PPI Prediction Project

January 22, 2024

1 PPI Prediciton Project

The upcoming project revolves around predicting Protein-Protein Interactions (PPI). Our objective is to extract data from our Timbal dataset, which currently includes UniProt target information but lacks UniProt partner details.

The project will involve taking the dataset that indeed contains both UniProt partner and UniProt target information. We will segregate molecules and their corresponding SMILES values that share the same UniProt target. This subset of data will be used for subsequent predictions, and new datasets will be constructed to encompass all such molecules.

Subsequently, we will visualize our datasets and analyze the data distribution to gain a deeper understanding.

Following that, we plan to develop three models: - Random Forest Multiclass Classifier Model - XGBoost Multiclass Classifier Model - GraphConvModel Multiclass Classifier Model

The first two models will be implemented on two types of dataframes for each dataset. The first dataframe will utilize feature augmentation techniques with RDKitDescriptors, while the second will employ Morgan Fingerprint.

The last model will be implemented using the DeepChem library.

For each dataset, we will select the most suitable model. Finally, we will predict the UniProt partners for each dataset using their corresponding models. Following the prediction phase, we will comataframe containing all the predicted data of interest.

- RDKit library official website : https://www.rdkit.org/docs/index.html
- DeepChem library official website : https://deepchem.io/://deepchem.io/

[1]:
```
import deepchem as dc
import matplotlib.pyplot as plt
import numpy as pd
import pandas as pd
import seaborn as sns
import pickle # Inorder to save data frame dictionary
import os
```

```
WARNING:tensorflow:From C:\Users\gavvi\anaconda3\Lib\site-
packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

```
WARNING:tensorflow:From C:\Users\gavvi\anaconda3\Lib\site-
packages\tensorflow\python\util\deprecation.py:588: calling function (from
tensorflow.python.eager.polymorphic_function.polymorphic_function) with
experimental_relax_shapes is deprecated and will be removed in a future version.
Instructions for updating:
experimental_relax_shapes is deprecated, use reduce_retracing instead
```

[2]:
```
def PRINT(text) -> None: print(f"{'~'*80}\n{text}\n{'~'*80}")

def is_numeric(value):
    try:
        float(value)
        return True
    except (ValueError, TypeError):
        return False

def print_dict_meaningful(dictionary):
    for key, value in dictionary.items():
        if is_numeric(value):
            formatted_value = "{:.3f}".format(float(value))
        else:
            formatted_value = value
        print(f'{key}: {formatted_value}')
```

1.1 Preparing Datasets for Predictive Modeling

1.1.1 Load Required Datasets

[3]:
```
pwd
```

[3]:
```
'C:\\Users\\gavvi\\Desktop\\Programming\\GitHub\\DeepLearningResearchStarship\\P
roject 4 Protein Relationship Prediction'
```

[4]:
```
pred_dataset_path = "data/dataset_for_prediction.csv"
ChEMBL_integrin_dataset_path = "data/ChEMBL_Integrins.csv"
```

[5]:
```
pred_df = pd.read_csv(pred_dataset_path)

pred_df.head(5)
```

[5]:
```
                                              smiles uniprot_id1
0  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…      P13612
1  C\C=C\[C@@H](CC(=O)O)NC(=O)C[C@@H](CC(C)C)NC(=…      P05556
2  CN1[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O…      P05106
3    OC(=O)C(CNC(=O)CCCCc1ccc2CCCNc2n1)c3cnc4ccccc4c3      P05106
4  OC(=O)C[C@H](NC(=O)CN1CCC[C@@H](CCC2CCNCC2)C1=…      P05106
```

Next, we aim to rename the column *uniprot_id1* to *uniprot_id*. The rationale behind this decision is that we intend to search for this value in the *ChEMBL* data frame within both the *uniprot1* and *uniprot2* columns. To minimize confusion, we will rename this column

```
[6]: pred_df = pred_df.rename(columns={'uniprot_id1':'uniprot_id'})

     PRINT(f'Renamed column name: {pred_df.columns[1]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Renamed column name: uniprot_id
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[15]: chmbl_df = pd.read_csv(ChEMBL_integrin_dataset_path)

      chmbl_df.head(5)
```

```
[15]:                             Canonical SMILES(RDKit)         Target Pref Name  \
 0  N=C(N)NCCC[C@H](NC(=O)[C@H](CCCNC(=N)N)NC(=O)[…  Integrin alpha-4/beta-7
 1  N=C(N)NCCC[C@H](NC(=O)CCCC[C@@H]1SC[C@@H]2NC(=…  Integrin alpha-4/beta-7
 2  N#Cc1ccc(-c2ccc(C[C@H](NC(=O)[C@H](CCCNC(=N)N)…  Integrin alpha-4/beta-7
 3  N=C(N)NCCC[C@H](NC(=O)CCCC[C@@H]1SC[C@@H]2NC(=…  Integrin alpha-4/beta-7
 4  N=C(N)NCCC[C@H](NC(=O)CCCC[C@@H]1SC[C@@H]2NC(=…  Integrin alpha-4/beta-7

        Organism UniProt1 UniProt2 UniProt3 UniProt4 UniProt5
 0  Mus musculus   Q00651   P26011      NaN      NaN      NaN
 1  Mus musculus   Q00651   P26011      NaN      NaN      NaN
 2  Mus musculus   Q00651   P26011      NaN      NaN      NaN
 3  Mus musculus   Q00651   P26011      NaN      NaN      NaN
 4  Mus musculus   Q00651   P26011      NaN      NaN      NaN
```

1.1.2 Generate Unique Datasets

Prechecks

```
[22]: unique_proteins = pred_df["uniprot_id1"].unique()
```

```
[26]: PRINT(f"The unique proteins we want to predict their partners in the PPI are :
       ↪\n {unique_proteins}\n\nThe are total {len(unique_proteins)} such proteins")
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The unique proteins we want to predict their partners in the PPI are :
 ['P13612' 'P05556' 'P05106' 'P05107' 'P08648' 'P17301']

The are total 6 such proteins
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Datasets Generation Phase The next step is to generate six datasets, each for the protein for which we intend to create a deep learning model to predict its companion in PPI (i.e., the second UniProt_id).

The way we are going to achieve this is by taking each unique *UniProt_id* value, searching for all the rows in the *ChEMBL* data frame we loaded from the previous project, where that *UniProt_id* value is one of their *UniProt_id{i}* columns, where i [1,5].

Each dataset will contain all the molecules' *SMILES* values, with both *UniProt_ids* forming the connection.

From these datasets, we will proceed to train our model. Thus, we can provide the unique SMILES value along with the UniProt_id to the model, and it will predict its partner.

```
[69]: protein_dataframes = {}

      for protein in unique_proteins:
          # Initialize an empty list to store rows for the current protein
          rows_for_protein = []

          # Iterate over each row in the ChEMBL DataFrame
          for index, row in chmbl_df.iterrows():
              # Check if the current protein is present in any of the UniProt columns
              if protein in row[['UniProt1', 'UniProt2', 'UniProt3', 'UniProt4',␣
       ↪'UniProt5']].values:
                  # Determine the correct order (UniProt1 and UniProt2) in the new␣
       ↪data frame
                  if row['UniProt1'] == protein:
                      relevant_info = [row['Canonical SMILES(RDKit)'],␣
       ↪row['UniProt1'], row['UniProt2']]
                  elif row['UniProt2'] == protein:
                      relevant_info = [row['Canonical SMILES(RDKit)'],␣
       ↪row['UniProt2'], row['UniProt1']]
                  else:
                      relevant_info = []

                  if relevant_info:
                      rows_for_protein.append(relevant_info)

          if rows_for_protein:
              protein_dataframes[protein] = pd.DataFrame(rows_for_protein,␣
       ↪columns=['SMILES', 'UniProt1', 'UniProt2'])
```

```
[49]: protein_dataframes['P13612']
```

```
[49]:                                                  SMILES UniProt1 UniProt2
      0     COc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2Cl)C(=O)…   P13612   P26010
      1     Cc1ccccc1NC(=O)Nc1ccc(CC(=O)N2C[C@@H](F)C[C@H]…   P13612   P26010
      2     CN(C)Cc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2Cl)C…   P13612   P26010
      3     Cc1cccc(Cl)c1C(=O)N[C@@H](Cc1ccc(NC(=O)c2c(Cl)…   P13612   P26010
      4     COc1cnn(C)c(=O)c1-c1ccc(C[C@H](NC(=O)c2c(C)noc…   P13612   P26010
      …                                                 …        …        …
      1969  CC(C)(C)[C@H]1CC[C@H](C[C@H](NC(=O)[C@@H]2CCC(…   P13612   P05556
      1970  O=C(Nc1ccc(C[C@H](/N=c2\c(O)c(O)\c2=N/Cc2ccccc…   P13612   P05556
      1971  N#Cc1cccc(S(=O)(=O)N2C[C@H](N3CCC(F)CC3)C[C@H]…   P13612   P05556
      1972  CCCCS(=O)(=O)N[C@@H](Cc1ccc(OCCCCC2CCNCC2)cc1)…   P13612   P05556
      1973  O=C(Cc1ccc2nc(-c3ccccc3)oc2c1)N1C[C@@H](F)C[C@…   P13612   P05556

      [1974 rows x 3 columns]
```

Save the Data Frames Dictionary

[323]:
```python
directory_path = 'obj'

# Save the dictionary to a file in the specified directory
with open(os.path.join(directory_path, 'data_frames_dictionary.pkl'), 'wb') as␣
↪file:
    pickle.dump(protein_dataframes, file)
```

Save the Generated Data Frame as CSV Files

[50]:
```python
out_dir = 'unique UniProt csv files'
```

[51]:
```python
for protein, df in protein_dataframes.items():
    try:
        # Generate csv file name with the desired format
        file_name = f'{protein}.csv'

        # Specify full path
        out_path = os.path.join(out_dir, file_name)

        # Save current data frame as csv file
        df.to_csv(out_path, index=False)

        PRINT(f'Saved data frame for {protein} as {file_name}')

    except Exception as e:
        PRINT(f'Error!\nVerify path name and the data')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Saved data frame for P13612 as P13612.csv
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Saved data frame for P05556 as P05556.csv
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Saved data frame for P05106 as P05106.csv
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Saved data frame for P05107 as P05107.csv
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
Saved data frame for P08648 as P08648.csv
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Saved data frame for P17301 as P17301.csv
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.1.3 Visualize Distributions for each Data Frame

[60]:
```
PRINT(f'We have {len(protein_dataframes.items())} data frames to visualize␣
↪information about their data distributions')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
We have 6 data frames to visualize information about their data distributions
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[62]:
```
PRINT(f'UniProt_ids -> {unique_proteins}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
UniProt_ids -> ['P13612' 'P05556' 'P05106' 'P05107' 'P08648' 'P17301']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Helper Functions

Helper One-Hot-Encoding Function

[215]:
```
def one_hot_encoding(df):

    df_encoded = pd.get_dummies(df[['UniProt1', 'UniProt2']], prefix='',␣
↪prefix_sep='').astype(int)
    df_encoded = pd.concat([df[['SMILES']], df_encoded], axis=1)
    return df_encoded
```

Helper Visualization Function

[203]:
```
def visualize_dist(df, target_prot)-> None:
    # Melt the DataFrame to long format for Seaborn countplot
    df_melted = df.melt(var_name='Protein', value_name='Interaction Status')

    # Set the size of the plot
    sns.set(rc={'figure.figsize':(12, 8)})

    sns.set_context("notebook", rc={"lines.linewidth": 2.5})
    # Create a grouped count plot
    sns.countplot(x='Protein', hue='Interaction Status', palette=["lightgrey",␣
↪"skyblue"], data=df_melted)

    # Add labels and title
    plt.xlabel('Protein')
    plt.ylabel('Count')
    plt.title(f'PPI with -> {target_prot}')
```

```
    sns.despine()
    sns.set_theme(style="whitegrid")
    sns.despine(offset=10, trim=True)
    sns.set_context("notebook")
    plt.show()
```

Helper Column Filter Function

```
[180]: def filter_proteins_list(df, columns_to_remove):

           filtered_columns = [col for col in df.columns if col not in␣
         ↪columns_to_remove]
           filtered_columns_list = list(filtered_columns)
           return filtered_columns
```

First Data Frame

```
[228]: first_df = protein_dataframes[unique_proteins[0]]

       first_df.head(2)
```

```
[228]:                                               SMILES UniProt1 UniProt2
       0  COc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2Cl)C(=O)…   P13612   P26010
       1  Cc1ccccc1NC(=O)Nc1ccc(CC(=O)N2C[C@@H](F)C[C@H]…   P13612   P26010
```

Visualize Distribution

```
[234]: first_df_encoded = one_hot_encoding(first_df)
```

```
[235]: print(first_df_encoded.columns)
```

```
Index(['SMILES', 'P13612', 'P05556', 'P26010'], dtype='object')
```

```
[236]: first_df_encoded.head(3)
```

```
[236]:                                               SMILES  P13612  P05556  P26010
       0  COc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2Cl)C(=O)…       1       0       1
       1  Cc1ccccc1NC(=O)Nc1ccc(CC(=O)N2C[C@@H](F)C[C@H]…       1       0       1
       2  CN(C)Cc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2Cl)C…       1       0       1
```

```
[237]: filtered_columns = filter_proteins_list(first_df_encoded, columns_to_remove =␣
         ↪['SMILES', 'P13612'])
       PRINT(f'Filtered columns -> {filtered_columns}')
```

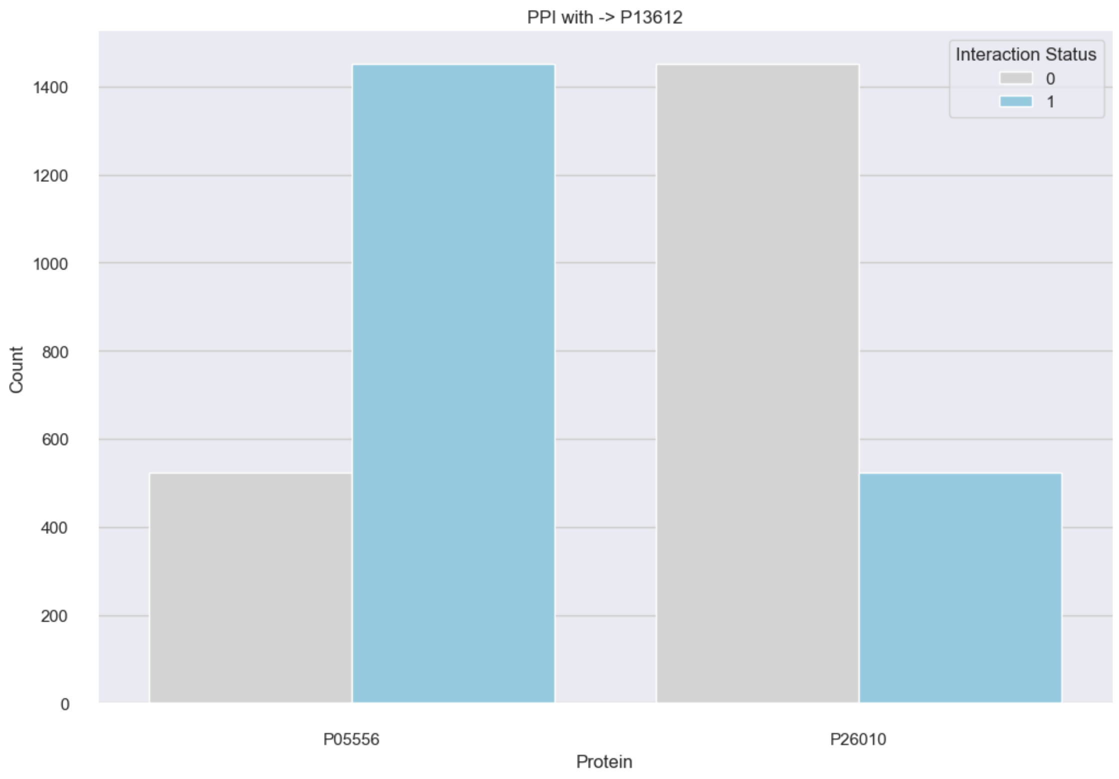
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Filtered columns -> ['P05556', 'P26010']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[238]: temp_df_1 = first_df_encoded[filtered_columns]
```

```
temp_df_1.head(2)
```

```
[238]:    P05556  P26010
       0       0       1
       1       0       1
```

```
[268]: visualize_dist(temp_df_1, unique_proteins[0])
```



As we can see from the histogram, `P05556` appears much more than `P26010` in the PPI with UniProt traget `P13612`

Explore the First Data Frame

```
[251]: PRINT(f'The size of the data frame is -> {len(first_df)}')
print(f'Number of times P05556 appears -> {len(first_df[first_df["UniProt2"] ==␣
 ↪"P05556"])}')
print(f'Number of times P26010 appears -> {len(first_df[first_df["UniProt2"] ==␣
 ↪"P26010"])}')
print(f'Size check -> {({len(first_df)} == {(len(first_df[first_df["UniProt2"]␣
 ↪== "P05556"]) + len(first_df[first_df["UniProt2"] == "P26010"]))})}')

PRINT('Done.')
```

~~~~~~
~~~~~~

```
The size of the data frame is -> 1974
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of times P05556 appears -> 1452
Number of times P26010 appears -> 522
Size check -> True
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Second Data Frame

[222]:
```
second_df = protein_dataframes[unique_proteins[1]]

second_df.head(2)
```

[222]:
```
                                              SMILES UniProt1 UniProt2
0  CC(C)(C)c1cc(Br)cc([C@H](CC(=O)O)NC(=O)CNC(=O)…   P05556   O75578
1  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…   P05556   P56199
```

Visualize Distribution

[223]:
```
second_df_encoded = one_hot_encoding(second_df)
```

[252]:
```
second_df_encoded.columns
```

[252]:
```
Index(['SMILES', 'P05556', 'O75578', 'P05106', 'P06756', 'P08648', 'P13612',
       'P17301', 'P23229', 'P56199', 'Q13797'],
      dtype='object')
```

[224]:
```
second_df_encoded.head(3)
```

[224]:
```
                                              SMILES  P05556  O75578  P05106  \
0  CC(C)(C)c1cc(Br)cc([C@H](CC(=O)O)NC(=O)CNC(=O)…       1       1       0
1  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…       1       0       0
2  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…       1       0       0

