P3_Data_Analysis_2

December 17, 2023

1 Project 3.2 - Data Analysis and Web Scraping

The second part of the fourth project is similar to the first part. However, while the first part dealt with `Target Pref. Name = Integrins`, which appeared many times in the *DLiP* database, and therefore, we switched those values with more specific ones from the *chEMBL* database, obtaining even more data because many molecules with `Target Pref. Name = Integrins` had more than one match with the corresponding values in the *chEMBL* database (as a reminder, we joined the data frames by the SMILES molecule values). In this part of the project (i.e., 3.2), we are going to deal with `Target Pref. Name = BCL2-Like_BAX`.

1.1 Import Libraries

[109]:
```
import os
import pandas as pd

def PRINT(sent) -> None : print(f"{'-'*80}\n{sent}\n{'-'*80}")
```

1.2 Merge the Data

In this phase of the project, our approach involves merging the enhanced data frame (following the addition of more informative target pref. names in the previous project, *3.1*) with another data frame obtained from a GitHub repository that also utilizes the *DLiP* database for their research.

The merging process encompasses several steps:

- Clone the relevant GitHub repository.
- Extract the desired data frame.
- Retain only the *dlip-id* and *Target_Pref_Name* columns, discarding all others.
- Merge it with our data frame, i.e., *ppi_cs1033_df_extended*.

Following the merge, the resulting data frame will consist of three columns:

- Target Pref Name == BCL2-Like_BAX derived from our data frame.
- DLiP-ID from our data frame.
- Informative Target Pref Name extracted from the external data frame.

Similar to the previous project (*3.1*), we anticipate Null values where there is no match between corresponding *DLiP-ID* values.

1.2.1 Clone to the Correct GitHub Repository

```
[2]: !git clone https://github.com/sun-heqi/MultiPPIMI.git
```

```
Cloning into 'MultiPPIMI'…
```

1.2.2 Generate Wanted Data Frame

```
[39]: file_path = 'MultiPPIMI\\data\\folds\\S1'  # Replace with your file path
      fold = 1
      train_path = os.path.join(file_path, f'train_fold{fold}.csv')
      valid_path = os.path.join(file_path, f'valid_fold{fold}.csv')
      test_path = os.path.join(file_path, f'test_fold{fold}.csv')

      train_df = pd.read_csv(train_path)
      valid_df = pd.read_csv(valid_path)
      test_df = pd.read_csv(test_path)

      full_ppim_dataset = pd.concat([train_df, valid_df, test_df], axis = 0).
        ↪drop_duplicates()
```

```
[40]: full_ppim_dataset
```

```
[40]:      dlip_id                                             SMILES  \
      0     C004CP  Cc1ccnc(N2C(=O)[C@](C)(CC(=O)O)C[C@@H](c3cccc(…
      1     T0036J  O=C(/C=C/c1ccc(Sc2ccc(Cl)cc2Cl)c(Cl)c1)NCCCN1C…
      2     C00459  CCOc1ccccc1N1CCN(C(=O)C2(Oc3ccc(C(F)(F)F)cc3)C…
      3     T006CF  N=C(N)NCCC[C@@H]1NC(=O)CNC(=O)CSC[C@@H](C(=O)O…
      4     T000GZ  CCOC(=O)NC(=N)c1ccc(C(=O)N[C@@H](Cc2ccc(O)cc2)…
      …        …                                                 …
      4647  T00734  CC(C)(C)C[C@@H]1C=C(C(=O)NCCN2CCOCC2)[C@@H](c2…
      4648  T0073W  CN(C)c1ccc(-c2cn3c(n2)CCC3)cc1CNC(=O)Cc1ccc(Cl…
      4649  T0075T  Cl.O=C(/C=C/c1ccc(C(F)(F)F)cc1)c1ccc(OCCCN2CCO…
      4650  T0076M  CCS(=O)(=O)N(C[C@@H](C1CC1)N1C(=O)[C@](C)(CC(=…
      4651  T0079E  COc1cnc2n1C(C)(Cc1ccc(Br)cc1)C(=O)N2c1cc(Cl)cc…

                                       Target_Pref_Name  \
      0          Tumour suppressor p53/oncoprotein Mdm2
      1     Intercellular adhesion molecule (ICAM-1), Inte…
      2          Tumour suppressor p53/oncoprotein Mdm2
      3                      Integrin alpha-IIb/beta-3
      4                      Integrin alpha-IIb/beta-3
      …                                             …
      4647                                FKBP1A/FK506
      4648                                FKBP1A/FK506
      4649                                FKBP1A/FK506
      4650                                FKBP1A/FK506
      4651                                FKBP1A/FK506
```

```
     Common_Target_Pref_Name uniprot_id1 uniprot_id2 uniprot_id3 uniprot_id4  \
0                   P53/HDM2      P04637      Q00987         NaN         NaN
1                  Integrins      P05362      P20701      P05107         NaN
2                   P53/HDM2      P04637      Q00987         NaN         NaN
3                  Integrins      P08514      P05106         NaN         NaN
4                  Integrins      P08514      P05106         NaN         NaN
…                        …         …         …          …          …
4647             FKBP1A/FK506      P62942          na         NaN         NaN
4648             FKBP1A/FK506      P62942          na         NaN         NaN
4649             FKBP1A/FK506      P62942          na         NaN         NaN
4650             FKBP1A/FK506      P62942          na         NaN         NaN
4651             FKBP1A/FK506      P62942          na         NaN         NaN

      uniprot_id5  ppi_label  label
0             NaN        6.0      1
1             NaN      102.0      1
2             NaN        6.0      1
3             NaN      104.0      1
4             NaN      104.0      1
…            …         …     …
4647          NaN        4.0      0
4648          NaN        4.0      0
4649          NaN        4.0      0
4650          NaN        4.0      0
4651          NaN        4.0      0

[23260 rows x 11 columns]
```

