```
----------------------
Command Line Arguments
----------------------

ExpansionProgram.py:

Example: python ExpansionProgram.py --i=input_file.xlsx --
o=output_file.csv

--i | --input | --inputfile <name of input file including file
extension>
  (required) Specifies the name of the file to be used as input

--o | --output | --outputfile <name of output file including file
extension>
  (required) Specified the name of the file that will be output

-f | -full | -x | -e
  (optional) Specifies the format of data to be output

    -f | -full - Copies the full input data and includes expansion
data
    -x - Includes both expansion data and data used for expansion <--
default
    -e - Includes only expansion data

-p | --page
  (optional) Specifies the page of the excel spreadsheet to convert
to .csv

-1 | -one
  (optional) Specifies to run stage 1 (default: 1 is automatically
selected) <-- default

-2 | -two
  (optional) Specifies to run stage 2 (if chosen without -1/-one, only
stage 2 will
  run)

GenerateTargetsBlastp.py:
Example: python GenerateTargetsBlastp.py --i=input_file.csv --
o=output_file.csv

--i | --input | --inputfile <name of input file including file
extension>
  (required) Specifies the name of the file to be used as input

--o | --output | --outputfile <name of output file including file
extension>
  (required) Specified the name of the file that will be output
```

```
-f | -full | -x | -e
  (optional) Specifies the format of data to be output

    -f | -full - Copies the full input data and includes expansion
data
    -x - Includes both expansion data and data used for expansion <--
default
    -e - Includes only expansion data

ReturnBlastPMutations.py:
Example: python ReturnBlastPMutations.py --i=input_file.csv
--i | --input | --inputfile <name of input file including file
extension>
  (required) Specifies the name of the file to be used as input

ConvertNameToGene.py:
Example: python ConvertNameToGene.py --i=input_file.csv --
out=output_file.csv

--i | --input | --inputfile <name of input file including file
extension>
  (required) Specifies the name of the file to be used as input

--o | --output | --outputfile <name of output file including file
extension>
  (required) Specified the name of the file that will be output

DiffToExpand.py:
Example: python DiffToExpand.py --old=old.csv --new=new.csv

--new <name of new database file including file extension>
  (required) Specifies the name of the file to be used as input

--old <name of old database file including file extension>
  (required) Specified the name of the file to be used as input

--------------
Required Files (version format: MMDDYYYY)
--------------
[Expansion]
1) Database file (on-hand version: cosmicmuts_withresiduelen-NL_KTB
01122018.xlsx)
2) ExpansionProgram.py (current version: 03262018-1)
3) GenerateTargetsBlastp.py (current version: 03262018-1)
4) ReturnBlastPMutations.py (current version: 03262018-1)
5) GRCh38_latest_protein.faa (download the latest version online)
6) ConvertNameToGene.py (current version: 02022018-1)
7) DiffToExpand.py (current version: 06152018-1)
```

[Validation]
1) PSProcess.py (current version: 03292018-1)
2) PreGSSProcess.py (current version: 03292018-1) NOT USED
3) PostGSSProcess.py (current version: 03292018-1) NOT USED

Required Steps Prior to Use

Python3 and BlastP are required to run the complete program (Stage 1 and Stage 2). xlrd is required to run the excel conversion to .csv in ExpansionProgram.py.

Note: Before BlastP can be run, a database needs to be made. Ensure your GRCh38 fasta is in the same directory as the other program files. Then enter the following to the terminal: makeblastdb -in GRCh38_latest_protein.faa -dbtype prot. If you choose to create the database in a different location, you will need to update the code.

It takes a short amount of time to create the files required. They end with .phr, .pin and .psq. These are required to run BlastP as part of Stage 2.

To run the validation portion, you will need PROSITE's psscan. Download the appropriate binary packages at ftp://ftp.lausanne.sib.swiss/pub/software/unix/pftools/. The files required are: ps_scan.pl, psa2msa, pfscan. To extract the information from the PROSITE results, you will need to download the latest prosite.dat file and extract the information with regards to the PROSITE ID. Some examples are provided.

Example Files as Output
----
1) The most updated database file as of January 15, 2018 is 'cosmicmuts_withresiduelen-NL_KTB 01122018.xlsx'.

Note: The .csv equivalent is 'cosmicmuts_withresiduelen-NL_KTB 01122018.csv' generated by the program if .xlsx is fed as input.

2) full_database_expansion.csv is the full copy and expansion of the database.