   P06756  P08648  P13612  P17301  P23229  P56199  Q13797
0       0       0       0       0       0       0       0
1       0       0       0       0       0       1       0
2       0       0       0       0       0       1       0
```

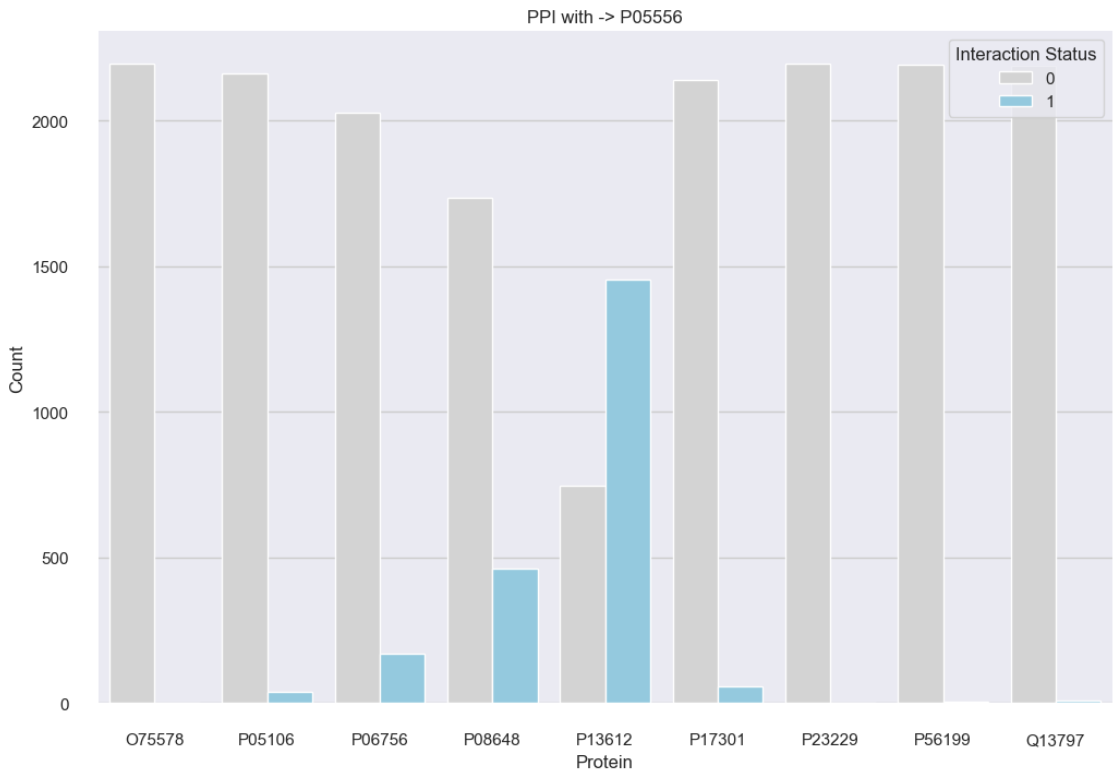
[225]:
```
filtered_columns = filter_proteins_list(second_df_encoded, columns_to_remove =␣
 ↪['SMILES', 'P05556'])
PRINT(f'Filtered columns -> {filtered_columns}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Filtered columns -> ['O75578', 'P05106', 'P06756', 'P08648', 'P13612', 'P17301',
'P23229', 'P56199', 'Q13797']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[226]: temp_df_2 = second_df_encoded[filtered_columns]

       temp_df_2.head(5)
```

```
[226]:    O75578  P05106  P06756  P08648  P13612  P17301  P23229  P56199  Q13797
       0       1       0       0       0       0       0       0       0       0
       1       0       0       0       0       0       0       0       1       0
       2       0       0       0       0       0       0       0       1       0
       3       0       0       0       0       0       0       0       1       0
       4       0       0       0       0       0       0       0       1       0
```

```
[267]: visualize_dist(temp_df_2, unique_proteins[1])
```



In the plot above, we observe that certain proteins, such as O75578 and P23229, have minimal occurrences in the PPI with P05106. In contrast, proteins like P13612 exhibit frequent appearances in the PPI with 'P05106.

Explore the Second Data Frame

```
[250]: PRINT(f'The size of the data frame is -> {len(second_df)}')
       print(f'Number of time O75578 appears -> {len(second_df[second_df["UniProt2"]␣
        ↪== "O75578"])}')
```

```
print(f'Number of time P23229 appears -> {len(second_df[second_df["UniProt2"]␣
↪== "P23229"])}')
print(f'Number of time P56199 appears -> {len(second_df[second_df["UniProt2"]␣
↪== "P56199"])}')
print(f'Number of time Q13797 appears -> {len(second_df[second_df["UniProt2"]␣
↪== "Q13797"])}')
print(f'Number of time P17301 appears -> {len(second_df[second_df["UniProt2"]␣
↪== "P17301"])}')
print(f'Number of time P05106 appears -> {len(second_df[second_df["UniProt2"]␣
↪== "P05106"])}')
print(f'Number of time P06756 appears -> {len(second_df[second_df["UniProt2"]␣
↪== "P06756"])}')
print(f'Number of time P08648 appears -> {len(second_df[second_df["UniProt2"]␣
↪== "P08648"])}')
print(f'Number of time P13612 appears -> {len(second_df[second_df["UniProt2"]␣
↪== "P13612"])}')

PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The size of the data frame is -> 2197
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of time O75578 appears -> 1
Number of time P23229 appears -> 1
Number of time P56199 appears -> 6
Number of time Q13797 appears -> 10
Number of time P17301 appears -> 57
Number of time P05106 appears -> 37
Number of time P06756 appears -> 170
Number of time P08648 appears -> 463
Number of time P13612 appears -> 1452
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Third Data Frame

[263]:
```
third_df = protein_dataframes[unique_proteins[2]]

third_df.head(2)
```

[263]:
```
                                              SMILES UniProt1 UniProt2
0  CC(C)Oc1ccc(C(CC(=O)O)NC(=O)CCC(=O)Nc2ccc3c(c2…   P05106   P26006
1  COc1ccc(C(CC(=O)O)NC(=O)c2cccc(C(=O)Nc3ccc4c(c…   P05106   P26006
```

Visualize Distribution

[264]:
```
third_df_encoded = one_hot_encoding(third_df)
```

[265]:
```
third_df_encoded.columns
```

[265]:
```
Index(['SMILES', 'P05106', 'P05556', 'P06756', 'P08514', 'P17301', 'P26006'],
dtype='object')
```

[261]:
```
third_df_encoded.head(3)
```

[261]:
```
                                              SMILES  P05106  P05556  P06756  \
0  CC(C)Oc1ccc(C(CC(=O)O)NC(=O)CCC(=O)Nc2ccc3c(c2…       1       0       0
1  COc1ccc(C(CC(=O)O)NC(=O)c2cccc(C(=O)Nc3ccc4c(c…       1       0       0
2  COc1ccc(C(CC(=O)O)NC(=O)CCC(=O)Nc2ccc3c(c2)CNC…       1       0       0

   P08514  P17301  P26006
0       0       0       1
1       0       0       1
2       0       0       1
```

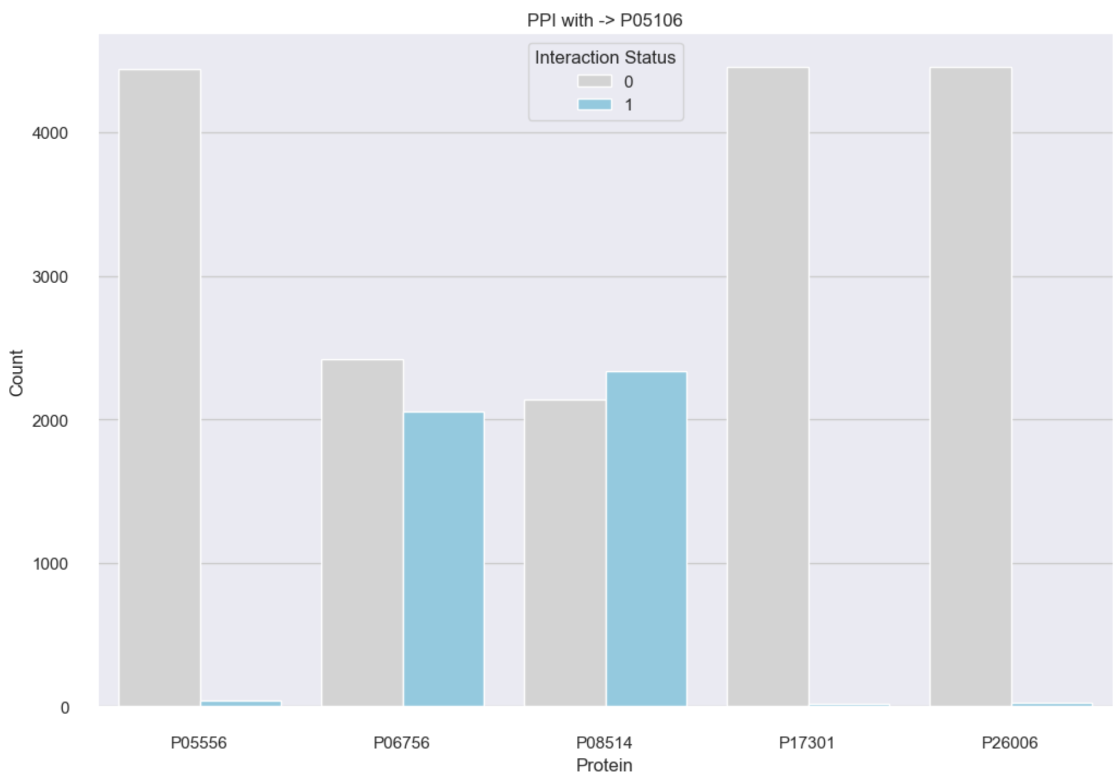
[262]:
```
filtered_columns = filter_proteins_list(third_df_encoded, columns_to_remove =␣
 ↪['SMILES', 'P05106'])
PRINT(f'Filtered columns -> {filtered_columns}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Filtered columns -> ['P05556', 'P06756', 'P08514', 'P17301', 'P26006']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[269]:
```
temp_df_3 = third_df_encoded[filtered_columns]

temp_df_3.head(5)
```

[269]:
```
   P05556  P06756  P08514  P17301  P26006
0       0       0       0       0       1
1       0       0       0       0       1
2       0       0       0       0       1
3       0       0       0       0       1
4       0       0       0       0       1
```

[270]:
```
visualize_dist(temp_df_3, unique_proteins[2])
```



Explore the Third Data Frame

[272]:
```
PRINT(f'The size of the data frame is -> {len(third_df)}')
print(f'Number of time P17301 appears -> {len(third_df[third_df["UniProt2"] ==␣
  ↪"P17301"])}')
print(f'Number of time P05556 appears -> {len(third_df[third_df["UniProt2"] ==␣
  ↪"P05556"])}')
print(f'Number of time P26006 appears -> {len(third_df[third_df["UniProt2"] ==␣
  ↪"P26006"])}')
print(f'Number of time P06756 appears -> {len(third_df[third_df["UniProt2"] ==␣
  ↪"P06756"])}')
print(f'Number of time P08514 appears -> {len(third_df[third_df["UniProt2"] ==␣
  ↪"P08514"])}')

PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The size of the data frame is -> 4478
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of time P17301 appears -> 20
Number of time P05556 appears -> 37
Number of time P26006 appears -> 25
Number of time P06756 appears -> 2058
```

```
Number of time P08514 appears -> 2338
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Fourth Data Frame

```
[282]: fourth_df = protein_dataframes[unique_proteins[3]]

       fourth_df.head(2)
```

```
[282]:                                              SMILES UniProt1 UniProt2
       0  COC(=O)CN1C(=O)S/C(=C\c2ccc(-c3ccc(C(=O)O)cc3)…   P05107   P11215
       1                    Cc1ccc(/C=C2\SC(=O)N(C)C2=O)o1   P05107   P11215
```

Visualize Distribution

```
[283]: fourth_df_encoded = one_hot_encoding(fourth_df)
```

```
[284]: fourth_df_encoded.columns
```

```
[284]: Index(['SMILES', 'P05107', 'P11215', 'P20701'], dtype='object')
```

```
[285]: fourth_df_encoded.head(3)
```

```
[285]:                                              SMILES  P05107  P11215  P20701
       0  COC(=O)CN1C(=O)S/C(=C\c2ccc(-c3ccc(C(=O)O)cc3)…       1       1       0
       1                    Cc1ccc(/C=C2\SC(=O)N(C)C2=O)o1       1       1       0
       2     CCN1/C(=C/C=C/c2sc3ccccc3[n+]2CC)Sc2ccccc21.[I-]       1       1       0
```

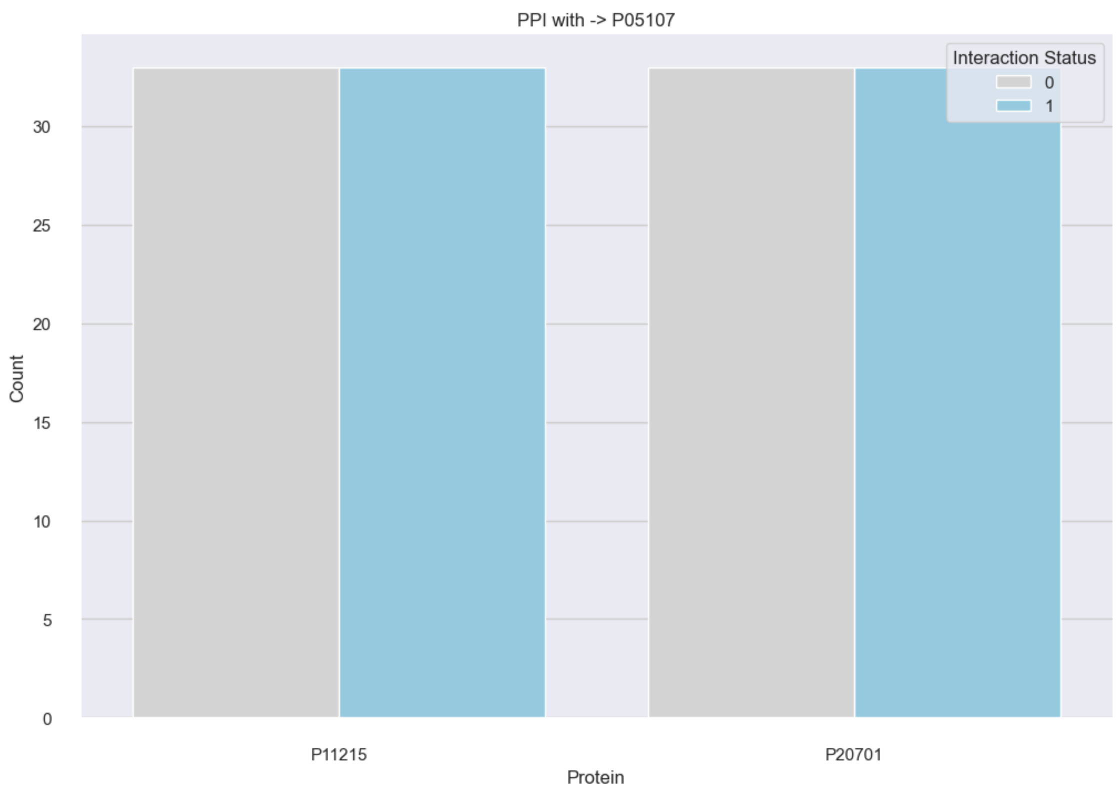
```
[286]: filtered_columns = filter_proteins_list(fourth_df_encoded,␣
        ↪columns_to_remove=['SMILES', 'P05107'])
       PRINT(f'Filtered columns -> {filtered_columns}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Filtered columns -> ['P11215', 'P20701']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[290]: temp_df_4 = fourth_df_encoded[filtered_columns]

       temp_df_4.head(2)
```

```
[290]:    P11215  P20701
       0       1       0
       1       1       0
```

```
[291]: visualize_dist(temp_df_4, unique_proteins[3])
```



The data above is quite interesting, indicating that both proteins appear the same number of times in the PPI with `P05107`.

Explore the Fourth Data Frame

[294]:
```
PRINT(f'The size of the data frame is -> {len(fourth_df)}')
print(f'Number of time P11215 appears -> {len(fourth_df[fourth_df["UniProt2"]␣
  ↪== "P11215"])}')
print(f'Number of time P20701 appears -> {len(fourth_df[fourth_df["UniProt2"]␣
  ↪== "P20701"])}')

PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The size of the data frame is -> 66
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of time P11215 appears -> 33
Number of time P20701 appears -> 33
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Fifth Data Frame

[296]:
```
fifth_df = protein_dataframes[unique_proteins[4]]

fifth_df.head(2)
```

[296]:
```
                                              SMILES UniProt1 UniProt2
0  O=C(N[C@@H](Cc1cccc(OCCCCNc2ccccn2)c1)C(=O)O)c…   P08648   P05556
1  CC(C)[C@@H]1NC(=O)[C@@H](Cc2c[nH]c3c(-c4ccc(C(…   P08648   P05556
```

Visualize Distribution

[297]:
```
fifth_df_encoded = one_hot_encoding(fifth_df)
```

[298]:
```
fifth_df_encoded.columns
```

[298]:
```
Index(['SMILES', 'P08648', 'P05556', 'P06756'], dtype='object')
```

[302]:
```
fifth_df_encoded.head(3)
```

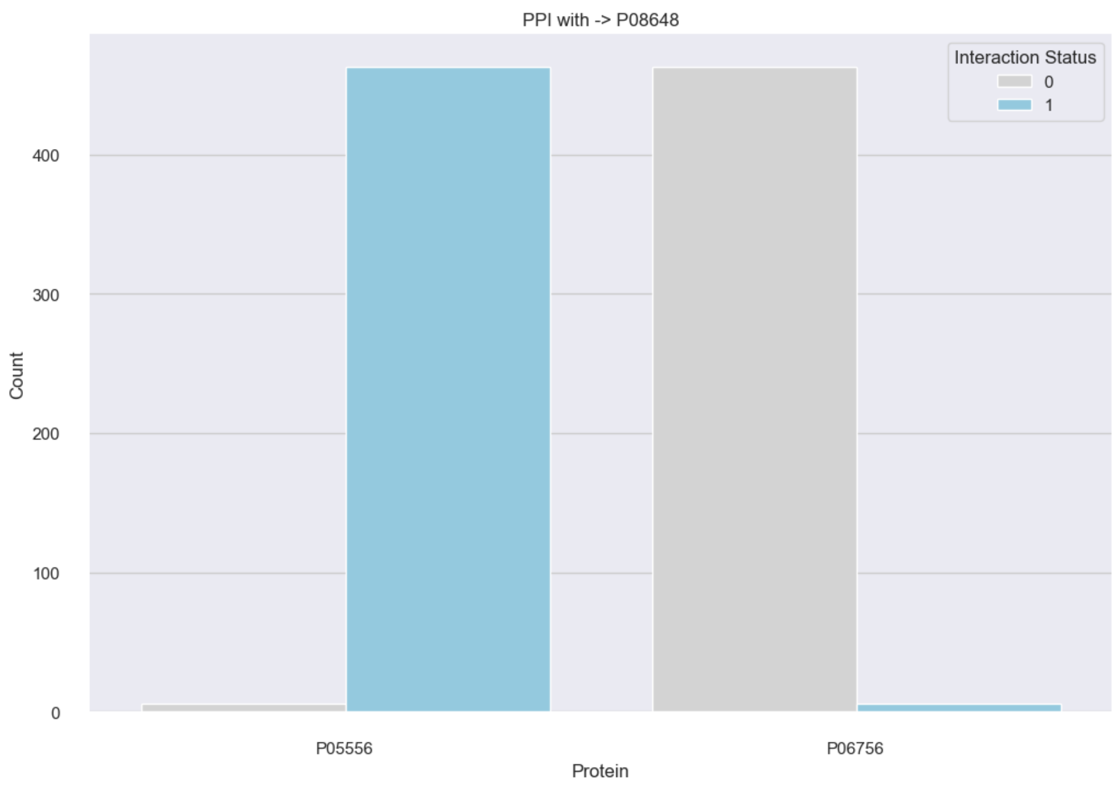
[302]:
```
                                              SMILES  P08648  P05556  P06756
0  O=C(N[C@@H](Cc1cccc(OCCCCNc2ccccn2)c1)C(=O)O)c…       1       1       0
1  CC(C)[C@@H]1NC(=O)[C@@H](Cc2c[nH]c3c(-c4ccc(C(…       1       1       0
2  CC(C)[C@@H]1NC(=O)[C@@H](Cc2c[nH]c3c(-c4ccc5cc…       1       1       0
```

[303]:
```
filtered_columns = filter_proteins_list(fifth_df_encoded,␣
 ↪columns_to_remove=['SMILES', 'P08648'])
PRINT(f'Filtered columns -> {filtered_columns}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Filtered columns -> ['P05556', 'P06756']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[305]:
```
temp_df_5 = fifth_df_encoded[filtered_columns]
```

[307]:
```
visualize_dist(temp_df_5, unique_proteins[4])
```



Here, we observe a particularly interesting distribution of the data. The majority of the fifth dataset represents PPI between the target protein with `UniProt = P08648` and `P05556`. Conversely, there are very few interactions involving `P06756`.

