The University of Texas at San Antonio

Project I

Final Report

Sang Woo Choi - zzb279

Gabriela De La Torre -nmp273

Juan Banderas - zfl957

Intro to Comp Biology

Kamal Al Nasr

9 Nov 2019

A. Introduction

We were given the task “to apply concepts and techniques of data analysis to analyze some features of protein structures”. Each group member was given subtasks/questions to do and a shared task to work on the report and slides. Throughout this course, we learned different techniques to analyze proteins and our work is collaborated and stored on github(https://github.com/sangchoi93/bioinformatics_group_project_1). This report will explain the process and techniques that we took to accomplish our task.

B. Preprocessing PDB Data

Preprocessing PDB data is done in PDB_Parser class in pdb_parser.py. The class looks at dataset file cullpdb_pc30_res3 gzip file the instructor has provided and extracts the list of proteins to get PDB data for from rcsb.org. It uses requests module to individually download required PDB files and store them in ./pdb directory. The reason why it downloads the file is to expedite parsing the next run so there’s no need to reach out to the server unnecessarily, which is a time-consuming process dealing with approximately 18,000 proteins. The parser then opens each file, and it parses differently based on the first word of each line. In this case, it looks for keywords, ATOMS, HELIX, and SHEET and parses them differently based on the specification provided in format_spacing dictionary which tells parser the index of where each column starts. The parser has 3 data frames: df_atom, df_sheet, and df_helix so based on the keyword, it appends to the data frame. After, it drops rows that belong to the specified chain ID.

These three data frames are then manipulated so that they can easily be used to answer the provided questions.

C. Team member contributions

a. Sang Woo Choi

   i. Sang worked on the parser class that reads provided protein dataset, fetches the PDB file for each protein, and then parses them into data frames so they can be easily used to further analyze the protein datasets. Additionally, he built various utility methods like building a Ramachandran plot which was primarily used to answer questions 4 and 6

b. Gabriela De La Torre

   i. Gabriela worked on questions 5 and 7. She used the parser that Sang created and modified to analyze beta sheets for question 5. Sang's parser created a data frame that she saved to a CSV to be able to read from without having to run the code again. From the data frame, she was able to run counters for the initial, current, and end residues. For the second part of question 5 of another data frame was created to run a counter on the number of strands. That data frame is derived from the sheets data frame, with the removal of duplicates. She also used the parser for question 7 to create an atom data frame but, the number of atoms was so large the CSV only accounts from 12AS to 1DQP or a sample of 399705 lines.

The atom data frame was them cleaned to calculate the length from CA to CA atoms and saved to a list. For the analytics, she ran the max, min, median, and mean of the length results.

c. Juan Banderas

i. Juan worked on questions 1, 2, and 3. He used the parser that Sang created and modified it to analyze helices for all three questions. His parser created a data frame that he then saved into a CSV file for later analysis. He used the initial/end residues, helices lengths, and helix class portions of the data to answer the questions.

