In this project, I set out to carry out one of the pre-approved ideas listed for this project, that being to write a python script that analyzes a bacterial genome. Specifically, it scanned the genome for open reading frames (ORFs) that started with a start codon and ended with a stop codon. It then took these ORFs and translated them into the corresponding protein sequences, estimated their molecular weights, and ran five of the proteins through protein BLAST on NCBI.

The approach I took relied largely on functions in existing packages but with some extra flair so as to put all the applications together. Before any of these, I wrote a short function to read in the sequence of DNA to be analyzed from a FASTA file (via the 'pyfaidx' package); I was then able to conduct analysis. Firstly, for scanning the genome for ORFs, I first tried regular expressions (in the 're' package) that scanned both DNA strands for TAC, then any number of alphanumerics, then either ATT, ACT, or ATC in the 3' to 5' direction. This would correspond to AUG (start codon) and then UAA, UGA, or UAG (stop codon) in the 5' to 3' direction on the corresponding mRNA. While this approach worked for small data sequences, it had a myriad of problems: firstly, it was not able to account for overlapping ORFs. This was solved by moving the regular expression function call from the 're' package to the 'regex' package, whose functions take a parameter to check for overlapping. However, this led to a different problem in that overlapping regular expression matches, especially on large sequences, caused the python kernel to crash.

This led to an entirely different approach to find the ORFs. I turned to a smaller regular expression match, that being just the start codon TAC; this also ensures that no matches overlap, and that matches can be easily found. From there, I retrieved the positions of all matches and then tasked the program with scanning the input DNA sequence in groups of 3 bases starting from each start codon position; if the group is not a stop codon, the program would add it to the current reading frame, and if it did find a stop codon, it would still add but then stop building the current reading frame. This process would repeat until all start codons were associated with a reading frame, or if the finder goes to the end of the sequence without finding a frame, at which point it would stop searching and not add the frame at all.

Finding the ORFs was the difficult part to refine, but the following procedures were fairly easy. Using Biopython's 'Seq' module, I implemented translation and weight estimation easily, and using Biopython's 'Entrez' and 'NCBIWWW' modules, I was able to conduct the BLASTing for five of the most suitable sequences. This takes a long time to do with Python, and there was unfortunately nothing I could do about that.

To implement and test my script, I used a Jupyter notebook to construct the functions and test them on preliminary small-scale data. Once this was done, I incrementally put the smaller functions together (also in this notebook) before exporting the results as a .py file and implementing a feature to run the script from the command line and take various arguments. To get data for testing the script, I used a combination of data files given in class (including pGL3.fa, given in a homework assignment) and various bacterial genome FASTA files found on the NCBI database.

The result is the script given in this submission as 'final_project.py'. It can be run from the command line and takes three arguments, though two of them are optional. The first is the path to the FASTA file to analyze; this is not optional. The second is the mode of the results to return. I implemented modes to just get ORFs (set the second argument to 'orfs'), just get protein sequences ('prot'), just get molecular weights ('mw'), do all of the above but not BLAST ('noblast'), and to do all three and BLAST ('all'). The third argument is the email to use if BLASTing; if not given, the email is automatically set to my CMU email address, jazhu@andrew.cmu.edu.

When run, the script returns (depending on the selected mode) a FASTA file containing all open reading frames, a text file of all proteins that are more than one amino acid long, a text file of all (estimated) molecular masses of these proteins, and a .xml file with the results of the BLAST.

Finally, if I were to expand or improve upon this project, I would seek to do several refinements. Firstly, all the result ORFs are given in a single file without much elaboration as to whether they are on the forward or reverse strand of the input DNA, and if there are multiple sequences given in the FASTA file, the result file does not detail which ORF is on which sequence; though this could be figured out, implementing this refinement would make life much easier for the user. Secondly, I would like to extend this similar philosophy to elaborating on the resulting protein sequences and molecular weights, possibly even condensing the results into a single file. Finally, I would seek to improve my algorithm to find ORFs. While this algorithm works and is sufficient for a bacterial genome, it would take much longer for many eukaryotic genomes. As such, more advanced implementations could use data structures such as suffix trees to reduce the time needed to find all ORFs in a DNA sequence; since much work has been done in the area of suffix trees, it may also make storage of the string more efficient.