Explore the Fifth Data Frame

```
[309]: PRINT(f'The size of the data frame is -> {len(fifth_df)}')
print(f'Number of time P05556 appears -> {len(fifth_df[fifth_df["UniProt2"] ==␣
 ↪"P05556"])}')
print(f'Number of time P0756 appears -> {len(fifth_df[fifth_df["UniProt2"] ==␣
 ↪"P06756"])}')

PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The size of the data frame is -> 469
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of time P05556 appears -> 463
Number of time P0756 appears -> 6
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Sixth Data Frame

[310]:
```
sixth_df = protein_dataframes[unique_proteins[5]]

sixth_df.head(2)
```

[310]:
```
                                              SMILES UniProt1 UniProt2
0  Cc1cccc(Cl)c1C(=O)N[C@@H](Cc1ccc(NC(=O)c2c(Cl)…   P17301   P05556
1  COc1ccc(S(=O)(=O)N2Cc3[nH]c4ccccc4c3CC2C(N)=O)cc1   P17301   P05556
```

Visualize Distribution

[311]:
```
sixth_df_encoded = one_hot_encoding(sixth_df)
```

[312]:
```
sixth_df_encoded.columns
```

[312]:
```
Index(['SMILES', 'P17301', 'P05106', 'P05556'], dtype='object')
```

[313]:
```
sixth_df_encoded.head(3)
```

[313]:
```
                                              SMILES  P17301  P05106  P05556
0  Cc1cccc(Cl)c1C(=O)N[C@@H](Cc1ccc(NC(=O)c2c(Cl)…       1       0       1
1  COc1ccc(S(=O)(=O)N2Cc3[nH]c4ccccc4c3CC2C(N)=O)cc1       1       0       1
2  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…       1       0       1
```

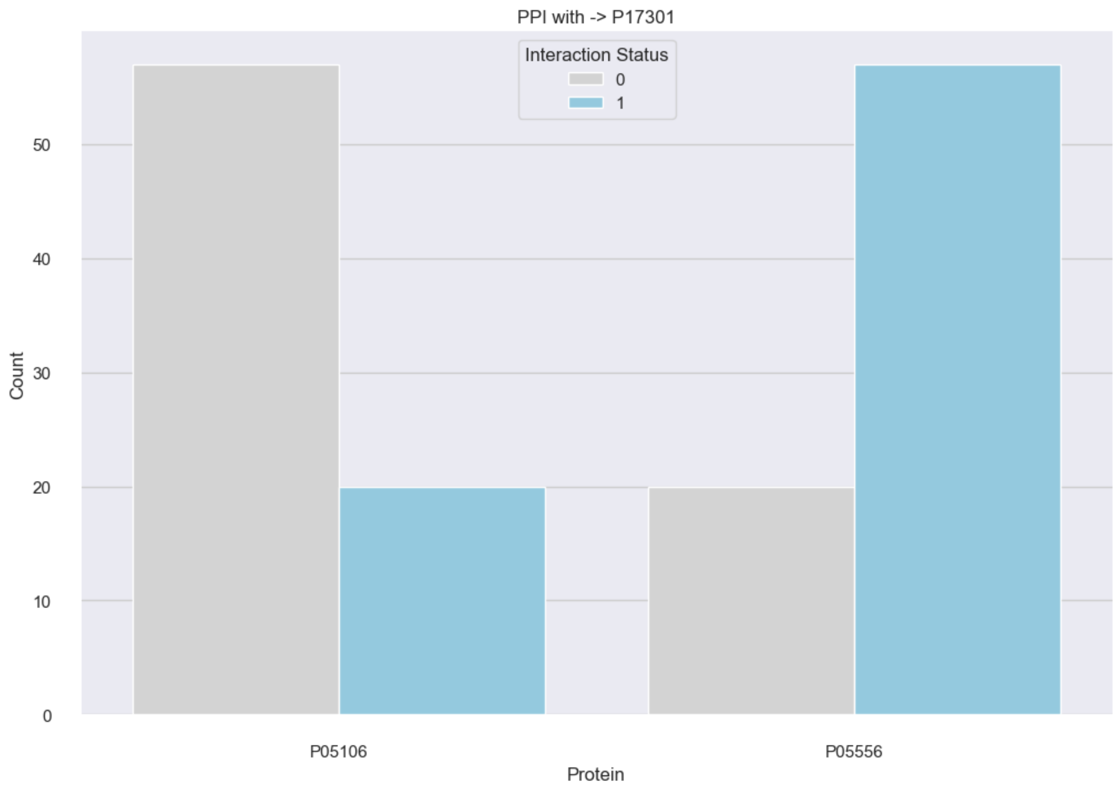
[314]:
```
filtered_columns = filter_proteins_list(sixth_df_encoded,␣
 ↪columns_to_remove=['SMILES', 'P17301'])
PRINT(f'Filtered columns -> {filtered_columns}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Filtered columns -> ['P05106', 'P05556']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[315]:
```
temp_df_6 = sixth_df_encoded[filtered_columns]

temp_df_6.head(2)
```

[315]:
```
   P05106  P05556
0       0       1
1       0       1
```

[316]:
```
visualize_dist(temp_df_6, unique_proteins[5])
```



Explore the Sixth Dath Frame

```
[317]: PRINT(f'The size of the data frame is -> {len(sixth_df)}')
print(f'Number of time P05106 appears -> {len(sixth_df[sixth_df["UniProt2"] ==␣
 ↪"P05106"])}')
print(f'Number of time P05556 appears -> {len(sixth_df[sixth_df["UniProt2"] ==␣
 ↪"P05556"])}')

PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The size of the data frame is -> 77
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Number of time P05106 appears -> 20
Number of time P05556 appears -> 57
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.1.4 Save the Encoded csv Files

[355]:
```
encoded_dir_path = 'one hot encoded csv files for training'

first_df_encoded.to_csv(os.path.join('one hot encoded csv files for training/
  ↪first_df_encoded.csv'), index=False)
second_df_encoded.to_csv(os.path.join('one hot encoded csv files for training/
  ↪second_df_encoded.csv'), index=False)
third_df_encoded.to_csv(os.path.join('one hot encoded csv files for training/
  ↪third_df_encoded.csv'), index=False)
fourth_df_encoded.to_csv(os.path.join('one hot encoded csv files for training/
  ↪fourth_df_encoded.csv'), index=False)
fifth_df_encoded.to_csv(os.path.join('one hot encoded csv files for training/
  ↪fifth_df_encoded.csv'), index=False)
sixth_df_encoded.to_csv(os.path.join('one hot encoded csv files for training/
  ↪sixth_df_encoded.csv'), index=False)

PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.2 Build Classification Models for PPI Prediction

In the upcoming phase, we plan to develop three multiclass classification models to predict Protein-Protein Interactions (PPI) across our six datasets. This entails creating and evaluating six distinct models, one for each dataset. Subsequently, we aim to employ these trained models for predicting PPI on new, unlabeled data.

The models we intend to construct include:

1. Graph Convolution Model using the DeepChem library.
2. Random Forest Multiclass Classifier using the sklearn library.
3. XGBoost Multiclass Classifier.

1.2.1 Import Libraries

[1268]:
```
import pickle  # To load the saved data frames dictionary

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import (
    roc_curve,
    auc,
    roc_auc_score,
    make_scorer,
    accuracy_score,
```

```
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix,
    classification_report,
)
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import label_binarize
from sklearn.utils.class_weight import compute_sample_weight,␣
 ↪compute_class_weight
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪StratifiedKFold

from joblib import dump, load  # For saving & loading tained models

from rdkit import Chem
from rdkit.Chem import AllChem, PandasTools, Descriptors, rdmolops

import xgboost as xgb

import deepchem as dc
from deepchem.feat import RDKitDescriptors
from deepchem.models import GraphConvModel
from deepchem.hyper import GridHyperparamOpt, HyperparamOpt
from deepchem.splits.splitters import RandomGroupSplitter
from deepchem.trans import undo_transforms
from deepchem.trans.transformers import BalancingTransformer
```

1.2.2 Graph Convolution Model

Hyperparameter Tuning for the Model

```
[4]: def gc_model_builder(**model_params,):
        """
        Helper function that constructs and configures a GraphConvModel for the PPI␣
     ↪prediction task.
        This function is intended to be used to provide the necessary model for␣
     ↪hyperparameter tuning
        with the `GridHyperparamOpt()` object.

        Parameters:
        - learning_rate (float): The learning rate for the optimizer.
        - dropout (float): Dropout rate to prevent overfitting.
        - batch_normalize (bool): Whether to apply batch normalization.
        - n_classes (int): Number of classes for classification.

        Returns:
```

```
    - GraphConvModel: Configured instance of GraphConvModel for PPI prediction␣
↪tas.
    """

    learning_rate = model_params['learning_rate']
    dropout = model_params['dropout']
    batch_normalize = model_params['batch_normalize']
    n_classes=model_params['n_classes']

    return GraphConvModel(n_tasks=1,
                          dropout=dropout,
                          mode='classification',
                          batch_normalize=batch_normalize,
                          n_classes=n_classes,
                          learning_rate=learning_rate
                          )
```

[225]:
```
def execute_hyperparameter_tuning_for_graph_conv(csv_data, df, params):
    """
    Perform hyperparameter tuning for a Graph Convolutional Model using a grid␣
↪search approach.

    Parameters:
    - csv_data (str or pd.DataFrame): Path to a CSV file containing molecular␣
↪data for `deepchem.data.CSVLoader.featurize()` object
    - df (pd.DataFrame): A Pandas DataFrame containing the data set for the␣
↪model.
    - params (dict): Dictionary of hyperparameters to be tuned.

    Returns:
    - list: A list containing the best hyperparameters and detailed results of␣
↪the hyperparameter search.
    """
    tasks = ['NumericUniProtTargetLabels']
    featurizer = dc.feat.ConvMolFeaturizer()
    loader = dc.data.CSVLoader(tasks=tasks,
                               smiles_field='SMILES',
                               featurizer=featurizer)

    #splitter = dc.splits.RandomSplitter()
    splitter = dc.splits.RandomStratifiedSplitter()
    mean_roc_auc_metric = dc.metrics.Metric(metric=dc.metrics.roc_auc_score,␣
↪task_averager=np.mean, mode='classification', n_tasks=1)

    dataset = loader.featurize(csv_data)
    res = splitter.train_valid_test_split(dataset,frac_train=0.6, frac_valid=0.
↪2, frac_test=0.2)
    train_dataset, valid_dataset, test_dataset = res

    # Create a hyperparameter optimization object
    opt = GridHyperparamOpt(gc_model_builder)
    best_model, best_hyperparams, all_results = opt.hyperparam_search(params,␣
↪train_dataset, valid_dataset, mean_roc_auc_metric)

    return [best_hyperparams, all_results]
```

[1269]:
```
def generate_graph_conv_model(dropout, batch_normalize, n_classes,␣
↪learning_rate, model_dir):
    """
    Generate a Graph Convolutional Neural Network (GraphConvModel) for␣
↪classification tasks.

    Parameters:
    - dropout (float): Dropout rate to apply in the model.
    - batch_normalize (bool): Whether to apply batch normalization.
    - n_classes (int): Number of classes for classification.
    - learning_rate (float): Learning rate for model training.
    - model_dir (str): Directory to save the trained model.

    Returns:
    - model (GraphConvModel): The configured GraphConvModel for classification.
    """
    batch_size = 64
    model = GraphConvModel(n_tasks=1,
                           dropout=dropout,
                           batch_size=batch_size,
                           batch_normalize=batch_normalize,
                           mode='classification',
                           model_dir=model_dir,
                           n_classes=n_classes,
                           learning_rate=learning_rate)

    return model
```

[364]:
```
def GenerateBoxplotForModelPreformaceVisualization(UniProt, cv_folds,␣
↪training_score_list, validation_score_list) ->None :
    """
    Generate a boxplot to visualize the performance of a model on training and␣
↪validation sets.

    Parameters:
    - UniProt (str): The UniProt ID.
    - cv_folds (int): Number of cross-validation folds.
    - training_score_list (list): List of training scores for each fold.
    - validation_score_list (list): List of validation scores for each fold.

    Returns:
    - None
    """

    data = {
        'Group': ["Training"] * cv_folds + ["Validation"] * cv_folds,
        'Score': training_score_list + validation_score_list
        }

    sns.boxplot(x="Group", y="Score", data=data)

    plt.title(label=f"{UniProt} Mean-Roc-Auc-Score Boxplot Graph",
              fontsize=15,
              color="blue")

    plt.show()
```

[1270]:
```
def get_class_labels(predicted_probs):
    """
    Extract class labels from predicted probabilities.

    Parameters:
    - predicted_probs (numpy.ndarray): Array containing predicted probabilities␣
 ↪for each class.

    Returns:
    - class_labels (numpy.ndarray): Array containing the class labels␣
 ↪corresponding to the highest probability.
    """
    # Remove the extra dimension
    squeezed_probs = np.squeeze(predicted_probs, axis=1)

    # Get the class labels
    class_labels = np.argmax(squeezed_probs, axis=1)
    return class_labels
```

[307]:
```
def draw_roc_auc_score_plot(true_labels, predicted_probs) -> None:
    """
    Draw ROC-AUC score plot for multiclass classification.

    Parameters:
    - true_labels: True class labels
    - predicted_probs: Predicted probabilities for each class
    Returns:
    - None
    """
    # Binarize the true labels
    true_labels_bin = label_binarize(true_labels,␣
↪classes=list(range(predicted_probs.shape[1])))

    # Compute micro-average ROC curve and ROC area
    fpr_micro, tpr_micro, _ = roc_curve(true_labels_bin.ravel(),␣
↪predicted_probs.ravel())
    roc_auc_micro = auc(fpr_micro, tpr_micro)

    # Compute macro-average ROC curve and ROC area
    roc_auc_macro = roc_auc_score(true_labels_bin, predicted_probs,␣
↪multi_class='ovr')

    print(f'Micro-Averaged Roc-Auc-Score -> {roc_auc_micro:.4f}')
    print(f'Macro-Averaged Roc-Auc-Score -> {roc_auc_macro:.4f}')

    # Plot micro-average ROC curve
    plt.figure(figsize=(10, 6))
    plt.plot(fpr_micro, tpr_micro, color='darkorange', lw=2,␣
↪label=f'Micro-Averaged ROC curve (AUC = {roc_auc_micro:.4f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

    plt.title(label="Roc-Auc-Score Graph (Micro-Averaged)", fontsize=15,␣
↪color="blue")
    plt.xlabel('False Positive Rate')
    plt.legend()
    plt.show()

    # Plot macro-average ROC curve (if needed)
    plt.figure(figsize=(10, 6))
    for i in range(predicted_probs.shape[1]):
        fpr, tpr, _ = roc_curve(true_labels_bin[:, i], predicted_probs[:, i])
        plt.plot(fpr, tpr, lw=2, label=f'Class {i} ROC curve')

    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.title(label="Roc-Auc-Score Graph (Macro-Averaged)", fontsize=15,␣
↪color="blue")
    plt.xlabel('False Positive Rate')
    plt.legend()
    plt.show()
```

1.2.3 Random Forest & XGBoost Multiclass Classifiers Models

Next, our objective is to construct Random Forest and XGBoost multiclass classification models. However, before delving into model development, we recognize the need for additional features to enhance performance. The Graph Convolutional Model from the DeepChem library, employed in our previous model, automatically generates features from the molecular SMILES values during training. Consequently, we were able to feed only two columns, namely SMILES and NumericUniProtTargetLabels, to the model.

In contrast, Random Forest and XGBoost do not generate features during training. To address this, we will utilize RDKitDescriptors & Morgan Fingerprints to generate additional features. These additional features will provide valuable information for our models to learn from, potentially improving their overall performance

Generate Fetures using RDKitDescriptors ####e.