E. Questions

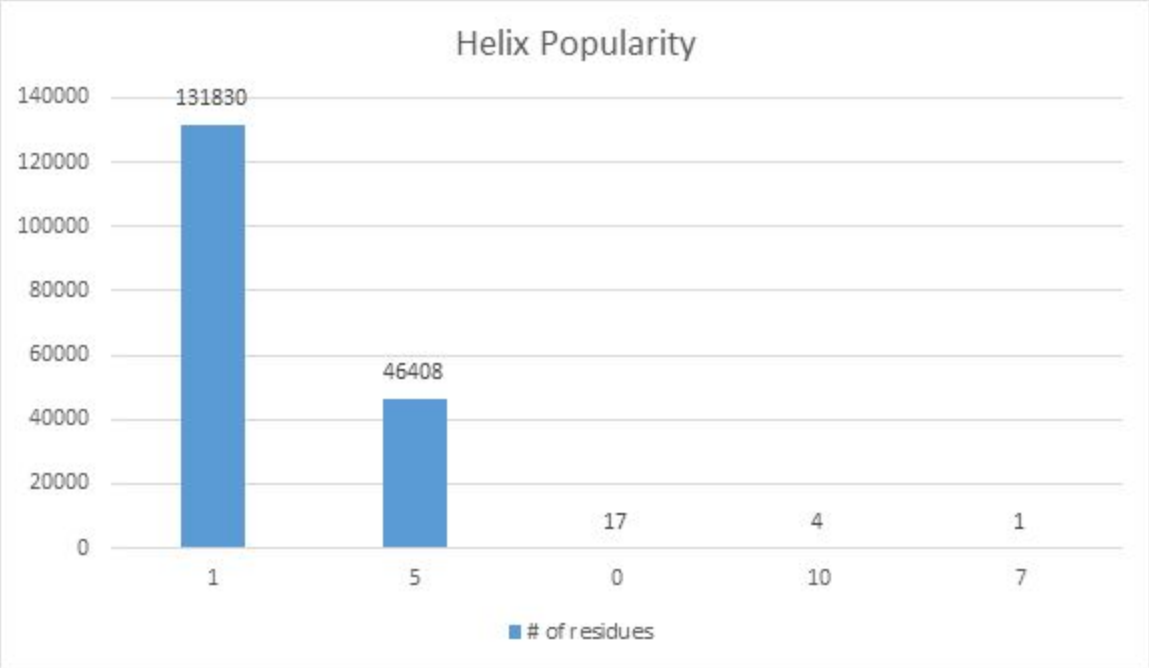
1. Find the popularity of different kinds of helices: read the data set and analyze it to find the popularity of each type of helices.

   Implementation:

   Since our code parses out the data from the file by column into their own separate data frames, we can use Python's Counter tool to see how many occurrences of an object exists within a column, in our case. And for this question, we get to see which helices exist in our data and how many of each type, therefore answering the question by figuring out the popularity of each type of helix in our data set.

Results:

Our results were predictable when it came to which helices were going to be the most popular, but not by this much of a gap. Alpha-helices and 3-10 helices dominated by a gap of approximately 150 residues. It was interesting to see that there were more than those kinds of helices in our data set since we've had a majority of our course work surround those two popular helices.



2. Analyze the amino acids that prefer to form/be part of helices. Does an AA prefer to form/be part of a specific type of helices?

Implementation:

As mentioned before, our data is parsed and ready to be observed through our code, so the Counter tool comes in handy once again. We get to see the most popular residues in the initial and end tails of the helices. One could even go as far as trying to see the “current residues” (residues between the middle and end), but for this assignment, we kept it brief on this question, due to overloading amounts of data that cause memory errors within our systems at the moment.

Results:

With a little research on residue popularity, one could say that our results seem to be in parallel with what many have found in their studies. Residues like Leucine, Alanine, Lysine, Glutamate, and Asparagine are found in the top half of the most popular residues in the helices dataframe. (Glycine is found the most in the ends of helices because it is considered a helix breaker, so usually when it is seen, the protein stops creation on such helix and returns to a coil or turns into another secondary structure.) We also couldn’t tell whether some residues preferred to be in certain types of helices because all helices shared similar residues across the board but at the same time, we had a small number of other helices in our data set relative to alpha and 3-10 helices. See the results below for residue popularity.

```
Initial: Counter({'ASP': 24552, 'SER': 23523, 'THR': 16535, 'ASN': 15336,
'GLY': 14993, 'PRO': 14450, 'LEU': 8328, 'GLU': 8144, 'ALA': 7717, 'LYS':
6938, 'ARG': 6333, 'HIS': 4894, 'GLN': 4598, 'VAL': 4282, 'PHE': 4150,
'TYR': 4105, 'ILE': 3534, 'CYS': 2082, 'MET': 1668, 'TRP': 1563, 'MSE':
```