3) database_expansion_only.csv is the expansion only of the database.

4) database_expansion_and_original.csv is the expansion and the original of the database.

Files Generated
----

1)<output file name>.csv
2)<output file name>
If input file was an excel sheet, then:
<input file name>.csv is also generated
3) Multiple intermediate files

Design Decisions and Issues

ExpansionProgram is designed to simplify the calling of subroutines for database expansion. ExpansionProgram is meant for converting .xls or .xlsx to .csv prior to calling subroutines to be used for database expansion. It detects the input file format and lets the user know if the file format is not compatible. Once a compatible .csv file is obtained, subroutines are called and appropriate arguments as passed.

Original concept was to pipeline stage 1 output into stage 2 input. However, this was not desired as it causes difficulty in solving potential problems if we link the two. As such, this program will now serve as an input conversation program.

Currently, the database has additional two fields to determine duplicates (one as unique identifier and another as indicator of duplicates).

Version of program is now commented on top of each program as MMDDYYYY-Iteration.

ExpansionProgram command lines are too complex (but customizable). Try to find a middle ground between UX and DX.

The program writes then reads the written file as input. We have not considered multiple processes running at the same time writing and reading the same files.

Validation may be better with implementation of genetic ontology annotation. Please see (https://bioconductor.org/packages/release/bioc/vignettes/GOSemSim/inst/doc GOSemSim.html#gosim-and-mgosim-function) for more information. Genetic ontology annotation might require either SQL or continuation in R rather than python. This was attempted but the information provided was uninformative and disregarded.

For now, validation is based on identifying appropriate protein domains and motifs between the BLASTp results and the query sequence. The validation after PROSITE is a simple check to validate that gene and functional impact are meaningful against a gene database known for tumor suppressors and oncogenes.

PROSITE takes a long time to run, thus it is more effective to store the sequences and scan each sequence and store the matching motifs. This stored information is then matched with the sequences in the expanded database. This concept is to be returned to at a future time. Some sample (spaghetti) code has been provided. It would also be effective to split the data into multiple sets to lower the overall time required by running multiple processes of PROSITE. This is where DiffToExpand.py comes in. DiffToExpand.py finds the difference between the stored information and the new information to run PROSITE on to lower the overall required time.

Subroutines

Both stage 1 and 2 are implemented to fully expand the database by first amino acid substitution then cluster homology. Validation stages have been implemented, but PROSITE can be improved.

Known Issues

The current program does not take care of duplicates (from both input and output). (i.e. if a prediction mutation is an actual mutation, see CDKN2A R112M, R112L)

Automated column identification lacks elegance (look into cleaning up the code).

GRCh38 requires external download from NCBI's GRCh38_latest_protein.faa.

No code to extract information from prosite.dat file to be used to checked for domains and motifs. (The parsed information is provided but should be checked for any updates)

Fixed Issues and Updates

(Jan. 15, 2018): fixed bug where empty aa sequence field was throwing error,
automated column identification.
(Jan. 17, 2018): stage 1 now truncates the last two columns from the database which
used identify duplicates.
stage 2 is now implemented as part of ExpansionProgram.py.
stage 2 output is now uniform with current database layout.
files are now used as: stage 1: input --> input.csv --> output.csv; stage 2:
output.csv --> output_BlastP_Output.csv
(Jan. 18, 2018): changed stage 1 and stage 2's output under references column to

'amino acid substitution prediction' and 'cluster homology prediction'
respectively.
(Jan. 23, 2018): updated stage 2 header lines so they are now separate
columns.
updated stage 1 header lines to remove trailing ',' causing issues in
format for
stage 2
ExpansionProgram.py defaults are now expansion and original (-x) and
now offer
options to choose stage 1 or stage 2 or both which pipelines stage 1
into stage 2
cleaned up code for ExpansionProgram.py, ReturnBlastPMutations.py, and
GenerateTargetsBlastP.py
added check to make sure output name is not the same as input name
(Jan. 31, 2018): small updates to ReturnBlastPMutations.py to make
sure format is
usable for conversion of protein names to genes (in progress)
(Feb. 2, 2018): added ConvertNameToGene.py. corrected
ExpansionProgram.py help info
(Mar. 26, 2018): fixed format bug; gene name conversion called after
stage 2 automatically
(Jun. 15, 2018): Added DiffToExpand.py in hopes for lowering PROSITE
runtime
Compiled all necessary files together
(Jul. 9, 2018): Files compiled and handed over to be maintained