[10]:
```
def calculate_descriptors(smiles):
    """
    Helper function that takes a molecule's SMILES value and generates a list␣
 ↪of the best 8 features
    found to be the most significant for our PPI prediction task.

    Params:
    - smiles (str): Molecule's SMILES value as a string.

    Returns:
    - list: A list of 8 features generated from the molecule's SMILES.
    """
    mol = Chem.MolFromSmiles(smiles)
    if mol is not None:
        descriptors = [
            Descriptors.MolWt(mol),
            Descriptors.NumValenceElectrons(mol),
            Descriptors.TPSA(mol),
            Descriptors.MolLogP(mol),
            Descriptors.NumHeteroatoms(mol),
            Descriptors.NumRotatableBonds(mol),
            Descriptors.HeavyAtomCount(mol),
            Descriptors.FractionCSP3(mol)
        ]
        return descriptors
    else:
        return [None] * 8  # Return None for each descriptor if SMILES cannot␣
 ↪be parsed
```

[11]:
```
def GenerateFeaturesByMoleculeSMILES(df) -> pd.DataFrame:
    """
    Takes a DataFrame containing data for a PPI prediction task and adds␣
 ↪features using the
```

```
    `calculate_descriptors(smiles)` feature augmentation helper function.

    Params:
    - df (pd.DataFrame): DataFrame containing data for the task.

    Returns:
    - pd (pd.DataFrame): The same DataFrame after adding the new features.
    """
    df_ = df.copy()
    # Apply the `calculate_descriptors` method in order to generate 8 new␣
↪features for df
    df_['MolecularDescriptors'] = df_['SMILES'].apply(calculate_descriptors)

    # Transfer the array at each row under the 'MolecularDescriptors' column␣
↪into column with their corresponding names & drop the colunn
    df_[['MolWt', 'NumValenceElectrons', 'TPSA', 'MolLogP', 'NumHeteroatoms',␣
↪'NumRotatableBonds', 'HeavyAtomCount', 'FractionCSP3']] = pd.
↪DataFrame(df_['MolecularDescriptors'].tolist(), index=df_.index)
    df_.drop(columns=['MolecularDescriptors'], axis=1, inplace=True)

    # Reorder the columns names so that the label column will be the last␣
↪column in df
    df_ = df_[['SMILES', 'MolWt', 'NumValenceElectrons', 'TPSA', 'MolLogP',␣
↪'NumHeteroatoms', 'NumRotatableBonds', 'HeavyAtomCount', 'FractionCSP3',␣
↪'NumericUniProtTargetLabels']]

    return df_
```

Generate Features using Morgan Fingerprints

```
[992]: def GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df, size, radius) -> pd.
↪DataFrame:
    """
    Generate Morgan fingerprints features for molecules based on their SMILES␣
↪representation.

    Parameters:
    - df (pd.DataFrame): DataFrame containing data for the task.
    - size (int): Size of the circular fingerprint (number of bits).
    - radius (int): Radius parameter for the circular fingerprint.

    Returns:
    - pd.DataFrame: DataFrame with Morgan fingerprints features added.
    """

    # Define the CircularFingerprint featurizer to generate Morgan Fingerprints␣
↪features
```

```
    featurizer = dc.feat.CircularFingerprint(size=size, radius=radius)

    # Convert SMILES to features using the featurizer
    X = [featurizer.featurize(smiles) for smiles in df['SMILES']]
    X_flat = [x.flatten() for x in X]
    feature_columns = [f'Feature_{i}' for i in range(len(X_flat[0]))]
    df_features = pd.DataFrame(X_flat, columns=feature_columns)

    # Combine the features with the original dataframe
    df_combined = pd.concat([df, df_features], axis=1)

    df_with_morgan_fingerprints_features = df_combined

    return df_with_morgan_fingerprints_features
```

Buildint Random Forest Multiclass Classifier Model

[13]:
```
def GenerateRandomForestModel(df, weight_dict):
    """
    Takes data frame with columns ['SMILES', ... molecule fetures ...,␣
↪'NumericUniProtTargetLabels'], traing and evaluate Random Forest Classifier
    model after choosing the best hyperparameters by `GrudSearchCV`. The␣
↪function also takes `weight_dict`, which is dictionary of weights assigned
    for each class in case of imbalanced data, or 'balanced' if the data is␣
↪balanced.

    Params:
    df - data frame
    weight_dict - dictionary of weight, e.g., {0:1, 1:1.8, 2:1, 3:1.3}. In case␣
↪the data balanced, pass 'balanced' instead.

    Return:
    tuple - (best_rf_model, model_preformance_dictionary)
    """

    # Drop SMILES' and labels columns
    X = df.drop(['SMILES', 'NumericUniProtTargetLabels'], axis=1)
    y = df['NumericUniProtTargetLabels']

    # Split the dataset into training and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

    # Generate RF model for hyperparameter tuning phase
    rf_model = RandomForestClassifier(class_weight=weight_dict, random_state=42)

    # Use StratifiedKFold for cross-validation
    stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True,␣
↪random_state=42)

    # Define a parameter distribution
    param_grid = {
        'n_estimators': [50, 100, 150, 200],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10, 15],
        'min_samples_leaf': [1, 2, 4, 8]
    }

    # Use a custom scoring function (weighted F1) for GridSearchCV
    scoring = make_scorer(f1_score, average='weighted')

    # Perform GridSearchCV with StratifiedKFold & get the best hyperparameters
    grid_search = GridSearchCV(rf_model, param_grid, cv=stratified_kfold,␣
↪scoring=scoring)
    grid_search.fit(X_train, y_train)

    best_params = grid_search.best_params_

    # Create a Random Forest classifier with the best hyperparameters
    best_rf_model = RandomForestClassifier(class_weight=weight_dict,␣
↪random_state=42, **best_params)

    # Train the model on the entire training set
    best_rf_model.fit(X_train, y_train)

    # Make predictions on the test set
    y_test_pred = best_rf_model.predict(X_test)

    # Evaluate the model on the test set
    accuracy_test = accuracy_score(y_test, y_test_pred)
    precision_test = precision_score(y_test, y_test_pred, average='weighted')
    recall_test = recall_score(y_test, y_test_pred, average='weighted')
    f1_test = f1_score(y_test, y_test_pred, average='weighted')
    conf_matrix_test = confusion_matrix(y_test, y_test_pred)

    # Print classification report
    print('Classification Report:')
    print(classification_report(y_test, y_test_pred))

    # Return the trained model and evaluation metrics as tuple
    return (best_rf_model, {
        'accuracy': accuracy_test,
        'precision': precision_test,
        'recall': recall_test,
        'f1_score': f1_test,
        'confusion_matrix': conf_matrix_test.tolist()
    })
```

Buildint XGBoost Multiclass Classifier Model

```
[609]: def GenerateXGBoostModel(df, weight_dict):
           """
           Takes data frame with columns ['SMILES', ... molecule fetures ...,␣
        ↪'NumericUniProtTargetLabels'], traing and evaluate XGBoost model after
           choosing the best hyperparameters by `GrudSearchCV`. The function also␣
        ↪takes `weight_dict`, which is dictionary of weights assigned
           for each class in case of imbalanced data, or 'balanced' if the data is␣
        ↪balanced.

           Params:
           df - data frame
           weight_dict - dictionary of weight, e.g., {0:1, 1:1.8, 2:1, 3:1.3}. In case␣
        ↪the data balanced, pass 'balanced' instead.

           Return:
           tuple - (best_xbg_model, model_preformance_dictionary)
           """
           # Drop 'SMILES' and labels columns
           X = df.drop(['SMILES', 'NumericUniProtTargetLabels'], axis=1)
           y = df['NumericUniProtTargetLabels']

           # Split the dataset into training and test sets
           X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
        ↪random_state=42)

           # Generate XGB model for hyperparameter tuning phase
           xgb_model = xgb.XGBClassifier(objective='multi:softmax',␣
        ↪num_class=len(set(y_train)), random_state=42)

           # Use StratifiedKFold for cross-validation
           stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True,␣
        ↪random_state=42)

           param_grid = {
               'n_estimators': [50, 100],
               'max_depth': [3, 5],
               'learning_rate': [0.01, 0.1],
               'subsample': [0.8, 1.0],
               'colsample_bytree': [0.8, 1.0],
               'gamma': [0, 0.2],
               'min_child_weight': [1, 5],
        'reg_alpha': [0, 0.5],
        'reg_lambda': [0, 0.5],
    }

    # Use a custom scoring function (weighted F1) for GridSearchCV
    scoring = make_scorer(f1_score, average='weighted')

    # Perform GridSearchCV with StratifiedKFold & extract the best␣
↪hyperparameters
    grid_search = GridSearchCV(xgb_model, param_grid, cv=stratified_kfold,␣
↪scoring=scoring)
    grid_search.fit(X_train, y_train)

    best_params = grid_search.best_params_

    # Create an XGBoost classifier with the best hyperparameters
    best_xgb_model = xgb.XGBClassifier(objective='multi:softmax',
                                       num_class=len(set(y_train)),
                                       random_state=42, **best_params)

    # Calculate sample weights for each instance based on class weights
    sample_weights = compute_sample_weight(weight_dict, y_train)

    # Train the model on the entire training set with sample weights
    best_xgb_model.fit(X_train, y_train, sample_weight=sample_weights)

    # Make predictions on the test set
    y_test_pred = best_xgb_model.predict(X_test)

    # Evaluate the model on the test set
    accuracy_test = accuracy_score(y_test, y_test_pred)
    precision_test = precision_score(y_test, y_test_pred, average='weighted')
    recall_test = recall_score(y_test, y_test_pred, average='weighted')
    f1_test = f1_score(y_test, y_test_pred, average='weighted')
    conf_matrix_test = confusion_matrix(y_test, y_test_pred)

    # Print classification report
    print('Classification Report:')
    print(classification_report(y_test, y_test_pred))

    # Return the trained model and evaluation metrics as tuple
    return (best_xgb_model, {
        'accuracy': accuracy_test,
        'precision': precision_test,
        'recall': recall_test,
        'f1_score': f1_test,
        'confusion_matrix': conf_matrix_test.tolist()
    })
```

1.3 Build PPI Prediction Model for each Dataset

After constructing three models for our Protein-Protein Interaction (PPI) prediction task, including graph convolution, random forest, and XGBoost multiclass classifiers, the next phase involves developing and training a distinct model for each unique dataset extracted at the beginning of the project.

1.3.1 Construct the Datasets

First we need to load our saved data frames dictionary

[15]:
```
dict_path = 'obj/data_frames_dictionary.pkl'
```

[16]:
```
try:
    with open('obj/data_frames_dictionary.pkl', 'rb') as file:
        df_dict = pickle.load(file)
        PRINT(f'Done.')
except Exception as e:
    PRINT(f'Error in loading the saved data frames dicitonary from obj dir')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[17]:
```
prot_ls = list(df_dict.keys())

PRINT(f'Unique proteins -> {prot_ls}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Unique proteins -> ['P13612', 'P05556', 'P05106', 'P05107', 'P08648', 'P17301']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[18]:
```
csv_dir_path = 'one hot encoded csv files for training'
```

1.3.2 Prepare the Datasets for Model Training

[446]:
```
def generate_df_for_training_(UniProt_str, csv_file_name, one_hot_encoded_csv):
    """
    Generate and prepare a DataFrame for model training.

    Parameters:
    - df (pd.DataFrame): Original DataFrame containing the dataset for training.
    - UniProt_str (str): String identifier for the specific UniProt.
    - csv_file_name (str): Name of the CSV file containing UniProt-specific␣
 ↪dataset.
    - one_hot_encoded_csv (str): Name of the CSV file containing one-hot␣
 ↪encoded labels.
```

```
    Returns:
    - tuple: A tuple containing two DataFrames: one for model training and the␣
↪other with UniProt-specific data.
    """

    # Define the directories for CSV files
    csv_dir = 'unique UniProt csv files'
    csv_dir_ohe = 'one hot encoded csv files for training'

    # Read the UniProt-specific CSV file and the one-hot encoded CSV file
    curr_df = pd.read_csv(os.path.join(csv_dir, csv_file_name))
    ohe_df = pd.read_csv(os.path.join(csv_dir_ohe, one_hot_encoded_csv))

    # Drop unnecessary column and rename the target column
    curr_df.drop('UniProt1', axis=1, inplace=True)
    curr_df = curr_df.rename(columns={'UniProt2':'UniProtTargetLabels'})

    # Extract the list of labels from the one-hot encoded DataFrame
    labels = ohe_df.columns[2:].tolist()

    # Print the UniProt model labels
    PRINT(f'{UniProt_str} model labels -> {labels}')

    # Create a mapping of column names to indices for label encoding
    column_name_to_index = {label: i for i, label in enumerate(labels)}

    # Map the 'labels' column in df to column indices
    curr_df['NumericUniProtTargetLabels'] = curr_df['UniProtTargetLabels'].
↪map(column_name_to_index)

    # Shuffle the rows
    curr_df = curr_df.sample(frac=1, random_state=42).reset_index(drop=True)

    # Create a DataFrame for model training by dropping the original␣
↪'UniProtTargetLabels' column
    df_for_model = curr_df.drop('UniProtTargetLabels', axis=1)

    PRINT(f'Finished generating DataFrames for UniProt -> {UniProt_str}.')

    # Return the tuple of DataFrames & the label mapping dictionary
    return (df_for_model, curr_df, column_name_to_index)
```

1.3.3 Helper Functions for Picking the Best Model

```
[1271]: def visualize_best_models_testing_preformace_(df_for_testing, rf_model,␣
         ↪xbg_model, features_method) -> None:
             """
             Visualize and compare the testing performance of the best models using ROC␣
         ↪curves.

             Parameters:
             - df_for_testing (pd.DataFrame): DataFrame containing the testing data.
             - rf_model (RandomForestClassifier): Trained Random Forest model.
             - xgb_model (XGBClassifier): Trained XGBoost model.
             - features_method (str): The method used to extract features.

             Returns:
             - None: The function displays ROC curves and ROC-AUC scores for the given␣
         ↪models.
             """

             X = df_for_testing.drop(['SMILES', 'NumericUniProtTargetLabels'], axis=1)
             y = df_for_testing['NumericUniProtTargetLabels']

             # Split the dataset into training and test sets
             X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
         ↪random_state=42)

             # List to store model names and their ROC-AUC scores
             model_names = []
             roc_auc_scores = []

             models = [rf_model, xbg_model]

             # Plot ROC curves for each model
             plt.figure(figsize=(10, 6))
             for model in models:
                 # Assuming 'predict_proba' method gives the predicted probabilities
                 predicted_probs = model.predict_proba(X_test)[:, 1]

                 # Calculate ROC curve
                 fpr, tpr, _ = roc_curve(y_test, predicted_probs)
                 roc_auc = auc(fpr, tpr)

                 # Plot ROC curve for each model
                 plt.plot(fpr, tpr, lw=2, label=f'{model.__class__.__name__} (AUC =␣
         ↪{roc_auc:.3f})')

                 # Store model name and ROC-AUC score for comparison
```

```
        model_names.append(model.__class__.__name__)
        roc_auc_scores.append(roc_auc)

    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.title(f"ROC Curves using {features_method}")
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.show()

    # Display ROC-AUC scores for each model
    for name, score in zip(model_names, roc_auc_scores):
        PRINT(f'{name}: ROC-AUC Score -> {score:.3f}')
```

1.3.4 Models for P13612 Protein

[239]:
```
P13612_df_for_training, P13612_df_with_uniprots_col, mapped_label_dict_P13612 =␣
↪generate_df_for_training('P13612', 'P13612.csv', 'first_df_encoded.csv')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
P13612 model labels -> ['P05556', 'P26010']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished generating DataFrames for UniProt -> P13612.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[240]:
```
P13612_df_for_training.head(3)
```

[240]:
```
                                              SMILES  \
0  COc1ccccc1-c1ccc(C[C@H](NC(=O)C2(S(=O)(=O)c3cc…
1  C/C=C/[C@H](CC(=O)O)NC(=O)CN(CCC(C)C)C(=O)Cc1c…
2  CC(C)CCNCc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2C…

   NumericUniProtTargetLabels
0                           0
1                           0
2                           0
```

[65]:
```
P13612_df_with_uniprots_col.head(3)
```

[65]:
```
                                              SMILES UniProtTargetLabels  \
0  COc1ccccc1-c1ccc(C[C@H](NC(=O)C2(S(=O)(=O)c3cc…              P05556
1  C/C=C/[C@H](CC(=O)O)NC(=O)CN(CCC(C)C)C(=O)Cc1c…              P05556
2  CC(C)CCNCc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2C…              P05556

   NumericUniProtTargetLabels
0                           0
1                           0
```

```
2                              0
```

```
[66]: PRINT(f'The mapped labels in ("UniProt": "index_label") format:
       ↪\n\n{mapped_label_dict_P13612}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The mapped labels in ("UniProt": "index_label") format:

{'P05556': 0, 'P26010': 1}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Quit Dataset Analysis

- **Size of the data frame:** 1974

- **Number of occurrences for each protein:**
  - P05556: 1452
  - P26010: 522

As evident from the data, there is an imbalance in the dataset. To tackle this issue, we plan to assign different weights to the classes. This approach aims to encourage the models to account for the imbalances in the data during training.

Random Forest Multiclass Classifier Model using RKDitDescriptors features for P13612

```
[475]: P13612_df_for_training_ =␣
        ↪GenerateFeaturesByMoleculeSMILES(df=P13612_df_for_training)
```

```
[476]: P13612_df_for_training_.head(3)
```

```
[476]:                                              SMILES    MolWt  \
0  COc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2Cl)C(=O)…  409.869
1  Cc1ccccc1NC(=O)Nc1ccc(CC(=O)N2C[C@@H](F)C[C@H]…  539.991
2  CN(C)Cc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2Cl)C…  436.939

   NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  NumRotatableBonds  \
0                  148   75.63  4.44130               6                  7
1                  198  107.97  5.55112              10                  8
2                  160   69.64  4.49430               6                  8

   HeavyAtomCount  FractionCSP3  NumericUniProtTargetLabels
0              29      0.130435                           1
1              38      0.250000                           1
2              31      0.200000                           1
```

```
[478]: weight_dict = 'balanced'

rf_model_tuple_P13612_01 =␣
  ↪GenerateRandomForestModel(df=P13612_df_for_training_,␣
  ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.67      0.74       289
           1       0.41      0.60      0.48       106

    accuracy                           0.66       395
   macro avg       0.61      0.64      0.61       395
weighted avg       0.71      0.66      0.67       395
```

```
[486]: weight_dict = {0:1, 1:1.5}

rf_model_tuple_P13612_02 =␣
  ↪GenerateRandomForestModel(df=P13612_df_for_training_,␣
  ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.91      0.83       289
           1       0.49      0.23      0.31       106

    accuracy                           0.73       395
   macro avg       0.63      0.57      0.57       395
weighted avg       0.69      0.73      0.69       395
```

```
[487]: weight_dict = {0:1, 1:1.75}

rf_model_tuple_P13612_03 =␣
  ↪GenerateRandomForestModel(df=P13612_df_for_training_,␣
  ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.87      0.82       289
           1       0.46      0.30      0.37       106

    accuracy                           0.72       395
   macro avg       0.62      0.59      0.59       395
weighted avg       0.69      0.72      0.70       395
```

```
[496]: weight_dict = {0:1, 1:1.825}

       rf_model_tuple_P13612_03 =␣
        ↪GenerateRandomForestModel(df=P13612_df_for_training_,␣
        ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.86      0.82       289
           1       0.47      0.34      0.40       106

    accuracy                           0.72       395
   macro avg       0.63      0.60      0.61       395
weighted avg       0.70      0.72      0.71       395
```

```
[490]: weight_dict = {0:1, 1:2}

       rf_model_tuple_P13612_04 =␣
        ↪GenerateRandomForestModel(df=P13612_df_for_training_,␣
        ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.84      0.81       289
           1       0.45      0.36      0.40       106

    accuracy                           0.71       395
   macro avg       0.62      0.60      0.60       395
weighted avg       0.69      0.71      0.70       395
```

```
[507]: PRINT(f'The results of the best Random Forest Multiclass Classifier model␣
        ↪for\nUniProt P13612 are:')
       print_dict_meaningful(rf_model_tuple_P13612_03[1])
       PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of the best Random Forest Multiclass Classifier model for
UniProt P13612 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.722
precision: 0.698
recall: 0.722
f1_score: 0.705
```

```
confusion_matrix: [[249, 40], [70, 36]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[500]:
```
best_rf_model_P13612 = rf_model_tuple_P13612_03[0]

best_rf_model_P13612
```

[500]:
```
RandomForestClassifier(class_weight={0: 1, 1: 1.825}, max_depth=10,
                       min_samples_leaf=8, n_estimators=50, random_state=42)
```

Save the Best Random Forest Multicalss Classifier Model for P13612

[558]:
```
rf_model_filename = os.path.join('trained models/Random Forest Multiclass␣
↪Classifier Models', 'rf_model_P13612.joblib')
dump(best_rf_model_P13612, rf_model_filename)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model using RKDitDescriptors features for P13612