466, 'MLY': 12, 'OCS': 7, 'UNK': 7, 'FME': 5, 'SEP': 4, 'LLP': 4, 'CSO': 4, 'CME': 3, 'CYG': 2, 'SEB': 2, 'CSX': 1, 'PTR': 1, 'MIR': 1, 'SEC': 1, 'AIB': 1, 'GM8': 1, 'CAS': 1, '2ZC': 1, 'DDZ': 1, 'SMC': 1, 'ALO': 1, 'OAS': 1, 'HYP': 1, 'KCX': 1, 'OSE': 1, 'LYR': 1, 'CSD': 1, 'CXM': 1, 'PCA': 1})

End Counter({'GLY': 23286, 'LEU': 17926, 'ALA': 15554, 'SER': 12323, 'ASN': 11344, 'LYS': 11096, 'GLU': 10074, 'ARG': 9940, 'ASP': 8678, 'GLN': 7981, 'PHE': 7904, 'VAL': 7246, 'THR': 7017, 'ILE': 6740, 'TYR': 6374, 'HIS': 5503, 'MET': 3380, 'CYS': 2707, 'TRP': 1922, 'MSE': 965, 'PRO': 239, 'MLY': 25, 'UNK': 9, 'CSO': 5, 'MLZ': 4, 'MHS': 3, 'CGU': 2, 'SEC': 1, 'DBZ': 1, 'CSX': 1, 'XPC': 1, 'CSD': 1, 'OCS': 1, 'SCH': 1, 'CME': 1, 'BAL': 1, 'SCY': 1, 'TPO': 1, 'LYR': 1, 'YCM': 1})

3. Analyze the length of helices (in terms of number of AAs). Is there any relation between the type of a helix and its length?

   Implementation:

   For this question, we had to create a subset in our helix data frame to see helix classes and their lengths. So that meant dropping duplicates from our data set so that we had a chance to see all of them and not just an overwhelming amount of the popular helices and/or not even see the least popular helices at all.

   Results:

helix_class	helix_id	init_chain_id	init_iCode	init_res_name	init_seq_num	length
5	6	A	NaN	ARG	143	55
5	3	A	NaN	ASP	454	24
7	3	A	NaN	LYS	57	7
10	H1	A	NaN	SER	697	9
10	L2	A	NaN	GLY	250	4
...	...	...	...	...	...	...
1	H04	A	NaN	ALA	109	21
1	H06	A	NaN	ASN	177	16
1	H07	A	NaN	ASP	206	13
1	H09	A	NaN	ASP	257	17
1	H10	A	NaN	ASN	293	9

We see a very slight correlation here. Alpha helices seem to usually be in lengths between 10 and 30, a bit reserved, while 3-10 helices seem to overshoot the prior lengths with their own range of lengths: 15 to 60, a little bit more widespread. As for the Polyproline and Omega helices, their lengths seem very minimal/small, compared to the previous two helices mentioned. We see lengths less than 10 residues here, but these types of helices can range from the lengths seen here to the ballpark of 15. See results above.

4. Analyze the torsion angles in each type of helices. Build something similar to Ramachandran plot but for different types of helices.

Implementation:

Utilities class contains various modules to be able to extract coordinates of each atom belonging to helices, calculate torsion angles of each amino acid, and plot

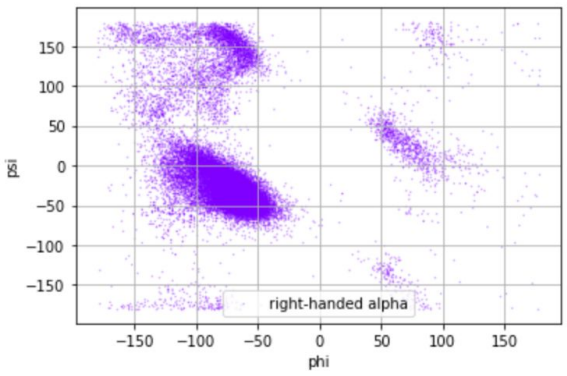
Ramachandran graph out of the data points. It was evident that getting coordinates of a particular atom out of a huge DataFrame was an extremely time-consuming process. In order to expedite the extraction, the team built a dictionary whose keys are concatenated strings of protein name, chain ID, residue sequence, and the name of the atom and whose values are x, y, and z coordinates of the atom.