[42]:
```
full_ppim_dataset.to_csv('full_ppim_dataset.csv', index=False)
```

Drop Redundant Columns

[43]:
```
full_ppim_dataset_ = full_ppim_dataset[['dlip_id', 'Target_Pref_Name']]
```

[44]:
```
full_ppim_dataset_
```

[44]:
```
     dlip_id                                   Target_Pref_Name
0     C004CP             Tumour suppressor p53/oncoprotein Mdm2
1     T0036J  Intercellular adhesion molecule (ICAM-1), Inte…
2     C00459             Tumour suppressor p53/oncoprotein Mdm2
3     T006CF                         Integrin alpha-IIb/beta-3
4     T000GZ                         Integrin alpha-IIb/beta-3
…      …                                                …
4647  T00734                                      FKBP1A/FK506
4648  T0073W                                      FKBP1A/FK506
4649  T0075T                                      FKBP1A/FK506
4650  T0076M                                      FKBP1A/FK506
```

```
4651  T0079E                                       FKBP1A/FK506

[23260 rows x 2 columns]
```

Rename the Name of the Columns

```
[45]: full_ppim_dataset_.rename(columns={'dlip_id': 'DLiP-ID', 'Target_Pref_Name':
       ↪'Informative Target Pref Name'}, inplace=True)

      full_ppim_dataset_.head(5)
```

```
C:\Users\gavvi\AppData\Local\Temp\ipykernel_36912\111510329.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  full_ppim_dataset_.rename(columns={'dlip_id': 'DLiP-ID',
'Target_Pref_Name':'Informative Target Pref Name'}, inplace=True)
```

```
[45]:   DLiP-ID                     Informative Target Pref Name
      0  C004CP            Tumour suppressor p53/oncoprotein Mdm2
      1  T0036J  Intercellular adhesion molecule (ICAM-1), Inte…
      2  C00459            Tumour suppressor p53/oncoprotein Mdm2
      3  T006CF                        Integrin alpha-IIb/beta-3
      4  T000GZ                        Integrin alpha-IIb/beta-3
```