[542]:
```
weight_dict = 'balanced'

xgb_model_tuple_P13612_01 = GenerateXGBoostModel(df=P13612_df_for_training_,␣
↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.59      0.69       289
           1       0.38      0.69      0.49       106

    accuracy                           0.62       395
   macro avg       0.61      0.64      0.59       395
weighted avg       0.71      0.62      0.64       395
```

[543]:
```
weight_dict = {0: 1, 1: 1.5}

xgb_model_tuple_P13612_01 = GenerateXGBoostModel(df=P13612_df_for_training_,␣
↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.93      0.84       289
```

```
           1       0.56      0.24      0.33       106

    accuracy                           0.74       395
   macro avg       0.66      0.58      0.59       395
weighted avg       0.71      0.74      0.70       395
```

[544]:
```
weight_dict = {0: 1, 1: 1.75}

xgb_model_tuple_P13612_02 = GenerateXGBoostModel(df=P13612_df_for_training_,␣
 ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.89      0.83       289
           1       0.53      0.35      0.42       106

    accuracy                           0.74       395
   macro avg       0.66      0.62      0.63       395
weighted avg       0.72      0.74      0.72       395
```

[545]:
```
weight_dict = {0: 1, 1: 1.825}

xgb_model_tuple_P13612_03 = GenerateXGBoostModel(df=P13612_df_for_training_,␣
 ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.84      0.82       289
           1       0.48      0.42      0.45       106

    accuracy                           0.72       395
   macro avg       0.64      0.63      0.63       395
weighted avg       0.71      0.72      0.72       395
```

[546]:
```
PRINT(f'The results of the best XGBoost Multiclass Classifier model␣
 ↪for\nUniProt P13612 are:')
print_dict_meaningful(xgb_model_tuple_P13612_02[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of the best XGBoost Multiclass Classifier model for
UniProt P13612 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.742
```

```
precision: 0.718
recall: 0.742
f1_score: 0.723
confusion_matrix: [[256, 33], [69, 37]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[547]:
```
best_xgb_model_P13612 = xgb_model_tuple_P13612_02[0]

best_xgb_model_P13612
```

[547]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0.2, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
              min_child_weight=5, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=50, n_jobs=None, num_class=2,
              num_parallel_tree=None, …)
```

Save the Best Random Forest Multicalss Classifier uaing RKDitDescriptors Model for P13612

[557]:
```
xgb_model_filename = os.path.join('trained models/XGBoost Multiclass Classifier␣
  ↪Models', 'xgb_model_P13612.joblib')
dump(best_xgb_model_P13612, xgb_model_filename)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Random Forest Multiclass Classifier Model for P13612 with added Morgan Fingerprints Features

[67]:
```
P13612_df_for_training__ =␣
  ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P13612_df_for_training,␣
  ↪size=1024, radius=2)
```

[68]:
```
P13612_df_for_training__.head(3)
```

[68]:
```
                                              SMILES  \
0  COc1ccccc1-c1ccc(C[C@H](NC(=O)C2(S(=O)(=O)c3cc…
1  C/C=C/[C@H](CC(=O)O)NC(=O)CN(CCC(C)C)C(=O)Cc1c…
2  CC(C)CCNCc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2C…
```

```
   NumericUniProtTargetLabels  Feature_0  Feature_1  Feature_2  Feature_3  \
0                           0        0.0        1.0        0.0        0.0
1                           0        0.0        1.0        0.0        0.0
2                           0        0.0        1.0        0.0        0.0

   Feature_4  Feature_5  Feature_6  Feature_7  …  Feature_1014  \
0        1.0        0.0        0.0        0.0  …           0.0
1        0.0        0.0        0.0        0.0  …           0.0
2        0.0        0.0        0.0        0.0  …           0.0

   Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
0           0.0           0.0           0.0           0.0           1.0
1           0.0           0.0           0.0           0.0           0.0
2           0.0           0.0           0.0           0.0           0.0

   Feature_1020  Feature_1021  Feature_1022  Feature_1023
0           0.0           0.0           0.0           0.0
1           0.0           0.0           0.0           0.0
2           0.0           0.0           0.0           0.0

[3 rows x 1026 columns]
```

[526]:
```
weight_dict = 'balanced'

rf_model_tuple_P13612_01_ =␣
↪GenerateRandomForestModel(df=P13612_df_for_training__,␣
↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.72      0.82       289
           1       0.54      0.88      0.67       106

    accuracy                           0.76       395
   macro avg       0.74      0.80      0.74       395
weighted avg       0.83      0.76      0.78       395
```

[529]:
```
weight_dict = {0:1, 1:1.5}

rf_model_tuple_P13612_02_ =␣
↪GenerateRandomForestModel(df=P13612_df_for_training__,␣
↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support
```

```
           0       0.86      0.79      0.83       289
           1       0.53      0.65      0.59       106

    accuracy                           0.75       395
   macro avg       0.70      0.72      0.71       395
weighted avg       0.77      0.75      0.76       395
```

[69]:
```
weight_dict = {0:1, 1:1.75}

rf_model_tuple_P13612_03_ =␣
 ↪GenerateRandomForestModel(df=P13612_df_for_training__,␣
 ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.82      0.83       274
           1       0.61      0.62      0.61       121

    accuracy                           0.76       395
   macro avg       0.72      0.72      0.72       395
weighted avg       0.76      0.76      0.76       395
```

[532]:
```
weight_dict = {0:1, 1:1.85}

rf_model_tuple_P13612_04_ =␣
 ↪GenerateRandomForestModel(df=P13612_df_for_training__,␣
 ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.75      0.82       289
           1       0.53      0.77      0.63       106

    accuracy                           0.75       395
   macro avg       0.71      0.76      0.72       395
weighted avg       0.80      0.75      0.77       395
```

[71]:
```
PRINT(f'The results of the best Random Forest Multiclass Classifier␣
 ↪model\nusing Morgan Fingerprints features for UniProt P13612 are:')
print_dict_meaningful(rf_model_tuple_P13612_03_[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
The results of the best Random Forest Multiclass Classifier model
using Morgan Fingerprints features for UniProt P13612 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.762
precision: 0.763
recall: 0.762
f1_score: 0.763
confusion_matrix: [[226, 48], [46, 75]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[72]:
```
best_rf_model_P13612_ = rf_model_tuple_P13612_03_[0]

best_rf_model_P13612_
```

[72]:
```
RandomForestClassifier(class_weight={0: 1, 1: 1.75}, min_samples_leaf=8,
                       n_estimators=150, random_state=42)
```

Save the Best Random Forest Multicalss Classifier Model using Morgan Fingerprint features for P13612

[73]:
```
rf_model_filename = os.path.join('trained models/Random Forest Multiclass␣
 ↪Classifier Models', 'rf_model_P13612_.joblib')
dump(best_rf_model_P13612_, rf_model_filename)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model for P13612 with added Morgan Fingerprints Features

[536]:
```
weight_dict = 'balanced'

xgb_model_tuple_P13612_01_ = GenerateXGBoostModel(df=P13612_df_for_training__,␣
 ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.70      0.79       289
           1       0.50      0.84      0.63       106

    accuracy                           0.73       395
   macro avg       0.71      0.77      0.71       395
weighted avg       0.81      0.73      0.75       395
```

[537]:
```
weight_dict = {0: 1, 1: 1.5}

xgb_model_tuple_P13612_02_ = GenerateXGBoostModel(df=P13612_df_for_training__,␣
↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.76      0.82       289
           1       0.54      0.77      0.64       106

    accuracy                           0.76       395
   macro avg       0.72      0.77      0.73       395
weighted avg       0.80      0.76      0.77       395
```

[70]:
```
weight_dict = {0: 1, 1: 1.75}

xgb_model_tuple_P13612_03_ = GenerateXGBoostModel(df=P13612_df_for_training__,␣
↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.81      0.83       274
           1       0.61      0.67      0.64       121

    accuracy                           0.77       395
   macro avg       0.73      0.74      0.74       395
weighted avg       0.78      0.77      0.77       395
```

[74]:
```
PRINT(f'The results of the best XGBoost Multiclass Classifier model\nusing␣
↪Morgan Fingerprints features for UniProt P13612 are:')
print_dict_meaningful(xgb_model_tuple_P13612_03_[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of the best XGBoost Multiclass Classifier model
using Morgan Fingerprints features for UniProt P13612 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.770
precision: 0.776
recall: 0.770
f1_score: 0.772
confusion_matrix: [[223, 51], [40, 81]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[75]: best_xgb_model_P13612_ = xgb_model_tuple_P13612_03_[0]

      best_xgb_model_P13612_
```

```
[75]: XGBClassifier(base_score=None, booster=None, callbacks=None,
                    colsample_bylevel=None, colsample_bynode=None,
                    colsample_bytree=1.0, device=None, early_stopping_rounds=None,
                    enable_categorical=False, eval_metric=None, feature_types=None,
                    gamma=0.2, grow_policy=None, importance_type=None,
                    interaction_constraints=None, learning_rate=0.1, max_bin=None,
                    max_cat_threshold=None, max_cat_to_onehot=None,
                    max_delta_step=None, max_depth=3, max_leaves=None,
                    min_child_weight=5, missing=nan, monotone_constraints=None,
                    multi_strategy=None, n_estimators=50, n_jobs=None, num_class=2,
                    num_parallel_tree=None, …)
```

Save the Best XGBoost Multicalss Classifier Model using Morgan Fingerprint features for P13612

```
[77]: xgb_model_filename = os.path.join('trained models/XGBoost Multiclass Classifier␣
        ↪Models', 'xgb_model_P13612_.joblib')
      dump(best_xgb_model_P13612_, xgb_model_filename)

      PRINT('Model Saved.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

GraphConvModel Multiclass Classifier Model for P13612

```
[613]: P13612_df_for_training.head(5)
```

```
[613]:                                                SMILES  \
       0  COc1ccccc1-c1ccc(C[C@H](NC(=O)C2(S(=O)(=O)c3cc…
       1  C/C=C/[C@H](CC(=O)O)NC(=O)CN(CCC(C)C)C(=O)Cc1c…
       2  CC(C)CCNCc1ccccc1-c1ccc(C[C@H](NC(=O)c2ccccc2C…
       3  Cc1c(-c2ccc(C[C@H](NC(=O)c3c(C(C)C)cccc3C(C)C)…
       4  CCCCCOc1ccc(C[C@H](NC(=O)[C@@H]2CSCN2C(C)=O)C(…

          NumericUniProtTargetLabels
       0                           0
       1                           0
       2                           0
       3                           1
       4                           0
```

```
[614]: csv_dataset_P13612_for_GraphConv_path = os.path.join('data', 'csv Files for␣
         ↪DeepChem GraphConvModel', 'P13612_df_GCM.csv')
```

```
[ ]: P13612_df_for_training.to_csv(csv_dataset_P13612_for_GraphConv_path,␣
      ↪index=False)
```

Hyperparameter Tuning for Graph Conv Model

```
[19]: # Define the hyperparameter grid
      params = {
          'learning_rate': [1e-3, 5e-4, 1e-4],
          'dropout': [0.2, 0.4],
          'batch_normalize': [True, False],
          'n_classes': [2]
      }

      # Execute hyperparameter tuning for graph conv model for the first dataset
      res_ls = execute_hyperparameter_tuning_for_graph_conv(csv_data =␣
       ↪csv_dataset_P13612_for_GraphConv_path, df=P13612_df_for_training,␣
       ↪params=params)
```

```
smiles_field is deprecated and will be removed in a future version of
DeepChem.Use feature_field instead.
C:\Users\gavvi\anaconda3\Lib\site-packages\deepchem\data\data_loader.py:160:
FutureWarning: featurize() is deprecated and has been renamed to
create_dataset().featurize() will be removed in DeepChem 3.0
  warnings.warn(

WARNING:tensorflow:From C:\Users\gavvi\anaconda3\Lib\site-
packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated.
Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:5 out of the last 5 calls to <function
KerasModel._compute_model at 0x000001CA854039C0> triggered tf.function
retracing. Tracing is expensive and the excessive number of tracings could be
due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with
different shapes, (3) passing Python objects instead of tensors. For (1), please
define your @tf.function outside of the loop. For (2), @tf.function has
reduce_retracing=True option that can avoid unnecessary retracing. For (3),
please refer to https://www.tensorflow.org/guide/function#controlling_retracing
and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 6 calls to <function
KerasModel._compute_model at 0x000001CA854039C0> triggered tf.function
retracing. Tracing is expensive and the excessive number of tracings could be
due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with
different shapes, (3) passing Python objects instead of tensors. For (1), please
define your @tf.function outside of the loop. For (2), @tf.function has
reduce_retracing=True option that can avoid unnecessary retracing. For (3),
please refer to https://www.tensorflow.org/guide/function#controlling_retracing
and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

[21]:
```
PRINT(f"The results after preforming Grid Hyperparameter Optimization technique␣
↪are:")
PRINT(f"Best hyperparameters (learning_rate, dropout, batch_normalize,␣
↪n_classes) -> {res_ls[0]}")
PRINT(f"All results :\n\n{list(res_ls[1].values())}")
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results after preforming Grid Hyperparameter Optimization technique are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Best hyperparameters (learning_rate, dropout, batch_normalize, n_classes) ->
(0.0005, 0.2, False, 2)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
All results :

[0.7706786171574904, 0.8432778489116517, 0.6697823303457107, 0.6425096030729833,
0.8231754161331626, 0.8462227912932138, 0.7455825864276568, 0.6428937259923175,
0.5686299615877081, 0.6962868117797696, 0.6941101152368758, 0.4914212548015365]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Build and Train Graph Conv Model

[624]:
```
training_score_list = []
validation_score_list = []
cv_folds = 10

metrics = [dc.metrics.Metric(dc.metrics.roc_auc_score, np.mean,␣
↪mode='classification')]

featurizer = dc.feat.ConvMolFeaturizer()
tasks = ['NumericUniProtTargetLabels']
loader = dc.data.CSVLoader(tasks=tasks,
                           smiles_field='SMILES',
                           featurizer=featurizer)
dataset = loader.featurize(csv_dataset_P13612_for_GraphConv_path)
```

```
smiles_field is deprecated and will be removed in a future version of
DeepChem.Use feature_field instead.
C:\Users\gavvi\anaconda3\Lib\site-packages\deepchem\data\data_loader.py:160:
FutureWarning: featurize() is deprecated and has been renamed to
create_dataset().featurize() will be removed in DeepChem 3.0
  warnings.warn(
```

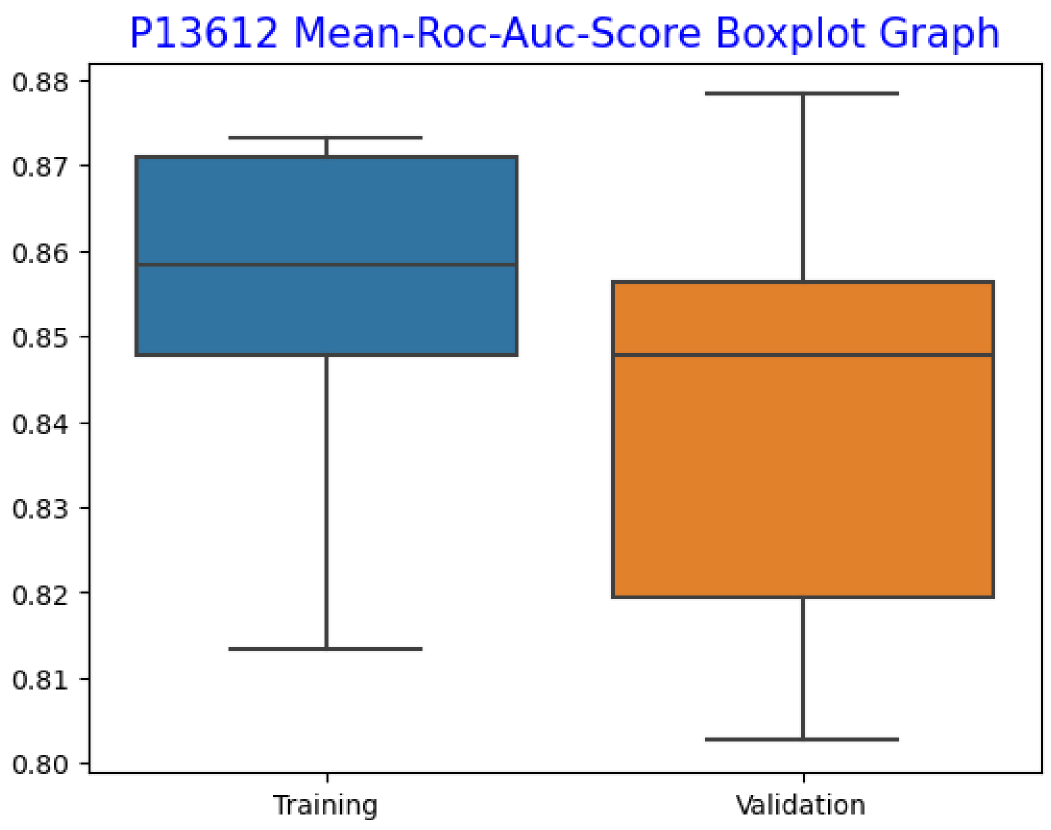
[625]:
```
# Use splitter only once to obtain consistent train/valid splits
splitter = dc.splits.RandomSplitter()

# Create the model outside the loop
```

```
model = generate_graph_conv_model(dropout=0.2, batch_normalize=False,␣
 ↪n_classes=2, learning_rate=0.0005, model_dir='models/gcm_P13612')

for i in range(0, cv_folds):
    split = splitter.train_valid_test_split(dataset)
    # Split the dataset into train, validation, and test sets
    train_dataset, valid_dataset, test_dataset = split

    # Train the model
    model.fit(train_dataset, nb_epoch=10)

    # Evaluate on training set
    train_scores = model.evaluate(train_dataset, metrics, [], n_classes=2)
    training_score_list.append(train_scores['mean-roc_auc_score'])

    # Evaluate on validation set
    validation_scores = model.evaluate(valid_dataset, metrics, [], n_classes=2)
    validation_score_list.append(validation_scores['mean-roc_auc_score'])
```

Visualize Model Preformance

[626]:
```
GenerateBoxplotForModelPreformaceVisualization(UniProt='P13612',␣
 ↪cv_folds=cv_folds , training_score_list=training_score_list ,␣
 ↪validation_score_list=validation_score_list )
```

```
C:\Users\gavvi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1765:
FutureWarning: unique with argument that is not not a Series, Index,
ExtensionArray, or np.ndarray is deprecated and will raise in a future version.
  order = pd.unique(vector)
```



Predict on the Test Dataset and Visualize Performance

[627]:
```
# Evaluate on test set
test_scores = model.evaluate(test_dataset, metrics, [], n_classes=2)
test_roc_auc = test_scores['mean-roc_auc_score']

# Make predictions on the test set
test_predictions = model.predict(test_dataset)

# Extract true labels and predicted probabilities for the positive class
true_labels = test_dataset.y.flatten()

# Get predicted label using helper function
predicted_probs = get_class_labels(predicted_probs=test_predictions)
```