The program first gets the helix dataframe, builds coordinate dictionary out of all atoms, and uses the dictionary to calculate psi and phi angles for each amino acid belonging to a helix. Then, the team used matplolib.scatter function to build scatter plots with phi as x axis and psi as y axis.

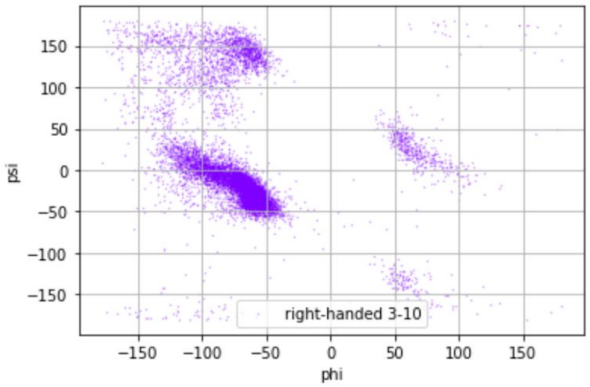
Results:

Torsion angles between right-handed omega and right-handed 3-10 helices showed virtually no difference between each other as you can see from the plots below:

```
In [7]: pdb_util.plot_ramachandran(df_ram[df_ram['helix_class'] == 1], 0.5, 0.1, 'helix_type')
```



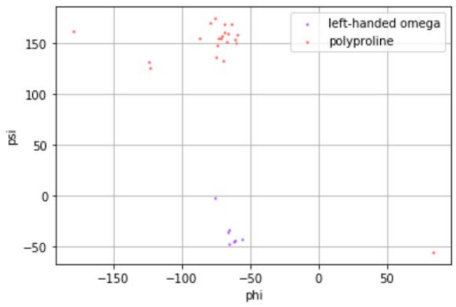
```
In [8]: pdb_util.plot_ramachandran(df_ram[df_ram['helix_class'] == 5], 0.5, 0.1, 'helix_type')
```



In the two plots above, it is clear that there are 4 clusters of dots at (-100, 0), (-75, 150), (50, 45), and (50, -130). Although they are technically different types of helix, they seem to show extremely similar torsion angle pattern as the are both right handed helix type.

As one can see from the distribution of helix classes in Question 1, there were not many other types of helices in the dataset to characterize them but the Ramachandran plot for rest of the helix types is shown below:

```
In [14]: pdb_util.plot_ramachandran(df_ram[df_ram['helix_class'].isin([2, 3, 4, 6, 7, 8, 9, 10])], 0.5, 2, 'helix_type')
```



With limited amount of data points, however, one can approximate a tiny cluster for left-handed helix class at around (-60, -45), which overlaps one of the clusters of

right-handed helix type. Polyproline shows a cluster at (-60, 150), which again overlaps with one of the right-handed clusters.

5. Analyze the amino acids that prefer to form/be part of beta-sheets. Analyze the number of strands in beta-sheets.

Implementation:

After using the preprocessor made by Sang, a data frame called df_sheet is created. Df_sheet is then passed into the Counter function to get a count of what residues are present currently, initially, and at the end. For the number of strands the dataframe df_sheet is cleaned by dropping duplicates by the subset of “protein_name” and “sheet_id”. Once cleaned the Counter function is used again to get a numbers for “num_strands”.

