TABLE OF CONTENTS

ABSTRACT

Proteomics is the study of protein, includes protein function and structure. One of the main goals of the field is to explore 3D protein structure. This application follows the main objective of proteomics. The user can explore the structure, and function of proteins involved in cancer. It allows user to see the basic information about the protein, certain key aspects of the protein, and look at its 3D models.

Specific Objectives

- User can access pre-populated data such as features, gene ontology, some structural aspects of the protein
- User look at the 3D structure of protein models

REQUIREMENTS

- **Python 2.7/3**
- **Bio Python**
- **UniProt REST API**
- **Requests module**
- **Pandas**
- **Matplotlib**
- **Pymol(optional)**
- **D3.js(optional)**
- **PyCharm IDE(optional)**

1.0 USAGE

1.1 Installation

The user can access limited about of protein data via a web page at Cancer Proteomics: Visualizer and Analyzer, this will allow users to see pre-populated data for a limited set of proteins. There is a drop-down menu, from where a user can choose the protein of interest.

Alternately, if the user is interested in a protein that is not listed on the web page, they can set the project locally by following these steps

Fork and clone the CancerProteome_Analyzer_Visualizer Repository from GitHub

To import the database

```
$ mysql –u <yourusername> -p <yourpassword> -h localhost proteomics < proteomics.sql
```

Install Dependencies using HomeBrew(if on Windows use pip for all these packages)

```
$ brew update
$ brew install Python
$ brew install mysql
$ brew tap homebrew/science
$ brew install matpoltlib

Pandas, Requests and BioPython should install via pip:
$ pip install pandas
$ pip install biopython
$ pip install requests
```

Server set-up to run CGI scripts (there are a lot of ways to do this)

```
$ brew install node
$ npm install http-server –g
```

To start the server type

```
$ http-server
```

You should see the port at which the page is available, for me it was Port 8080. Open the page at

```
localhost:8080
```

Another alternate is using either a server like Apache or WebBricks

1.2 Steps for Analyzing a Protein

There are two file in the directory that can be used for analysis uniprot.py and pdb.py.

1.21 Sample Analysis using UniProt Script(uniprot.py)

In the project directory

```
$ python
>>> import uniprot
>>> import requests
>>> import pandas as pd
>>> req(uniprot.server,  query='gene:<protein name> AND organism: Human' AND reviewed:yes')
>>> uniprot_list = pd.read_table(StringIO.StringIO(req.text))
>>> uniprot_list.rename(columns={]Organism IS: 'ID'}, inplace=True)
>>> <protein name> = uniprot_list[uniprot.list.ID == 9606]['Entry Name'].tolist()[0]
>>> handle = ExPASy.getsprot_raw(<protein name>)
>>> sp_rec = SwissProt.read(handle)
>>> uniprot.extract_features(sp_rec)
```

1.22 Sample Analysis of a model for above protein using BioPython PDB(pdb.py)

In the project directory

```
$ python
>>> from Bio import PDB
>>> from Bio.PDB import *
>>> import matplotliv.pyplot as plt
>>> repository = PDB.PDBList()
>>> parser = MMCIFParser()
>>> repository.retrieve_pdbfile(<model_of_choice>)
>>> <model name> = parser.get_structure('name', <protein_model.cif>)
>>> pdb.describe_model(<name, <model name>)
>>> pdb.plot(<model name>)
```

***Note** A protein can have many models, with many different configurations, this program and some of its methods are not generic methods. You might have to configure the methods to your model of choice e.g. looking for residues in a chain.*

Alternate would be to configure these scripts according to your protein model of choice. Added benefit of this is you will be able to save this data to the proteomics database. For that add too data and pdb_data lists.

2.0 DESIGN & IMPLEMENTATION

The central philosophy used to design this application is N-tier. There has been a conscious effort to separate the presentation layer, the logic layer, and the data layer.