[649]:
```
ture_labels = true_labels.astype(int)
```

[653]:
```
correct_predictions = [true == pred for true, pred in zip(true_labels,␣
 ↪predicted_probs)]

accuracy = sum(correct_predictions) / len(correct_predictions)
```

```
PRINT(f"Correct Predictions: {correct_predictions}")
PRINT(f"Accuracy: {accuracy:.4f}")
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Correct Predictions: [False, True, True, False, True, True, True, True, True,
True, True, False, True, False, False, True, True, True, True, True, True,
False, True, False, True, True, True, False, True, True, True, True, True, True,
False, True, True, True, True, True, True, True, True, True, True, True, False,
False, True, True, True, True, True, False, True, True, True, False, True, True,
True, False, True, True, False, True, True, True, True, False, False, True,
True, True, True, True, True, False, False, False, False, True, True, True,
True, True, True, False, False, True, True, True, True, True, True, True, True,
True, False, False, True, False, False, True, True, True, True, True, False,
True, True, True, True, True, True, True, True, True, True, False, False, False,
False, True, True, True, True, True, True, False, True, True, True, True, False,
False, True, True, True, True, True, True, True, True, True, False, False, True,
True, True, True, True, True, False, True, True, True, True, True, False, False,
True, True, True, True, True, True, True, False, True, False, True, True, True,
True, True, True, True, True, True, True, False, True, True, True, True, True,
False, True, False, True, True, True, True, True, True, True, True]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Accuracy: 0.7727
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Pick the Best Model for P13612 Protein

Load Preveious Trained Models

```
[510]: rf_P13612_rdkitd_path = 'trained models\\Random Forest Multiclass Classifier␣
         ↪Models\\rf_model_P13612.joblib'
       xbg_P13612_rdkitd_path = 'trained models\\XGBoost Multiclass Classifier␣
         ↪Models\\xgb_model_P13612.joblib'
       rf_P13612_morganf_path = 'trained models\\Random Forest Multiclass Classifier␣
         ↪Models\\rf_model_P13612_.joblib'
       xgb_P13612_morganf_path = 'trained models\\XGBoost Multiclass Classifier␣
         ↪Models\\xgb_model_P13612_.joblib'
```

```
[511]: rf_P13612_rdkit = load(rf_P13612_rdkitd_path)
       xbg_P13612_rdkitd = load(xbg_P13612_rdkitd_path)
       rf_P13612_morganf = load(rf_P13612_morganf_path)
       xgb_P13612_morganf = load(xgb_P13612_morganf_path)
```

```
[513]: P13612_df_for_testing, P13612_df_with_uniprots_col, _ =␣
         ↪generate_df_for_training('P13612', 'P13612.csv', 'first_df_encoded.csv')
```

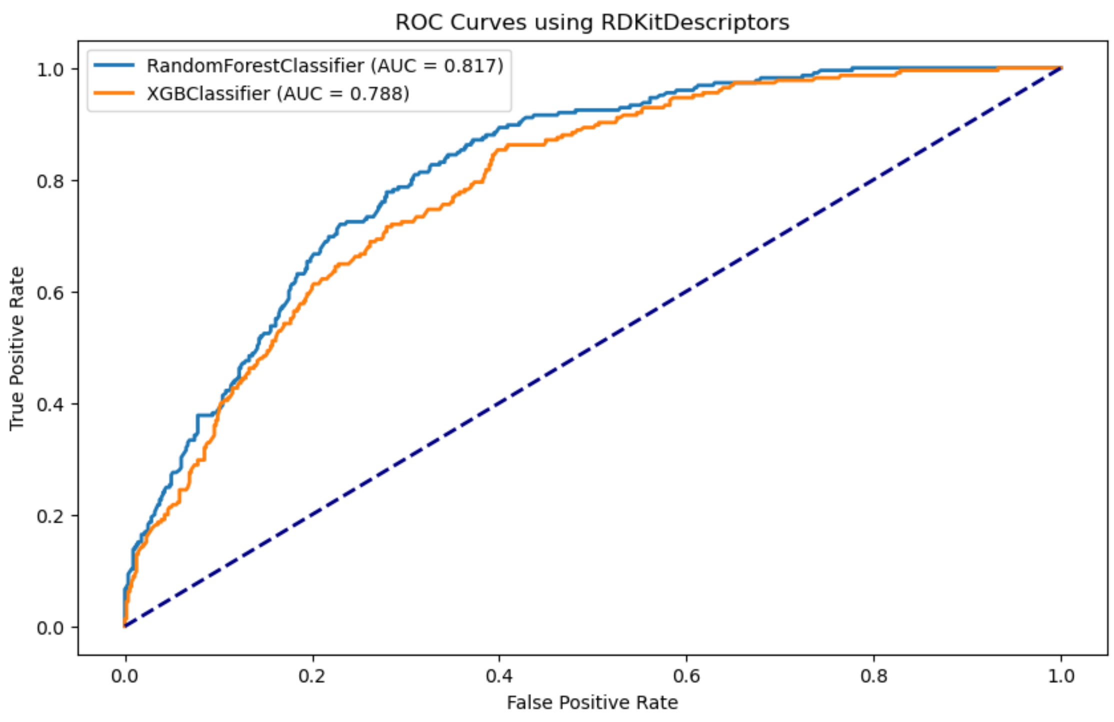
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
P13612 model labels -> ['P05556', 'P26010']
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished generating DataFrames for UniProt -> P13612.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[514]:
```
P13612_df_for_test_rdkd =␣
  ↪GenerateFeaturesByMoleculeSMILES(df=P13612_df_for_testing)
P13612_df_for_test_mfp =␣
  ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P13612_df_for_testing,␣
  ↪size=1024, radius=2)
```

[520]:
```
visualize_best_models_testing_preformace_(df_for_testing=P13612_df_for_test_rdkd,
                                          rf_model=rf_P13612_rdkit,
                                         xbg_model=xbg_P13612_rdkitd,
                                        features_method='RDKitDescriptors')
```

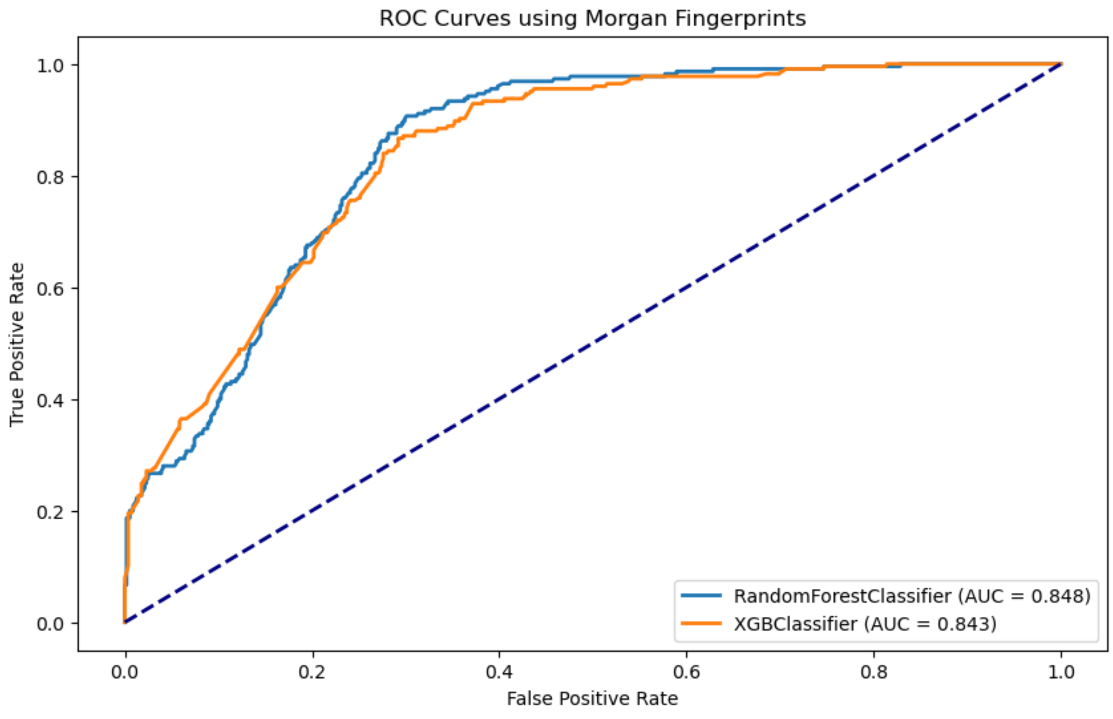
```
hi
hi
```



```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
RandomForestClassifier: ROC-AUC Score -> 0.817
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
XGBClassifier: ROC-AUC Score -> 0.788
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[521]: visualize_best_models_testing_preformace_(df_for_testing=P13612_df_for_test_mfp,
                                                rf_model=rf_P13612_morganf,
                                                xbg_model=xgb_P13612_morganf,
                                               features_method='Morgan Fingerprints')
```

```
hi
hi
```



```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
RandomForestClassifier: ROC-AUC Score -> 0.848
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
XGBClassifier: ROC-AUC Score -> 0.843
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

For that dataset, we will choose both the *GraphConvModel* and *XGBoost* models utilizing *Morgan Fingerprints* features. Both models performed very well, and we will attempt predictions on the unseen dataset using both of them.

```
[654]: final_model_P13612 = os.path.join('trained models/Best Model of each UniProt',␣
         ↪'final_xgb_P13612.joblib')
       dump(xgb_P13612_morganf, final_model_P13612)

       PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

We don't need to save our *GraphConvModel* since we specified a directory for saving during its training. Later, we will simply load the model for our needs.

1.3.5 Models for P05556 Protein

[572]:
```
P05556_df_for_training, P05556_df_with_uniprotes_col, mapped_label_dict =␣
↪generate_df_for_training('P05556', 'P05556.csv', 'second_df_encoded.csv')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
P05556 model labels -> ['O75578', 'P05106', 'P06756', 'P08648', 'P13612',
'P17301', 'P23229', 'P56199', 'Q13797']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished generating DataFrames for UniProt -> P05556.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[573]:
```
P05556_df_for_training.head(3)
```

[573]:
```
                                              SMILES  \
0  O=C(O)CC1OC(=O)N(CC(=O)NCC2CCC(Nc3nc4ccccc4[nH…
1  N[C@@H](Cc1ccc(O)cc1)C(=O)N[C@H]1CSSC[C@@H](C(…
2  CCCCOc1ccc(C[C@@H]2NC(=O)[C@@H](CC(=O)O)NC(=O)…

   NumericUniProtTargetLabels
0                           3
1                           4
2                           3
```

[574]:
```
P05556_df_with_uniprotes_col.head(3)
```

[574]:
```
                                              SMILES UniProtTargetLabels  \
0  O=C(O)CC1OC(=O)N(CC(=O)NCC2CCC(Nc3nc4ccccc4[nH…              P08648
1  N[C@@H](Cc1ccc(O)cc1)C(=O)N[C@H]1CSSC[C@@H](C(…              P13612
2  CCCCOc1ccc(C[C@@H]2NC(=O)[C@@H](CC(=O)O)NC(=O)…              P08648

   NumericUniProtTargetLabels
0                           3
1                           4
2                           3
```

[575]:
```
PRINT(f'The mapped labels in ("UniProt": "index_label") format:
↪\n\n{mapped_label_dict}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The mapped labels in ("UniProt": "index_label") format:

{'O75578': 0, 'P05106': 1, 'P06756': 2, 'P08648': 3, 'P13612': 4, 'P17301': 5,
```

```
'P23229': 6, 'P56199': 7, 'Q13797': 8}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Handle Bad Rows Further in the code, we encountered an exception where the function attempted to extract RDKitDescriptors using a molecule SMILES value of *float* type. We cannot pass this type of value; only *str* type is accepted.

As a solution, we will remove those lines from our dataset.

[576]:
```
PRINT(P05556_df_for_training['SMILES'].apply(type).value_counts())
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
SMILES
<class 'str'>      2196
<class 'float'>       1
Name: count, dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[577]:
```
# Identify rows with 'float' values in the 'SMILES' column
float_rows = P05556_df_for_training['SMILES'].apply(lambda x: isinstance(x,␣
 ↪float))

# Display the rows with 'float' values
float_rows_data = P05556_df_for_training[float_rows]

PRINT(float_rows_data)
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    SMILES  NumericUniProtTargetLabels
878    NaN                           3
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[578]:
```
# Drop rows with 'float' values in the 'SMILES' column
P05556_df_for_training = P05556_df_for_training[~float_rows]
```

[579]:
```
PRINT(P05556_df_for_training['SMILES'].apply(type).value_counts())
PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
SMILES
<class 'str'>    2196
Name: count, dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Quick Dataset Analyse

- **Size of the data frame:** 2197

- **Number of times each protein appears:**
  - O75578: 1
  - P23229: 1
  - P56199: 6
  - Q13797: 10
  - P17301: 57
  - P05106: 37
  - P06756: 170
  - P08648: 463 - 1 for the *bad row*, thus 462
  - P13612: 1452

As we can observe, our dataset exhibits a significant imbalance, with two classes having only one instance each in the entire dataset. Furthermore, some classes occur in the range of 6-60 instances, while others are more prevalent, with frequencies exceeding 100 and even reaching as high as 1452.

To mitigate this issue, we will introduce weights during the training phase and specify that we intend to treat each class in a balanced manner, ensuring that the smaller classes receive more consideration.

Random Forest Multiclass Classifier Model for P05556 with added RDKitDescriptors Features

[601]:
```
P05556_df_for_training_ =␣
↪GenerateFeaturesByMoleculeSMILES(df=P05556_df_for_training)
```

[602]:
```
P05556_df_for_training_.head(3)
```

[602]:
```
                                              SMILES     MolWt  \
0  O=C(O)CC1OC(=O)N(CC(=O)NCC2CCC(Nc3nc4ccccc4[nH…   491.548
1  N[C@@H](Cc1ccc(O)cc1)C(=O)N[C@H]1CSSC[C@@H](C(…   504.590
2  CCCCOc1ccc(C[C@@H]2NC(=O)[C@@H](CC(=O)O)NC(=O)…  1177.423

   NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  NumRotatableBonds  \
0                  188  136.65   3.8224              10                  8
1                  180  170.85   0.9635              12                  5
2                  450  471.53  -3.5630              31                 20

   HeavyAtomCount  FractionCSP3  NumericUniProtTargetLabels
0              36      0.384615                           3
1              34      0.272727                           4
2              81      0.620000                           3
```

[603]:
```
weight_dict = 'balanced'
```

```
rf_model_tuple_P05556_01 =␣
  ↪GenerateRandomForestModel(df=P05556_df_for_training_,␣
  ↪weight_dict=weight_dict)
```

```
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\model_selection\_split.py:700: UserWarning: The least populated
class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
```

```
Classification Report:
              precision    recall  f1-score   support

           1       0.25      0.71      0.37         7
           2       0.50      0.69      0.58        32
           3       0.64      0.54      0.59        94
           4       0.93      0.88      0.90       295
           5       0.88      0.64      0.74        11
           7       0.00      0.00      0.00         0
           8       0.25      1.00      0.40         1

    accuracy                           0.79       440
   macro avg       0.49      0.64      0.51       440
weighted avg       0.82      0.79      0.80       440


C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Recall
is ill-defined and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
[1203]: PRINT(f'The results of Random Forest Multiclass Classifier model for\nUniProt␣
          ↪P05556 are:')
```

```
print_dict_meaningful(rf_model_tuple_P05556_01[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier model for
UniProt P05556 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.786
precision: 0.822
recall: 0.786
f1_score: 0.799
confusion_matrix: [[5, 0, 0, 2, 0, 0, 0], [0, 22, 7, 3, 0, 0, 0], [13, 15, 51,
15, 0, 0, 0], [2, 7, 22, 260, 1, 0, 3], [0, 0, 0, 0, 7, 4, 0], [0, 0, 0, 0, 0,
0, 0], [0, 0, 0, 0, 0, 0, 1]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Save the Random Forest Multicalss Classifier Model for P05556

[1201]:
```
rf_model_filename = os.path.join('trained models/Random Forest Multiclass␣
  ↪Classifier Models', 'rf_model_P05556.joblib')
dump(rf_model_tuple_P05556_01[0], rf_model_filename)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Random Forest Multiclass Classifier Model for P05556 with added Morgan Fingerprints Features

[586]:
```
P05556_df_for_training__ =␣
  ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P05556_df_for_training,␣
  ↪size=1024, radius=2)
```

[582]:
```
P05556_df_for_training__.head(3)
```

[582]:
```
                                              SMILES  \
0  O=C(O)CC1OC(=O)N(CC(=O)NCC2CCC(Nc3nc4ccccc4[nH…
1  N[C@@H](Cc1ccc(O)cc1)C(=O)N[C@H]1CSSC[C@@H](C(…
2  CCCCOc1ccc(C[C@@H]2NC(=O)[C@@H](CC(=O)O)NC(=O)…

   NumericUniProtTargetLabels  Feature_0  Feature_1  Feature_2  Feature_3  \
0                         3.0        0.0        0.0        0.0        1.0
1                         4.0        0.0        1.0        0.0        0.0
2                         3.0        0.0        1.0        0.0        0.0

   Feature_4  Feature_5  Feature_6  Feature_7  …  Feature_1014  \
```

```
0        0.0        0.0        0.0        0.0  …            0.0
1        0.0        0.0        0.0        0.0  …            0.0
2        1.0        1.0        0.0        0.0  …            0.0

   Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
0           0.0           0.0           0.0           0.0           1.0
1           0.0           0.0           1.0           0.0           1.0
2           0.0           0.0           1.0           0.0           1.0

   Feature_1020  Feature_1021  Feature_1022  Feature_1023
0           0.0           0.0           0.0           0.0
1           0.0           0.0           0.0           0.0
2           0.0           0.0           0.0           0.0

[3 rows x 1026 columns]
```