Results:

The results found align a bit with some research I did after to double check my work. “An interesting bit of information is that certain amino acids making up the polypeptide will actually prefer certain folding structures … For example, amino acids such as Valine, Isoleucine, and Threonine all have branching at the beta carbon, this will cause steric clashes in an alpha helix arrangement. Glycine is the smallest amino acid and can fit into all structures so it does not favor the helix formation in particular. Therefore, these amino acids are mostly found where their side chains can fit nicely into the beta configuration."

```
Current: Counter({'VAL': 20473, 'LEU': 17167, 'ILE': 16399, 'PHE': 9150,
'ALA': 8617, 'TYR': 8380, 'THR': 7962, 'ARG': 6404, 'LYS': 6337, 'SER':
6187, 'GLY': 6177, 'GLU': 6026, 'GLN': 4060, 'ASP': 3296, 'HIS': 3263,
'ASN': 2936, 'TRP': 2736, 'MET': 2666, 'CYS': 2192, 'MSE': 775, 'PRO':
168, 'MLY': 18, 'KCX': 10, 'UNK': 6, 'CME': 2, 'KPI': 2, 'CSO': 2, 'SEC':
1, 'LVN': 1, 'SEP': 1, 'CSD': 1, 'GPL': 1, 'PTR': 1, '6V1': 1, 'OCY': 1,
'MLZ': 1, 'SMC': 1})

Initial: Counter({'VAL': 16965, 'LEU': 12828, 'ILE': 12054, 'THR': 10851,
'LYS': 9409, 'GLY': 8872, 'ALA': 8441, 'ARG': 8382, 'PHE': 7404, 'GLU':
7258, 'SER': 7106, 'TYR': 6975, 'GLN': 4983, 'HIS': 3852, 'ASN': 3527,
'ASP': 3172, 'TRP': 2588, 'MET': 2294, 'CYS': 2160, 'PRO': 1495, 'MSE':
747, 'MLY': 27, 'CME': 5, 'TYI': 4, 'UNK': 4, 'OCS': 2, 'TPQ': 2, 'CSO':
2, 'MLZ': 2, 'CSX': 1, 'SEC': 1, 'MHO': 1, 'TYQ': 1, 'CSS': 1, '1OP': 1,
'KCX': 1, '6V1': 1, 'YCM': 1, 'OCY': 1})

End: Counter({'VAL': 13869, 'LEU': 12465, 'ILE': 11157, 'SER': 10131,
'ASP': 9991, 'THR': 9128, 'ALA': 8048, 'PHE': 7559, 'ARG': 6386, 'PRO':
6385, 'GLU': 6359, 'GLY': 6358, 'TYR': 6205, 'ASN': 6101, 'LYS': 6017,
'GLN': 4007, 'HIS': 3835, 'CYS': 2504, 'TRP': 2205, 'MET': 2077, 'MSE':
581, 'MLY': 23, 'KCX': 6, 'UNK': 5, 'CSD': 3, 'CSX': 3, 'CSO': 3, 'CME':
3, 'OCS': 2, 'CSS': 1, 'LVN': 1, 'SEP': 1, 'CYQ': 1, 'SME': 1})
```

6. Build Ramachandran plot for a given AA?

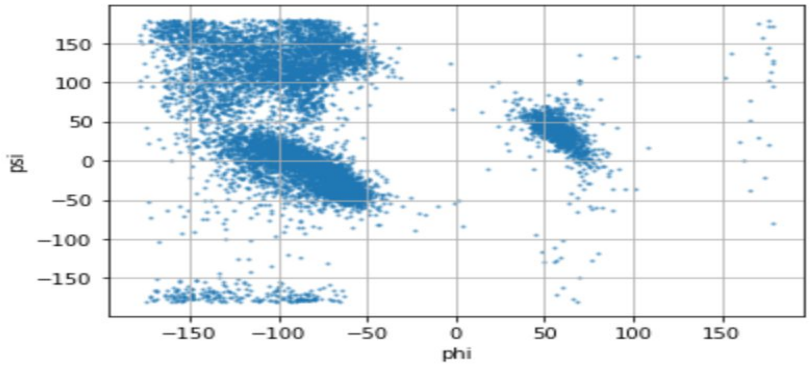
Implementation:

To answer the question, the program uses the same functions as those used to answer #4 to extract atom coordinates and build Ramachandran scatterplot out of them. The only difference is that instead of using helix dataframe, it started with atom dataframe, then only filtered out the rows belonging to the user-specified amino acid and fed them into preprocessing and plotting functions.