1.2.3 Load out Data Frame

Next, we can load our extended data frame after project *3.1*

```
[46]: ppi_cs1033_df_extended = pd.read_csv('ppi_cs1033_extended.csv')
```

```
[47]: ppi_cs1033_df_extended
```

```
[47]:      DLiP-ID                            Canonical SMILES(RDKit)       MW  \
      0      T00000  CCC(C)(C)C(=O)C(=O)N1CCCCC1C(=O)OCCCc1cc(OC)cc…  433.545
      1      T00001  COc1ccccc1C1C2=C(N=c3s/c(=C\c4ccc(/C=C/C(=O)O)…  520.610
      2      T00002     CSc1ccc(-c2c(C#N)c3cccc(Cl)n3c2NCCc2ccccc2)cc1  417.965
      3      T00003  COc1cccc(OC)c1-c1ccc(C[C@H](NC(=O)[C@@H]2CCCN2…  519.554
      4      T00003  COc1cccc(OC)c1-c1ccc(C[C@H](NC(=O)[C@@H]2CCCN2…  519.554
      …       …                                                …      …
      65560  T00014  Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…  502.593
      65561  T00014  Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…  502.593
      65562  T00014  Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…  502.593
      65563  T00014  Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…  502.593
      65564  T00014  Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…  502.593

             XLogP  HBA  HBD     PSA  nRotatableBonds  nRings  \
0      3.548    6    0    82.14             10       2
1      5.492    6    1    80.89              5       6
2      7.388    4    1    40.23              6       4
3      5.147    7    2   131.24             10       4
4      5.147    7    2   131.24             10       4
…        …    …   …      …              …      …
65560  2.100    5    4   144.91              9       3
65561  2.100    5    4   144.91              9       3
65562  2.100    5    4   144.91              9       3
65563  2.100    5    4   144.91              9       3
65564  2.100    5    4   144.91              9       3

                              Target Pref Name Common Target Pref Name  \
0                                 FKBP1A/FK506            FKBP1A/FK506
1      Bcl-2 and Bcl-XL with BAX; BAK and BID        BCL-like/BAX,BAK
2                         Neuropilin-1/VEGF-A     Neuropilin-1/VEGF-A
3                     Integrin alpha-4/beta-7               Integrins
4                     Integrin alpha-4/beta-1               Integrins
…                                          …                     …
65560                   Integrin alpha2/beta1               Integrins
65561           Integrin alpha1/beta1 complex               Integrins
65562                 Integrin alpha-5/beta-1               Integrins
65563                 Integrin alpha-4/beta-1               Integrins
65564                   Integrin alpha2/beta1               Integrins

         Active
0        Active
1      Inactive
2        Active
3        Active
4        Active
…          …
65560  Inactive
65561  Inactive
65562  Inactive
65563  Inactive
65564  Inactive

[65565 rows x 12 columns]
```

[48]:
```
# Check the number of rows in partial_ppi_cs1033_df_ containing 'BCL2-Like_BAX␣
↪before the merge
before_merge_count_initial =␣
↪ppi_cs1033_df_extended[ppi_cs1033_df_extended['Target Pref Name'] ==␣
↪'BCL2-Like_BAX'].shape[0]
```

```
[49]: PRINT(f'Number of rows contain target pref. name -> BCL2-Like_BAX :␣
        ↪{before_merge_count_initial}\nNumber of rows in total ->␣
        ↪{ppi_cs1033_df_extended.shape[0]}')
```

```
------------------------------------------------------------------------------------------------
Number of rows contain target pref. name -> BCL2-Like_BAX : 518
Number of rows in total -> 65565
------------------------------------------------------------------------------------------------
```

1.2.4 Merge Step

```
[50]: # Select rows in ppi_cs1033_df_extended where 'Target Pref Name' is␣
        ↪'BCL2-Like_BAX'
      bcl2like_bax_cs1033_rows =␣
        ↪ppi_cs1033_df_extended[ppi_cs1033_df_extended['Target Pref Name'] ==␣
        ↪'BCL2-Like_BAX'].copy()

      # Merge the selected rows from ppi_cs1033_df_extended with full_ppim_dataset
      result_df = pd.merge(bcl2like_bax_cs1033_rows[['Target Pref Name', 'DLiP-ID']],
                           full_ppim_dataset_[['DLiP-ID', 'Informative Target Pref␣
        ↪Name']],
                           how='left',
                           left_on='DLiP-ID',
                           right_on='DLiP-ID'
                          )

      # Check for missing values in the merged data frame
      missing_values = result_df[result_df['Target Pref Name'].isnull()]
```

```
[51]: result_df
```

```
[51]:      Target Pref Name DLiP-ID Informative Target Pref Name
      0       BCL2-Like_BAX  I0011I                          NaN
      1       BCL2-Like_BAX  I00006                     BCL2/BAX
      2       BCL2-Like_BAX  I00006                     BCL2/BAX
      3       BCL2-Like_BAX  I00009                     BCL2/BAX
      4       BCL2-Like_BAX  I00009                     MCL1/BAX
      …                 …      …                          …
      1226    BCL2-Like_BAX  I000AB                          NaN
      1227    BCL2-Like_BAX  I0019J                     MCL1/BAX
      1228    BCL2-Like_BAX  I0019J                     MCL1/BAX
      1229    BCL2-Like_BAX  I0019J                     BCL2/BAX
      1230    BCL2-Like_BAX  I0019J                     BCL2/BAX

      [1231 rows x 3 columns]
```