Check for Generated Features With NaN values

```
[587]: original_rows = P05556_df_for_training__.shape[0]

P05556_df_for_training__ = P05556_df_for_training__.dropna()

# Calculate the number of dropped rows
dropped_rows = original_rows - P05556_df_for_training__.shape[0]

PRINT(f"{dropped_rows} rows were dropped.")
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
2 rows were dropped.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[584]: weight_dict = 'balanced'

rf_model_tuple_P05556_01_ =␣
  ↪GenerateRandomForestModel(df=P05556_df_for_training__,␣
  ↪weight_dict=weight_dict)
```

```
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\model_selection\_split.py:700: UserWarning: The least populated
class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(

Classification Report:
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00         1
         1.0       0.17      0.38      0.23         8
         2.0       0.32      0.41      0.36        29
         3.0       0.38      0.27      0.32        88
         4.0       0.78      0.78      0.78       299
```

```
         5.0       0.27      0.31      0.29        13
         6.0       0.00      0.00      0.00         0
         7.0       0.00      0.00      0.00         0
         8.0       0.00      0.00      0.00         1

    accuracy                           0.63       439
   macro avg       0.21      0.24      0.22       439
weighted avg       0.64      0.63      0.63       439

C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Recall
is ill-defined and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
```

```
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
[1205]: PRINT(f'The results of Random Forest Multiclass Classifier using Morgan␣
         ↪Fingerprints features model for\nUniProt P05556 are:')
        print_dict_meaningful(rf_model_tuple_P05556_01_[1])
        PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier using Morgan Fingerprints
features model for
UniProt P05556 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.631
precision: 0.636
recall: 0.631
f1_score: 0.631
confusion_matrix: [[0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 3, 0, 1, 4, 0, 0, 0, 0], [0,
1, 12, 5, 11, 0, 0, 0, 0], [0, 9, 7, 24, 45, 3, 0, 0, 0], [0, 5, 17, 33, 234, 8,
1, 0, 1], [0, 0, 2, 1, 5, 4, 0, 1, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0,
0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Pick the Best Model for P05556 Protein "With that dataset, we encountered several challenges:

1. A highly unbalanced dataset with classes appearing 1-6 times in the entire dataset.
2. Issues arose when generating XGBoost models.
3. Difficulty in creating a GraphConvModel using DeepChem due to the dataset's small size and problems with the library in splitting the data in a way that GraphConvModel could effectively work with.

Nevertheless, we achieved commendable performance from the Random Forest Multiclass Classifier we trained using RDKitDescriptors, generating 8 meaningful features from the molecules' SMILES values. The confusion matrix we obtained indicates that we achieved true positives not only in the larger classes but also in the smaller ones. This suggests that our model generalizes to some extent and has the potential to predict the smaller classes present in our dataset:

```
[[5, 0, 0, 2, 0, 0, 0],
 [0, 22, 7, 3, 0, 0, 0],
 [13, 15, 51, 15, 0, 0, 0],
 [2, 7, 22, 260, 1, 0, 3],
 [0, 0, 0, 0, 7, 4, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 1]]
```

```
[1206]: rf_P05556_rdkitd_path = 'trained models\\Random Forest Multiclass Classifier␣
          ↪Models\\rf_model_P05556.joblib'
        rf_P08648_rdkitd = load(rf_P05556_rdkitd_path)
```

```
[1207]: rf_P08648_rdkitd
```

```
[1207]: RandomForestClassifier(class_weight='balanced', max_depth=10,
                               min_samples_leaf=2, min_samples_split=10,
                               n_estimators=200, random_state=42)
```

```
[1208]: final_model_P05556 = os.path.join('trained models/Best Model of each UniProt',␣
          ↪'final_rf_P05556.joblib')
        dump(rf_P08648_rdkitd, final_model_P05556)

        PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.3.6 Models for P05106 Protein

```
[119]: P05106_df_for_training, P05106_df_with_uniprotes_col, mapped_label_dict_P05106␣
         ↪= generate_df_for_training('P05106', 'P05106.csv', 'third_df_encoded.csv')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
P05106 model labels -> ['P05556', 'P06756', 'P08514', 'P17301', 'P26006']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished generating DataFrames for UniProt -> P05106.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[199]: P05106_df_for_training.head(3)
```

```
[199]:                                               SMILES  \
       0  COC(=O)c1ccc(COC(=O)N[C@@H](CCC(=O)N2CCN(c3ccc…
       1  COCCOCCOCCOCC(=O)Nc1cc(C[C@H](NS(=O)(=O)c2cccc…
       2  Cl.O=C(NC(Cc1ccc2cc(OCCCN3CCNCC3)ccc2c1)C(=O)O…

          NumericUniProtTargetLabels
       0                           1
       1                           1
       2                           2
```

```
[208]: P05106_df_with_uniprotes_col.head(3)
```

```
[208]:                                               SMILES UniProtTargetLabels  \
       0  COC(=O)c1ccc(COC(=O)N[C@@H](CCC(=O)N2CCN(c3ccc…              P06756
```

```
1  COCCOCCOCCOCC(=O)Nc1cc(C[C@H](NS(=O)(=O)c2cccc…           P06756
2  Cl.O=C(NC(Cc1ccc2cc(OCCCN3CCNCC3)ccc2c1)C(=O)O…           P08514

   NumericUniProtTargetLabels
0                           1
1                           1
2                           2
```

[209]:
```
PRINT(f'The mapped labels in ("UniProt": "index_label") format:
↪\n\n{mapped_label_dict_P05106}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The mapped labels in ("UniProt": "index_label") format:

{'P05556': 0, 'P06756': 1, 'P08514': 2, 'P17301': 3, 'P26006': 4}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Handle Bad Rows Further in the code, we encountered an exception where the function attempted to extract RDKitDescriptors using a molecule SMILES value of *float* type. We cannot pass this type of value; only *str* type is accepted.

As a solution, we will remove those lines from our dataset.

[123]:
```
PRINT(P05106_df_for_training['SMILES'].apply(type).value_counts())
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
SMILES
<class 'str'>      4475
<class 'float'>       3
Name: count, dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[124]:
```
# Identify rows with 'float' values in the 'SMILES' column
float_rows = P05106_df_for_training['SMILES'].apply(lambda x: isinstance(x,␣
↪float))

# Display the rows with 'float' values
float_rows_data = P05106_df_for_training[float_rows]

PRINT(float_rows_data)
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     SMILES  NumericUniProtTargetLabels
3434    NaN                           2
4202    NaN                           2
4343    NaN                           1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[125]:
```
# Drop rows with 'float' values in the 'SMILES' column
P05106_df_for_training = P05106_df_for_training[~float_rows]
```

[126]:
```
PRINT(P05106_df_for_training['SMILES'].apply(type).value_counts())
PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
SMILES
<class 'str'>    4475
Name: count, dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Quick Dataset Analyse

- **Size of the data frame:** 4478 - 3 =4475

- **Number of times each protein appears:**
  - P17301: 20
  - P05556: 37
  - P26006: 25
  - P06756: 2058 - 1 = 2057
  - P08514: 2338 - 2 = 2336

Random Forest Multiclass Classifier Model for P05106 with added RDKitDescriptors Features

[37]:
```
P05106_df_for_training_ =␣
↪GenerateFeaturesByMoleculeSMILES(df=P05106_df_for_training)
```

[85]:
```
P05106_df_for_training_.head(3)
```

[85]:
```
                                              SMILES    MolWt  \
0  COC(=O)c1ccc(COC(=O)N[C@@H](CCC(=O)N2CCN(c3ccc…  580.642
1  COCCOCCOCCOCC(=O)Nc1cc(C[C@H](NS(=O)(=O)c2cccc…  797.850
2  Cl.O=C(NC(Cc1ccc2cc(OCCCN3CCNCC3)ccc2c1)C(=O)O…  516.013

   NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  NumRotatableBonds  \
0                  224  161.90   2.0428              13                 10
1                  302  195.67   3.9484              19                 24
2                  192   90.90   3.5005               9                 10

   HeavyAtomCount  FractionCSP3  NumericUniProtTargetLabels
0              42      0.413793                           1
1              55      0.472222                           1
```

```
2              36      0.333333                           2
```

[41]:
```
weight_dict = 'balanced'

rf_model_tuple_P05106_01 =␣
  ↪GenerateRandomForestModel(df=P05106_df_for_training_,␣
  ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.20      0.60      0.30         5
           1       0.64      0.62      0.63       424
           2       0.66      0.67      0.66       458
           3       0.00      0.00      0.00         6
           4       0.12      0.50      0.20         2

    accuracy                           0.64       895
   macro avg       0.33      0.48      0.36       895
weighted avg       0.64      0.64      0.64       895
```

[58]:
```
weight_dict = {0: 2, 1: 1, 2: 1, 3: 2, 4: 2}

rf_model_tuple_P05106_02 =␣
  ↪GenerateRandomForestModel(df=P05106_df_for_training_,␣
  ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         5
           1       0.68      0.59      0.63       424
           2       0.66      0.76      0.70       458
           3       0.00      0.00      0.00         6
           4       0.00      0.00      0.00         2

    accuracy                           0.67       895
   macro avg       0.27      0.27      0.27       895
weighted avg       0.66      0.67      0.66       895


C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
```

```
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[82]:
```
PRINT(f'The results of the best Random Forest Multiclass Classifier model␣
 ↪for\nUniProt P05106 are:')
print_dict_meaningful(rf_model_tuple_P05106_01[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of the best Random Forest Multiclass Classifier model for
UniProt P05106 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.637
precision: 0.643
recall: 0.637
f1_score: 0.639
confusion_matrix: [[3, 2, 0, 0, 0], [10, 261, 152, 0, 1], [2, 145, 305, 2, 4],
[0, 1, 3, 0, 2], [0, 0, 0, 1, 1]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[83]:
```
best_rf_model_P05106 = rf_model_tuple_P05106_01[0]

best_rf_model_P05106
```

[83]:
```
RandomForestClassifier(class_weight='balanced', max_depth=10,
                       min_samples_leaf=4, min_samples_split=15,
                       random_state=42)
```

Save the Best Random Forest Multicalss Classifier Model for P05106

[84]:
```
rf_model_filename = os.path.join('trained models/Random Forest Multiclass␣
 ↪Classifier Models', 'rf_model_P05106.joblib')
dump(best_rf_model_P05106, rf_model_filename)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model using RKDitDescriptors features for P05106

[86]:
```
weight_dict = 'balanced'

xgb_model_tuple_P05106_01 = GenerateXGBoostModel(df=P05106_df_for_training_,␣
  ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.14      0.60      0.23         5
           1       0.68      0.66      0.67       424
           2       0.71      0.67      0.69       458
           3       0.10      0.17      0.12         6
           4       0.07      0.50      0.12         2

    accuracy                           0.66       895
   macro avg       0.34      0.52      0.37       895
weighted avg       0.69      0.66      0.67       895
```

[87]:
```
weight_dict = {0: 2, 1: 1, 2: 1, 3: 2, 4: 2}

xgb_model_tuple_P05106_02 = GenerateXGBoostModel(df=P05106_df_for_training_,␣
  ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.40      0.40      0.40         5
           1       0.70      0.61      0.65       424
           2       0.68      0.76      0.71       458
           3       0.00      0.00      0.00         6
           4       0.20      0.50      0.29         2

    accuracy                           0.68       895
   macro avg       0.39      0.45      0.41       895
weighted avg       0.68      0.68      0.68       895
```

[88]:
```
PRINT(f'The results of the best XGBoost Multiclass Classifier model␣
  ↪for\nUniProt P05106 are:')
print_dict_meaningful(xgb_model_tuple_P05106_01[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
The results of the best XGBoost Multiclass Classifier model for
UniProt P05106 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.665
precision: 0.687
recall: 0.665
f1_score: 0.674
confusion_matrix: [[3, 1, 1, 0, 0], [15, 281, 122, 2, 4], [3, 133, 309, 6, 7],
[0, 1, 1, 1, 3], [0, 0, 0, 1, 1]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[89]:
```
best_xgb_model_P05106 = xgb_model_tuple_P05106_01[0]

best_xgb_model_P05106
```

[89]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
              min_child_weight=1, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=100, n_jobs=None, num_class=5,
              num_parallel_tree=None, …)
```

Save the Best Random Forest Multicalss Classifier uaing RKDitDescriptors Model for P05106

[90]:
```
xgb_model_filename = os.path.join('trained models/XGBoost Multiclass Classifier␣
  ↪Models', 'xgb_model_P05106.joblib')
dump(best_xgb_model_P05106, xgb_model_filename)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Random Forest Multiclass Classifier Model for P05106 with added Morgan Fingerprints Features

[322]:
```
P05106_df_for_training__ =␣
  ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P05106_df_for_training,␣
  ↪size=1024, radius=2)
```

[325]:
```
original_rows = P05106_df_for_training__.shape[0]

P05106_df_for_training__ = P05106_df_for_training__.dropna()

# Calculate the number of dropped rows
dropped_rows = original_rows - P05106_df_for_training__.shape[0]

PRINT(f"{dropped_rows} rows were dropped.")
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
6 rows were dropped.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[326]:
```
P05106_df_for_training__['NumericUniProtTargetLabels'] =␣
 ↪P05106_df_for_training__['NumericUniProtTargetLabels'].astype(int)
```

[327]:
```
P05106_df_for_training__.head(5)
```

[327]:
```
                                              SMILES  \
0  COC(=O)c1ccc(COC(=O)N[C@@H](CCC(=O)N2CCN(c3ccc…
1  COCCOCCOCCOCC(=O)Nc1cc(C[C@H](NS(=O)(=O)c2cccc…
2  Cl.O=C(NC(Cc1ccc2cc(OCCCN3CCNCC3)ccc2c1)C(=O)O…
3                CC(C)(C)c1nn2c(=O)cc(N3CCNCC3)nc2s1
4  O=C(O)C[C@@H](CC1CCN(C(=O)CCc2ccc3c(n2)NCCC3)C…

   NumericUniProtTargetLabels  Feature_0  Feature_1  Feature_2  Feature_3  \
0                           1        0.0        1.0        0.0        0.0
1                           1        0.0        1.0        0.0        0.0
2                           2        0.0        1.0        0.0        0.0
3                           2        0.0        0.0        0.0        0.0
4                           1        0.0        1.0        0.0        0.0

   Feature_4  Feature_5  Feature_6  Feature_7  …  Feature_1014  \
0        1.0        0.0        0.0        0.0  …           0.0
1        1.0        0.0        0.0        0.0  …           0.0
2        0.0        0.0        0.0        0.0  …           0.0
3        0.0        0.0        0.0        0.0  …           0.0
4        1.0        0.0        0.0        0.0  …           0.0

   Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
0           0.0           0.0           0.0           1.0           0.0
1           0.0           0.0           0.0           0.0           1.0
2           0.0           0.0           0.0           0.0           0.0
3           0.0           0.0           0.0           0.0           0.0
4           0.0           0.0           1.0           0.0           1.0

   Feature_1020  Feature_1021  Feature_1022  Feature_1023
0           0.0           0.0           0.0           0.0
```

```
1            0.0           0.0           0.0           0.0
2            0.0           0.0           0.0           0.0
3            0.0           0.0           0.0           0.0
4            0.0           0.0           0.0           0.0

[5 rows x 1026 columns]
```

[139]:
```
weight_dict = 'balanced'

rf_model_tuple_P05106_01_ =␣
 ↪GenerateRandomForestModel(df=P05106_df_for_training__,␣
 ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.08      0.40      0.14         5
         1.0       0.66      0.73      0.69       421
         2.0       0.76      0.64      0.69       461
         3.0       0.17      0.67      0.27         3
         4.0       0.50      0.40      0.44         5

    accuracy                           0.68       895
   macro avg       0.43      0.57      0.45       895
weighted avg       0.70      0.68      0.68       895
```

[140]:
```
weight_dict = {0: 2, 1: 1, 2: 1, 3: 2, 4: 2}

rf_model_tuple_P05106_02_ =␣
 ↪GenerateRandomForestModel(df=P05106_df_for_training__,␣
 ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.33      0.20      0.25         5
         1.0       0.65      0.76      0.70       421
         2.0       0.73      0.64      0.68       461
         3.0       0.00      0.00      0.00         3
         4.0       0.40      0.40      0.40         5

    accuracy                           0.69       895
   macro avg       0.42      0.40      0.41       895
weighted avg       0.69      0.69      0.68       895


C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
```

```
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
[328]: PRINT(f'The results of the best Random Forest Multiclass Classifier␣
        ↪model\nusing Morgan Fingerprints features for UniProt P05106 are:')
       print_dict_meaningful(rf_model_tuple_P05106_02_[1])
       PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of the best Random Forest Multiclass Classifier model
using Morgan Fingerprints features for UniProt P05106 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.687
precision: 0.690
recall: 0.687
f1_score: 0.685
confusion_matrix: [[1, 3, 1, 0, 0], [2, 318, 100, 0, 1], [0, 165, 294, 0, 2],
[0, 0, 3, 0, 0], [0, 1, 2, 0, 2]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[329]: best_rf_model_P05106_ = rf_model_tuple_P05106_02_[0]

       best_rf_model_P05106_
```

```
[329]: RandomForestClassifier(class_weight={0: 2, 1: 1, 2: 1, 3: 2, 4: 2},
                              max_depth=10, min_samples_leaf=8, n_estimators=200,
                              random_state=42)
```

Save the Best Random Forest Multicalss Classifier Model using Morgan Fingerprint features for P13612

```
[330]: rf_model_filename = os.path.join('trained models/Random Forest Multiclass␣
         ↪Classifier Models', 'rf_model_P05106_.joblib')
       dump(best_rf_model_P05106_, rf_model_filename)

       PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model for P05106 with added Morgan Fingerprints Features

```
[141]: weight_dict = 'balanced'

       xgb_model_tuple_P05106_01_ = GenerateXGBoostModel(df=P05106_df_for_training__,␣
         ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.06      0.40      0.11         5
         1.0       0.65      0.68      0.67       421
         2.0       0.74      0.64      0.69       461
         3.0       0.10      0.67      0.17         3
         4.0       0.40      0.40      0.40         5

    accuracy                           0.66       895
   macro avg       0.39      0.56      0.41       895
weighted avg       0.69      0.66      0.67       895
```

```
[142]: weight_dict = {0: 2, 1: 1, 2: 1, 3: 2, 4: 2}

       xgb_model_tuple_P05106_02_ = GenerateXGBoostModel(df=P05106_df_for_training__,␣
         ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.13      0.40      0.20         5
         1.0       0.65      0.76      0.70       421
         2.0       0.76      0.62      0.68       461
         3.0       0.50      0.33      0.40         3
         4.0       0.50      0.40      0.44         5

    accuracy                           0.69       895
   macro avg       0.51      0.50      0.49       895
weighted avg       0.70      0.69      0.69       895
```

[145]:
```
PRINT(f'The results of the best XGBoost Multiclass Classifier model\nusing␣
↪Morgan Fingerprints features for UniProt P05106 are:')
print_dict_meaningful(xgb_model_tuple_P05106_02_[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of the best XGBoost Multiclass Classifier model
using Morgan Fingerprints features for UniProt P05106 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.686
precision: 0.701
recall: 0.686
f1_score: 0.688
confusion_matrix: [[2, 2, 1, 0, 0], [10, 321, 89, 0, 1], [3, 168, 288, 1, 1],
[0, 0, 2, 1, 0], [0, 2, 1, 0, 2]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[146]:
```
best_xgb_model_P05106_ = xgb_model_tuple_P05106_02_[0]

best_xgb_model_P05106_
```

[146]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0.2, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.01, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
              min_child_weight=5, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=100, n_jobs=None, num_class=5,
              num_parallel_tree=None, …)
```

Save the Best XGBoost Multicalss Classifier Model using Morgan Fingerprint features for P05106

[148]:
```
xgb_model_filename = os.path.join('trained models/XGBoost Multiclass Classifier␣
↪Models', 'xgb_model_P05106_.joblib')
dump(best_xgb_model_P05106_, xgb_model_filename)

PRINT('Model Saved.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

GraphConvModel Multiclass Classifier Model for P05106