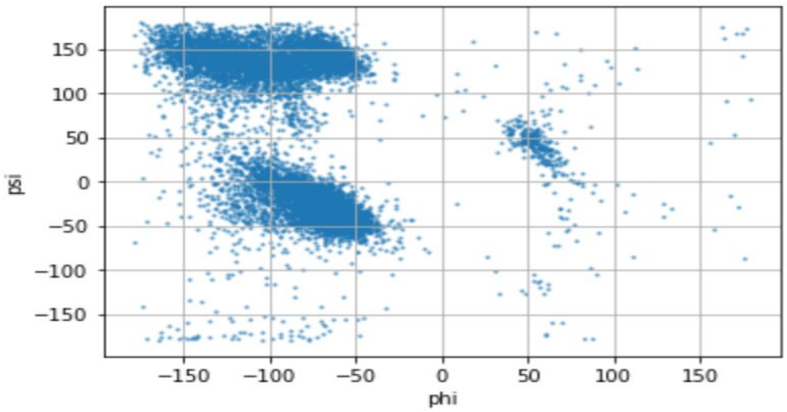
Results:

The plots for all amino acids showed very similar pattern in their Ramanchandran plots. They all showed big custers at (-120, 120) and (-75, -25) while some acids showed third cluster at (50,50). These plots look very similar to those of Helix plots shown in problem 4. Only difference is that instead instead of big cluster showing at (-75, 150), there are more data points to the left of that cluster. Since the majority of non-helix affiliated amino acids belong to beta strand, this behavior is seemingly directly correlated to torsion angles of beta strands.

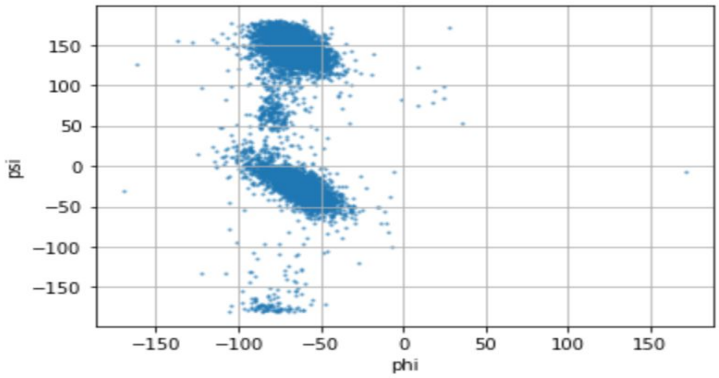
```
Please enter AA name to build Ramachandran plot for (i.e. VAL).ASN
```



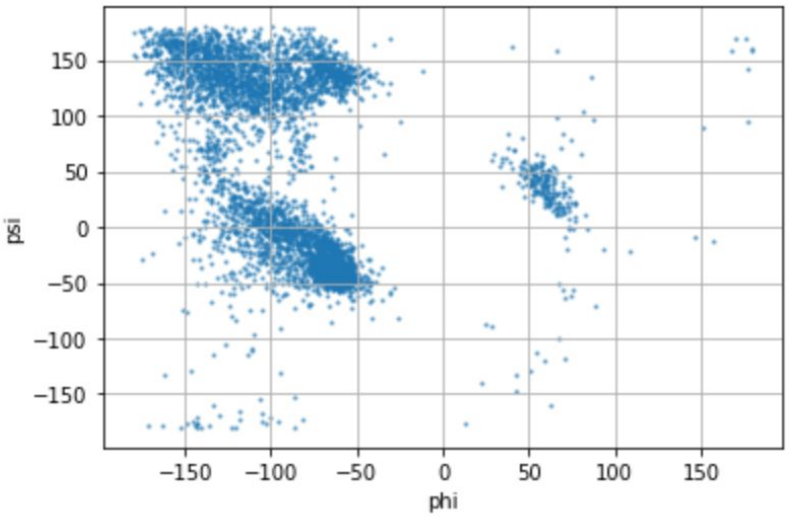

```
Please enter AA name to build Ramachandran plot for (i.e. VAL).GLU
```