2.1 Structure

The data layer consists of the data model, and the two Python scripts (uniprot, and pdb). Both the python scripts contain the logic for pulling unstructured protein data from Uniprot API, and BioPython PDB parser. Data was accessed from Uniprot REST API using the requests API, which is a wrapper for web requests. The same scripts also contain all the methods for parsing and analyzing this data. Once the data has been parsed and analyzed, it is stored in the mysql database. The logic layer is the CGI script. The presentation layer consists of JQuery, HTML, CSS and Vanilla JavaScript. JQuery $. ajax method is used to perform asynchronous HTTP requests.

2.11 Data Model

Database contains two tables, cancer table stores the uniprot data, and pdb stores the pdb parser data. There is 1:1 relationship between the two tables, with cancer id being the foreign key constraint.

The database was designed with scalability in mind. Once more models are added in the pdb database, it would be easier to change the relationship to 1:N. Also this allows for addition of multiple tables as size and scope of the data grows. For example, once the pdb table starts growing with could abstract additional model data out of the pdb table, and put it in a new table, without the need to modify the original pdb schema.

2.2 Challenges

2.21 Modularization

Working with a complex and restricted data set has its challenges. One of the biggest issue faced was “how to modularize code?”. One of the design choices made was to keep the protein and protein model analysis separate. The challenge here was working with protein models, each model is different, so designing fit-all methods was difficult. This lead to long verbose python script, and violation of DRY principle.

One way to solve this issue would to treat each model separately, and extract its methods into its own file. This also presents an issue of scalability, but this could easily be solved by grouping models according to their structure similarity, by using a method similar to PDB comparison tool, that does a structure alignment. Then writing a set of generic methods that can be used with these models.

2.22 Scope of data

Since there is so much information contained in the raw scripts, it was hard to narrow the scope of analysis. The solution for this is similar to the points discussed above, once similar models for a protein are found, it would be easier to run the analysis.

The goal of this iteration of the application was to establish certain benchmarks to run initial analysis.

3.0 ANALYSIS

3.1 Uniprot API

[1]Contains a function to perform REST queries. While looking at options to do requests, I also looked at using the BioPython interface, but some of the URLs seem to not work, it could be that it is referencing some old ExPasy URLs. This function was used to query the API for all the proteins of interest. The requests function returns an object that contains information about the protein. To parse this information, I used a method called extract_features(protein), which takes the object as an argument. Some of the feature that I extracted were: length, description, and GO. There are many other features that I decided not to extract, atleast not for this iteration.

3.2 PDB Analysis

For this portion of the analysis, Biopython's PDB module was used to interact with the PDB. For this interaction of the application; I decided to focus on an inverted pyramid approach, by extracting information about the model, its chains, the residues on single chains, and atoms per residues. A mentioned in section 2.0 proteins can have many models, and it is impractical, almost impossible to run analysis simultaneously (unless you build some fancy algorithm) for all the models.

I decided to focus on one model per protein. The approach I took for deciding on a model to study, was initially picking a sub-set of the most widely studied cancer proteins. Assumption I made was that since these are commonly studied their models would be more well established. For there I searched on PDB, for all available models for that protein, and chose one.

I implemented a function that parsed the MMCIF file that was created once the model is downloaded, and extracted the chains, length, and atoms information from the file. This function is generic enough that it works with all models. I was interested in looking at specific residues on chains, since protein stability can have affected by mutations on residues. The challenge here was since each model is different for each protein, I had to analyze all the protein models separately for their [1]residues. I also implemented a plot(model) function that takes a model, and creates a 3D diagram of it.

[1]***NOTE for this iteration of the application I chose to focus on a sub-section of proteins that play a role in cancer. Over several iterations this set will grow.***

4.0 ROADMAP

4.1 ADDITIONAL COMPUTATION

It would be interesting to see atoms that are closer certain residues. We could implement a function that gets coordinates for reference atoms and computes its distance relative to specific residues. We can also calculate the masses for all chains.

The goal of extracting all this information from a model would to ultimately to compare models for different proteins. Understanding the topological difference between two protein models would be crucial in designing a cancer proteome interaction map.

4.2 Module-based approach

The project would need to be refactored to single model analysis. Where each model would have to be treated separately, and analyzed according to its structure, requiring additional methods that help us better understand the model.

4.3 Cancer Proteome Map

A cancer proteome map that displays the relationships between these protein would be the ultimate goal