1.2.5 Explore the Results

```
[55]: result_df.to_csv('merged_df_BCL2-Like_BAX.csv', index=False)
```

```
[62]: number_of_unmatched_DLiP_ID = result_df['Informative Target Pref Name'].
        ↪isnull().sum()
```

```
[68]: number_of_matches = result_df.shape[0] - number_of_unmatched_DLiP_ID
```

```
[112]: PRINT(f'The number of unmatched values in "DLiP-ID" ->␣
         ↪{number_of_unmatched_DLiP_ID}')
       PRINT(f'The number of time "Target Pref Name" = BCL2-Like_BAX in␣
         ↪ppi_cs1033_df_extended -> {before_merge_count_initial}')
       PRINT(f'The number of matched DLiP-ID that found for them unique target pref.␣
         ↪name for BCL2-Like_BAX -> {number_of_matches}')
       PRINT(f'The number of added rows to our data frame -> {number_of_matches -␣
         ↪before_merge_count_initial}')
```

```
--------------------------------------------------------------------------------
The number of unmatched values in "DLiP-ID" -> 195
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
The number of time "Target Pref Name" = BCL2-Like_BAX in ppi_cs1033_df_extended
-> 518
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
The number of matched DLiP-ID that found for them unique target pref. name for
BCL2-Like_BAX -> 1036
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
The number of added rows to our data frame -> 518
--------------------------------------------------------------------------------
```

From the results, it's apparent that we have additional rows in our data frame, doubling the count from the initial data frame rows with *BCL2-Like_BAX* as their target pref. name.

Possible reasons for this discrepancy include:

- One explanation could be that the *full_ppim_dataset* data frame contained more informative values for the same molecules with matching *DLiP-ID*.
- Additionally, the *full_ppi_dataset* included instances of molecules appearing multiple times but with different *Target Pref. Names*. Consequently, when searching for matches based on the molecules' *DLiP-ID*, we obtained more than one match for some molecules, resulting in duplicated data.

1.3 Generate New Extended Data Frame

The next step is to create a new extended data frame that will also include all the matches obtained from the previous merge step. We anticipate obtaining a larger data frame.

[73]:
```
# Create an empty list to store the modified rows
modified_rows = []

# Variable in order ro verify we indeed visited in each row
count = 0
count_unmatched_rows = 0
# Loop through each row in ppi_cs1033_df_extended
for index, row in ppi_cs1033_df_extended.iterrows():
    target_pref_name = row['Target Pref Name']

    # Check if the 'Target Pref Name' is 'BCL2-Like_BAX'
    if target_pref_name == 'BCL2-Like_BAX':
        # Find matches in result_df based on 'DLiP-ID'
        matches = result_df[result_df['DLiP-ID'] ==␣
 ↪row['DLiP-ID']]['Informative Target Pref Name'].tolist()
        #print(matches)
        count+=1

        if pd.isna(matches[0]):
            modified_rows.append(row)
            count_unmatched_rows+=1
            continue
        # Duplicate the row for each match and update 'Target Pref Name'
        for match_value in matches:
            duplicated_row = row.copy()
            duplicated_row['Target Pref Name'] = match_value
            modified_rows.append(duplicated_row)
    else:
        count+=1
        # If 'Target Pref Name' is not 'BCL2-Like_BAX', keep the original row
        modified_rows.append(row)


# Create a new data frame with the modified rows
modified_df = pd.DataFrame(modified_rows)
modified_df.reset_index(drop=True, inplace=True)
```

[76]:
```
PRINT(f'Done.\nVisited in :{count} rows, which means skipped over␣
 ↪{ppi_cs1033_df_extended.shape[0]-count} rows (should be 0)')
PRINT(f'Number of unmatched rows we got is -> {count_unmatched_rows}, and we␣
 ↪know we should get 195')
```

```
--------------------------------------------------------------------------------
--------------------
Done.
Visited in :65565 rows, which means skipped over 0 rows (should be 0)
--------------------------------------------------------------------------------
--------------------
```

```
--------------------------------------------------------------------------------
--------------------
Number of unmatched rows we got is -> 195, and we know we should get 195
--------------------------------------------------------------------------------
--------------------
```

[77]:
```python
modified_df.shape[0], ppi_cs1033_df_extended.shape[0]
```

[77]:
```
(66278, 65565)
```

[82]:
```python
#modified_df.shape[0] - ppi_cs1033_df_extended.shape[0] - 195
```

[82]:
```
518
```

1.3.1 Save the Data Frame

[83]:
```python
modified_df.to_csv('ppi_cs_1033_extended_f.csv', index=False)
```

1.4 Explore the Results

1.4.1 Check How Many Unique SMILES Got Unmatched

[86]:
```python
unmatched_DLiP_ID_df = result_df[result_df['Informative Target Pref Name'].
 ↪isnull()][['DLiP-ID']]
```