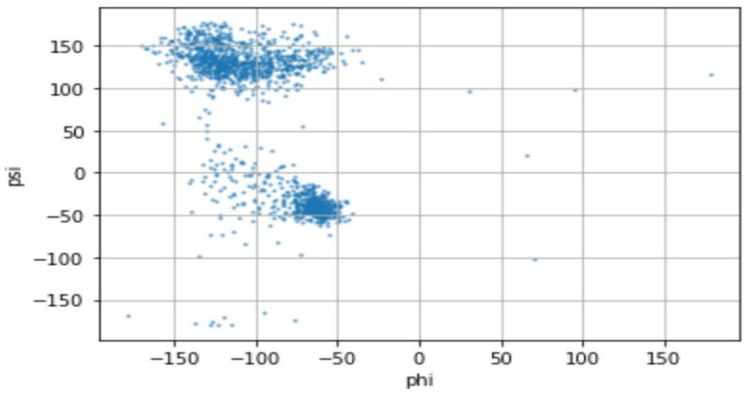
Please enter AA name to build Ramachandran plot for (i.e. VAL).PRO



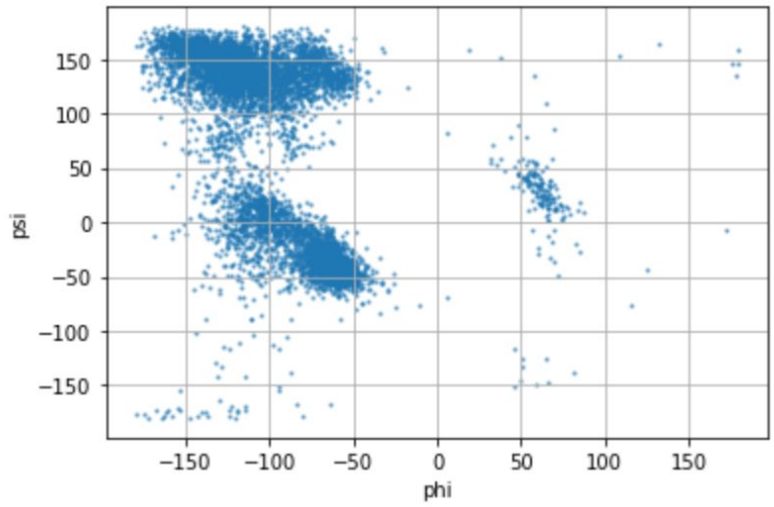
Please enter AA name to build Ramachandran plot for (i.e. VAL).HIS



Please enter AA name to build Ramachandran plot for (i.e. VAL).VAL



```
Please enter AA name to build Ramachandran plot for (i.e. VAL).TYR
```



7. Analyze the length of bonds and the angle between different type of backbone atoms in general and for individual AAs.

Implementation:

For this questions the same parser of preprocessing was used to create an atom dataframe. Since the amount of atom was so large I got a sample from 12AS to 1DQP which was a sample of 399705 lines. The atom data frame was them cleaned to calculate the length from CA to CA atoms and saved to a list. For the analytics, the max, min, median, and mean functions were used on the length results list.

Results:

I expected the lengths to be very similar since they were calculated from CA to CA bond. If the Counter function is used only pairs of two show up to be exact.

```
Distance from CA to CA

Max: 113.35855197116803

Min: 0.0064807406984157805
```

```
Mean: 3.826283882257711

Median: 3.8045274345179854
```

V. Conclusion

In conclusion, our goal was to analyze the given list of proteins, we believe that we were able to accomplish it. We were able to work in a team with each one of us contributing with our sub tasks. Each subtask used the preprocessor to pull our needed data. The results for each question we received are believed to be reasonable or justified with the techniques that we used.