=rownames(allc50), y=C50, fill=rownames(allc50)))+
  geom_errorbar(aes(x=rownames(allc50), ymin=C50-error, ymax=C50+error), na.rm=TRUE,
width=0.5,
              size=1.3)+
  xlab("F5 Proteins")+ggtitle("C50 Values Normalized to F5 Wild Type")+ylab("Normalized C50"
)+
  theme(legend.title = element_blank(), plot.title = element_text(hjust = 0.5, size=(25)),
        rect=element_rect(fill = "transparent"), axis.text=element_text(size = 14),
        axis.title = element_text(size=18), legend.text = element_text(size=12))
ggsave(paste(Sys.Date(), a, "C50 PowerPoint Bar Plot.png"), plot=last_plot(),
       device = "png", width=9.61, height=6.86, units="in", bg="transparent")

ggplot(allc50)+
  geom_col(aes(x=rownames(allc50), y=C50, fill=rownames(allc50)))+
  geom_errorbar(aes(x=rownames(allc50), ymin=C50-error, ymax=C50+error), width=0.5,
              size=.5)+
  xlab("F5 Proteins")+ggtitle("C50 Values Normalized to F5 Wild Type")+ylab("Normalized C50"
)+
  theme(legend.title = element_blank(), plot.title = element_text(hjust = 0.5, size=(8)),
        rect=element_rect(fill = "transparent"), axis.text=element_text(size = 4),
        axis.title = element_text(size=6), legend.text = element_text(size=4),
        legend.key.size = unit(.25, 'cm'))
ggsave(paste(Sys.Date(), a, "C50 Journal Bar Plot.png"), plot=last_plot(),
       device = "png", width=3, height=2.14, units="in", bg="transparent")

#Work Up ELISA Excel Data
myfiles <- list.files()
ELISAfile <- grepl(pattern = "\\s?[E][L][I][S][A]\\s?[F][i][l][e].[c][s][v]", myfiles)
allELISA <- read.csv(file=myfiles[ELISAfile==TRUE], header=FALSE, sep=",", dec=".")
colnames(allELISA) <- allELISA[1,]
allELISA <- allELISA[-1,]
colnames(allELISA)[1] <- "aa"
allELISA <- allELISA[,order(colnames(allELISA))]
allELISA <- allELISA[order(allELISA$aa),]
```

```
 for (i in 1:ncol(allELISA)){
  allELISA[[i]] <- as.numeric(allELISA[[i]])
 }

 #Create ELISA Plots
 dm <- melt(allELISA, id.var=1)
 ggplot(data=dm, aes(x=log10(aa), value, colour=variable))+
  geom_point()+labs(x="F5 Protein Concentration (ug/mL)", y="Normalized Absorbance at 450
nm",
                    title="Relative Binding Affinity of F5 Proteins to RTA")+
  scale_color_discrete(name="F5 Mutants")+
  theme(plot.title = element_text(hjust=0.5, size=(25)), rect=element_rect(fill = "transparent"),
        axis.text=element_text(size = 14), axis.title = element_text(size=18),
        legend.text = element_text(size=12), legend.title = element_text(size=14))+
  geom_smooth(method="loess", se=F)+ scale_x_reverse(breaks=c(0, -1, -2), labels=c("1", "0.
1", "0.01"))
 ggsave(paste(Sys.Date(),a,"ELISA PowerPoint Plot.png"), plot=last_plot(),
       device = "png", width=9.61, height=6.86, units="in", bg="transparent")

 ggplot(data=dm, aes(x=log10(aa), value, colour=variable))+
  geom_point(size=.5)+labs(x="F5 Protein Concentration (ug/mL)", y="Normalized Absorbance
at 450 nm",
                           title="Relative Binding Affinity of F5 Proteins to RTA")+
  scale_color_discrete(name="F5 Mutants")+
  theme(plot.title = element_text(hjust=0.5, size=(8)), rect=element_rect(fill = "transparent"),
        axis.text = element_text(size=4), axis.title=element_text(size=6),
        legend.title = element_text(size=6), legend.text = element_text(size = 4),
        legend.key.size = unit(.25, 'cm'))+
  geom_smooth(method="loess", se=F, size=.5)+ scale_x_reverse(breaks=c(0, -1, -2), labels=
c("1", "0.1", "0.01"))
 ggsave(paste(Sys.Date(),a,"ELISA Journal Plot.png"), plot=last_plot(),
       device = "png", width=3, height=2.14, units="in", bg="transparent")
}

vhh_work_up <- vhhread()
```