[87]:
```python
unmatched_DLiP_ID_df
```

[87]:
```
     DLiP-ID
0     I0011I
7     I0000A
12    I0000C
13    I0000F
28    I0001E
…        …
1196  I001IG
1201  I001IK
1210  I001IV
1219  I001JR
1226  I000AB

[195 rows x 1 columns]
```

[88]:
```python
# Check the number of unique values
num_unique_values = unmatched_DLiP_ID_df['DLiP-ID'].nunique()

# Print the result
print(f"Number of unique DLiP-ID values: {num_unique_values}")
```

```
# Save the unique DLiP-ID values to a new DataFrame
unique_DLiP_ID_df = pd.DataFrame({'DLiP-ID': unmatched_DLiP_ID_df['DLiP-ID'].
  ↪unique()})
```

```
Number of unique DLiP-ID values: 195
```

[92]:
```
unique_DLiP_ID_df
```

[92]:
```
    DLiP-ID
0    I0011I
1    I0000A
2    I0000C
3    I0000F
4    I0001E
..      …
190  I001IG
191  I001IK
192  I001IV
193  I001JR
194  I000AB

[195 rows x 1 columns]
```

[90]:
```
unmatched_DLiP_r = modified_df[['DLiP-ID', 'Canonical SMILES(RDKit)']]
```

[91]:
```
unmatched_DLiP_r
```

[91]:
```
       DLiP-ID                           Canonical SMILES(RDKit)
0       T00000  CCC(C)(C)C(=O)C(=O)N1CCCCC1C(=O)OCCCc1cc(OC)cc…
1       T00001  COc1ccccc1C1C2=C(N=c3s/c(=C\c4ccc(/C=C/C(=O)O)…
2       T00002     CSc1ccc(-c2c(C#N)c3cccc(Cl)n3c2NCCc2ccccc2)cc1
3       T00003  COc1cccc(OC)c1-c1ccc(C[C@H](NC(=O)[C@@H]2CCCN2…
4       T00003  COc1cccc(OC)c1-c1ccc(C[C@H](NC(=O)[C@@H]2CCCN2…
…          …                                                 …
66273   T00014  Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…
66274   T00014  Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…
66275   T00014  Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…
66276   T00014  Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…
66277   T00014  Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…

[66278 rows x 2 columns]
```

[94]:
```
# Create an empty list to store the modified rows
rows = []

# Loop through each row in ppi_cs1033_df
for index, row in unique_DLiP_ID_df.iterrows():
    DLiP_ID_195 = row['DLiP-ID']
    for idx, r in unmatched_DLiP_r.iterrows():
        DLiP_ID_ppi_cs = r['DLiP-ID']

        if DLiP_ID_195 == DLiP_ID_ppi_cs:
            rows.append(r)

r_df = pd.DataFrame(rows)
r_df.reset_index(drop=True, inplace=True)
```

[97]:
```
r_df
```

[97]:
```
    DLiP-ID                              Canonical SMILES(RDKit)
0    I00011I    COC1=CC(c2cc3ccccc3[nH]2)=N/C1=C/c1[nH]c(C)cc1C
1    I0000A   CC(C)CN(Cc1cccc(CN(Cc2ccc(-c3ccc(F)cc3)cc2)S(=…
2    I0000C                 Cc1cc(=O)c2c(O)c(O)c(O)c(CC(C)C)c2o1
3    I0000F   COC1(CC(C)C)CCN(c2ccc(C(=O)NS(=O)(=O)c3ccc(N[C…
4    I0001E   CN(C)CC[C@H](CSc1ccccc1)Nc1ccc(S(=O)(=O)NC(=O)…
..       …                                                   …
190  I001IG   CCOc1ccc(-c2sc(-c3ccc4c(c3)N(C(=O)Nc3nc5ccccc5…
191  I001IK   O=C(N[C@H]1C[C@@H]1c1ccccc1)c1ccc(CN(Cc2ccc(F)…
192  I001IV   CC(C)CCNC(=O)c1ccc(CN(Cc2ccc(F)cc2)S(=O)(=O)c2…
193  I001JR   CCOC(=O)[C@@H]1Cc2ccccc2CN1C(=O)c1ccccc1-n1nc(…
194  I000AB   CC(C)C[C@@H](C(=O)O)N1C(=O)/C(=C/c2ccc(Br)cc2)…

[195 rows x 2 columns]
```

[98]:
```
r_df_ = r_df
```

[99]:
```
r_df_ = r_df_['Canonical SMILES(RDKit)'].drop_duplicates()
```

[103]:
```
r_df_.to_csv('unmatched_df_only_SMILES.csv',index=False)
```

[102]:
```
len(r_df_)
```

[102]:
```
166
```