```
[818]: P05106_df_for_training
```

```
[818]:                                                    SMILES  \
       0     COC(=O)c1ccc(COC(=O)N[C@@H](CCC(=O)N2CCN(c3ccc…
       1     COCCOCCOCCOCC(=O)Nc1cc(C[C@H](NS(=O)(=O)c2cccc…
       2     Cl.O=C(NC(Cc1ccc2cc(OCCCN3CCNCC3)ccc2c1)C(=O)O…
       3                    CC(C)(C)c1nn2c(=O)cc(N3CCNCC3)nc2s1
       4     O=C(O)C[C@@H](CC1CCN(C(=O)CCc2ccc3c(n2)NCCC3)C…
       …                                                   …
       4473  CC(C)C[C@@H]1NC(=O)CNC(=O)[C@@H](CC(=O)O)NC(=O…
       4474  O=C(O)CC(Cc1nc(CCCc2ccc3c(n2)NCCC3)no1)c1ccc2n…
       4475              CC(Cn1ccc2cc(OCCCNc3ccccn3)ccc21)C(=O)O
       4476  Cc1cc(Cl)cc([C@H](CC(=O)O)NC(=O)CNC(=O)c2cc(O)…
       4477  O=C(CCc1ccccc1)N[C@@H](CNC(=O)c1ccc2c(cnn2CCCN…

             NumericUniProtTargetLabels
       0                              1
       1                              1
       2                              2
       3                              2
       4                              1
       …                            …
       4473                           1
       4474                           2
       4475                           1
       4476                           1
       4477                           2

       [4475 rows x 2 columns]
```

```
[830]: csv_dataset_P05106_for_GraphConv_path = os.path.join('data', 'csv Files for␣
        ↪DeepChem GraphConvModel', 'P05106_df_GCM.csv')
```

```
[832]: P05106_df_for_training.to_csv(csv_dataset_P05106_for_GraphConv_path,␣
        ↪index=False)
```

Build and Train Graph Conv Model

```
[833]: training_score_list = []
       validation_score_list = []
       cv_folds = 10

       metrics = [dc.metrics.Metric(dc.metrics.roc_auc_score, np.mean,␣
        ↪mode='classification')]

       featurizer = dc.feat.ConvMolFeaturizer()
       tasks = ['NumericUniProtTargetLabels']
       loader = dc.data.CSVLoader(tasks=tasks,
                                  smiles_field='SMILES',
                                  featurizer=featurizer)
       dataset = loader.featurize(csv_dataset_P05106_for_GraphConv_path)
```

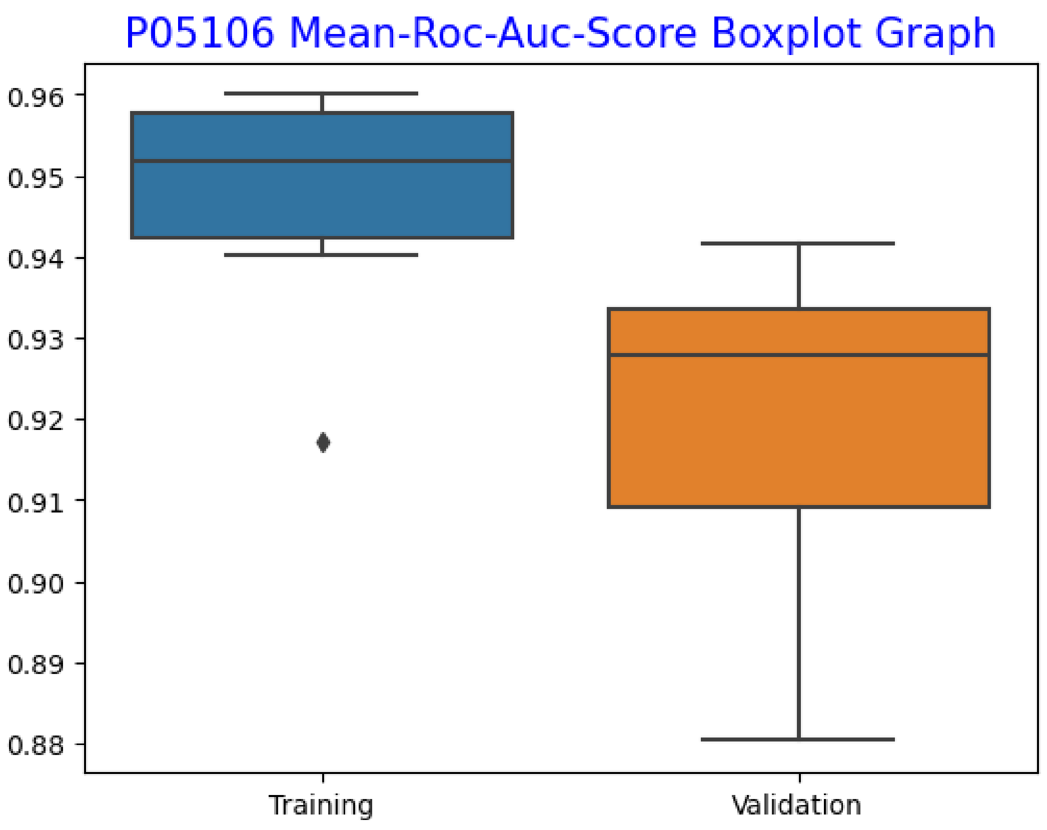
```
smiles_field is deprecated and will be removed in a future version of
DeepChem.Use feature_field instead.
```

[834]:
```
# Use splitter only once to obtain consistent train/valid splits
splitter = dc.splits.RandomSplitter()

# Create the model outside the loop
model = generate_graph_conv_model(dropout=0.2, batch_normalize=True,␣
  ↪n_classes=5, learning_rate=0.005, model_dir='models/gcm_P05106')

for i in range(0, cv_folds):

    # Give more weight to valid&test because of data unblance
    split = splitter.train_valid_test_split(dataset,frac_train=0.6,␣
  ↪frac_valid=0.2, frac_test=0.2)
    # Split the dataset into train, validation, and test sets
    train_dataset, valid_dataset, test_dataset = split

    # Train the model
    model.fit(train_dataset, nb_epoch=10)

    # Evaluate on training set
    train_scores = model.evaluate(train_dataset, metrics, [], n_classes=5)
    training_score_list.append(train_scores['mean-roc_auc_score'])

    # Evaluate on validation set
    validation_scores = model.evaluate(valid_dataset, metrics, [], n_classes=5)
    validation_score_list.append(validation_scores['mean-roc_auc_score'])
```

Visualize Model Preformance

[372]:
```
GenerateBoxplotForModelPreformaceVisualization(UniProt='P05106',␣
  ↪cv_folds=cv_folds , training_score_list=training_score_list ,␣
  ↪validation_score_list=validation_score_list)
```

```
C:\Users\gavvi\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1765:
FutureWarning: unique with argument that is not not a Series, Index,
ExtensionArray, or np.ndarray is deprecated and will raise in a future version.
  order = pd.unique(vector)
```



Predict on the Test Dataset and Visualize Performance

[836]:
```
# Evaluate on test set
test_scores = model.evaluate(test_dataset, metrics, [], n_classes=5)
test_roc_auc = test_scores['mean-roc_auc_score']

# Make predictions on the test set
test_predictions = model.predict(test_dataset)

# Extract true labels and predicted probabilities for the positive class
true_labels = test_dataset.y.flatten()

# Get predicted label using helper function
predicted_probs = get_class_labels(predicted_probs=test_predictions)
```

[838]:
```
PRINT(predicted_probs)
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[2 2 1 2 2 1 1 1 2 2 1 2 1 2 2 1 2 1 1 2 2 1 2 2 2 1 2 2 2 2 1 1 2 1 0 2 1
 1 1 1 2 1 1 1 2 2 1 2 2 1 2 2 1 1 1 1 1 1 2 1 2 1 1 1 1 2 2 1 2 2 1 1 2 2
 2 1 1 2 1 2 1 2 1 2 2 1 1 1 2 2 2 2 1 1 2 2 1 1 2 2 2 1 2 1 2 1 1 1 2 2 2
 2 2 2 1 2 2 2 1 1 2 2 1 1 2 2 2 1 1 1 1 1 1 2 2 1 2 1 2 1 2 1 1 1 2 1 1 2
```

```
 1 2 1 2 1 2 2 1 2 2 1 2 2 1 2 1 1 1 2 1 2 2 1 1 1 1 2 1 1 1 1 1 2 1 2 1 2
 1 1 1 1 2 1 2 1 2 2 2 1 1 1 2 1 1 2 2 2 2 0 1 1 1 1 2 1 1 0 1 2 2 1 1 1 2
 2 2 2 1 2 2 2 2 2 2 2 0 2 2 2 1 3 1 1 1 2 2 1 1 1 1 1 2 2 1 1 2 3 1 2 0 1
 2 2 2 2 2 2 2 2 0 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 2 1 2 1 2 2
 1 2 1 2 2 1 2 2 1 1 2 2 2 1 1 2 2 1 1 2 2 1 1 1 2 2 1 2 1 1 2 1 1 2 2 2 2
 1 2 1 1 2 2 2 1 1 1 1 1 1 2 2 2 1 1 1 2 2 1 1 2 2 2 2 1 2 2 2 1 1 2 1 1 1
 2 2 2 2 2 2 2 1 2 0 1 2 1 2 2 1 1 1 1 1 2 2 1 2 2 2 2 1 1 2 2 1 2 1 2 2 2
 1 1 1 2 2 2 2 2 2 2 2 2 1 2 2 2 1 2 1 2 2 1 2 2 2 2 1 1 1 1 2 2 1 2 1 2 1
 2 2 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 2 2 1 1 3 1 2 1 1 2 2 2 2 1 2 2 1 1 1 2
 1 1 1 2 2 2 2 2 1 0 1 2 2 1 2 1 2 1 1 1 2 2 1 2 2 1 1 2 1 2 2 1 1 1 1 1 1
 2 1 1 1 2 1 2 1 2 1 1 2 1 1 2 2 1 2 2 2 1 2 1 1 1 1 1 1 1 2 1 1 1 2 1 2 2
 2 1 1 1 1 2 1 1 1 1 2 2 2 2 1 1 1 1 2 1 2 1 2 2 2 1 2 0 2 1 2 1 2 1 2 2 1
 1 2 2 1 1 1 1 2 1 2 1 1 1 1 2 2 2 1 1 2 2 2 1 2 1 2 1 2 2 2 1 2 2 1 1 1 1
 1 0 2 1 2 2 1 2 1 2 2 2 2 2 2 4 2 2 2 1 2 2 2 1 1 1 2 1 2 2 2 2 2 2 1 1 2
 2 2 2 1 0 1 2 2 1 4 2 2 1 1 2 2 2 1 2 2 2 1 2 1 1 1 2 1 1 1 1 2 2 2 1 2 2
 1 2 1 1 2 2 2 2 1 2 2 1 1 1 2 2 2 1 2 1 2 2 2 2 1 1 2 2 2 1 2 2 2 1 1 2 2
 2 2 2 2 1 2 1 2 1 2 1 1 2 1 2 2 2 1 1 1 2 2 2 1 2 2 1 2 1 1 1 1 2 1 2 1 1
 1 2 2 1 1 2 1 2 1 2 1 1 1 2 2 2 1 1 2 1 1 2 2 2 2 1 2 1 2 1 2 0 1 2 2 1 1
 1 2 2 0 2 1 1 2 2 0 2 2 2 2 2 1 2 1 1 1 2 1 2 1 1 2 2 2 2 2 1 1 2 2 2 2 2
 2 1 2 2 1 2 2 1 2 2 2 2 2 2 2 1 1 1 1 1 2 2 1 1 1 1 1 2 2 1 1 1 1 1 2 0 1
 2 1 1 2 1 1 1]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[374]: report = classification_report(true_labels, predicted_probs)

       PRINT(report)
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
              precision    recall  f1-score   support

         0.0       0.38      0.50      0.43         6
         1.0       0.75      0.74      0.75       414
         2.0       0.78      0.77      0.78       468
         3.0       0.50      1.00      0.67         3
         4.0       0.50      1.00      0.67         4

    accuracy                           0.76       895
   macro avg       0.58      0.80      0.66       895
weighted avg       0.76      0.76      0.76       895

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Pick the Best Model for P05106 Protein Based on our observations, it is evident that the GraphConv model excels in every aspect, displaying a notably high mean ROC AUC score. Additionally, it yields favorable results in precision, recall, and F1-score, particularly for the larger classes, while maintaining satisfactory performance on the smaller classes. Moreover, it achieves the highest accuracy compared to all other models. Consequently, we have chosen it as the best model for UniProt P05106.6

```
[381]: final_P05106 = dc.models.GraphConvModel(model_dir='models/gcm_P05106',␣
         ↪n_tasks=1)
       final_P05106.restore()

       PRINT('Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.3.7 Models for P05107 Protein

```
[382]: P05107_df_for_training, P05107_df_with_uniprotes_col, mapped_label_dict_P05107␣
         ↪= generate_df_for_training('P05107', 'P05107.csv', 'fourth_df_encoded.csv')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
P05107 model labels -> ['P11215', 'P20701']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished generating DataFrames for UniProt -> P05107.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[383]: P05107_df_for_training.head(3)
```

```
[383]:                                              SMILES  \
       0  NCc1ccc(NC(=O)N2C(=O)CC2CC(=O)O)cc1.O=C(O)C(F)…
       1  O=C(Nc1ccc(C(=O)Nc2cccc3cccc(S(=O)(=O)O)c23)cc…
       2  COC(=O)CN1C(=O)S/C(=C\c2ccc(-c3ccc(C(=O)O)cc3)…

          NumericUniProtTargetLabels
       0                           1
       1                           0
       2                           0
```

```
[384]: P05107_df_with_uniprotes_col.head(3)
```

```
[384]:                                              SMILES UniProtTargetLabels  \
       0  NCc1ccc(NC(=O)N2C(=O)CC2CC(=O)O)cc1.O=C(O)C(F)…              P20701
       1  O=C(Nc1ccc(C(=O)Nc2cccc3cccc(S(=O)(=O)O)c23)cc…              P11215
       2  COC(=O)CN1C(=O)S/C(=C\c2ccc(-c3ccc(C(=O)O)cc3)…              P11215

          NumericUniProtTargetLabels
       0                           1
       1                           0
       2                           0
```

```
[385]: PRINT(f'The mapped labels in ("UniProt": "index_label") format:
         ↪\n\n{mapped_label_dict_P05107}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The mapped labels in ("UniProt": "index_label") format:

{'P11215': 0, 'P20701': 1}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Quick Dataset Analysis

- **Size of the data frame:** 66

- **Number of times each protein appears:**
  - P11215: 33
  - P20701: 33

Clearly, the dataset demonstrates perfect balance, obviating the necessity of assigning disparate weights to individual classes. Nonetheless, it is important to acknowledge the relatively diminutive size of the dataset. In light of this, the implementation of K-fold cross-validation in each model serves as a mitigating strategy for this limitation.

Furthermore, for a dataset of this modest size, opting for deep learning models such as the GraphConvModel from the DeepChem library may not be the most suitable choice. This is because the model could struggle to generalize effectively, leading to overfitting.

Consequently, we have decided to exclusively train Random Forest and XGBoost multiclass classifiers with balanced weights, selecting the superior performer from these two models..

Random Forest Multiclass Classifier Model for P05107 with added RDKitDescriptors Features

[387]:
```
P05107_df_for_training_ =␣
↪GenerateFeaturesByMoleculeSMILES(df=P05107_df_for_training)
```

[388]:
```
P05107_df_for_training_.head(3)
```

[388]:
```
                                              SMILES    MolWt  \
0  NCc1ccc(NC(=O)N2C(=O)CC2CC(=O)O)cc1.O=C(O)C(F)…  391.302
1  O=C(Nc1ccc(C(=O)Nc2cccc3cccc(S(=O)(=O)O)c23)cc…  491.481
2  COC(=O)CN1C(=O)S/C(=C\c2ccc(-c3ccc(C(=O)O)cc3)…  387.369

   NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  NumRotatableBonds  \
0                  148  150.03   1.3861              12                  4
1                  176  155.71   4.4993              11                  6
2                  138  114.12   2.8541               9                  5

   HeavyAtomCount  FractionCSP3  NumericUniProtTargetLabels
0              27      0.333333                           1
1              35      0.000000                           0
2              27      0.111111                           0
```

```
[389]: weight_dict = {0: 1, 1: 1}

       rf_model_tuple_P05107_01 =␣
        ↪GenerateRandomForestModel(df=P05107_df_for_training_,␣
        ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.80      0.89         5
           1       0.90      1.00      0.95         9

    accuracy                           0.93        14
   macro avg       0.95      0.90      0.92        14
weighted avg       0.94      0.93      0.93        14
```

```
[390]: PRINT(f'The results of Random Forest Multiclass Classifier model for\nUniProt␣
        ↪P05107 are:')
       print_dict_meaningful(rf_model_tuple_P05107_01[1])
       PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier model for
UniProt P05107 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.929
precision: 0.936
recall: 0.929
f1_score: 0.926
confusion_matrix: [[4, 1], [0, 9]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[391]: best_rf_model_P05107 = rf_model_tuple_P05107_01[0]

       best_rf_model_P05107
```

```
[391]: RandomForestClassifier(class_weight={0: 1, 1: 1}, random_state=42)
```

Save the Random Forest Multicalss Classifier Model for P05107

```
[392]: rf_model_filename = os.path.join('trained models/Random Forest Multiclass␣
        ↪Classifier Models', 'rf_model_P05107.joblib')
       dump(best_rf_model_P05107, rf_model_filename)

       PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model using RKDitDescriptors features for P05107

[393]:
```
weight_dict = {0: 1, 1: 1}

xgb_model_tuple_P05107_01 = GenerateXGBoostModel(df=P05107_df_for_training_,␣
  ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.80      0.80         5
           1       0.89      0.89      0.89         9

    accuracy                           0.86        14
   macro avg       0.84      0.84      0.84        14
weighted avg       0.86      0.86      0.86        14
```

[396]:
```
PRINT(f'The results of XGBoost Multiclass Classifier model for\nUniProt P05107␣
  ↪are:')
print_dict_meaningful(xgb_model_tuple_P05107_01[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of XGBoost Multiclass Classifier model for
UniProt P05107 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.857
precision: 0.857
recall: 0.857
f1_score: 0.857
confusion_matrix: [[4, 1], [1, 8]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[400]:
```
best_xgb_model_P05107 = xgb_model_tuple_P05107_01[0]

best_xgb_model_P05107
```

[400]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0, grow_policy=None, importance_type=None,
```

```
              interaction_constraints=None, learning_rate=0.01, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
              min_child_weight=1, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=50, n_jobs=None, num_class=2,
              num_parallel_tree=None, …)
```

Save the XGBoost Multicalss Classifier Model for P05107

[401]:
```
xgb_model_filename = os.path.join('trained models/XGBoost Multiclass Classifier␣
 ↪Models', 'xgb_model_P05107.joblib')
dump(best_xgb_model_P05107, xgb_model_filename)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Random Forest Multiclass Classifier Model for P05107 with added Morgan Fingerprints Features

[1016]:
```
P05107_df_for_training__ =␣
 ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P05107_df_for_training,␣
 ↪size=1024, radius=2)
```

[1017]:
```
P05107_df_for_training__.head(3)
```

[1017]:
```
                                              SMILES  \
0  NCc1ccc(NC(=O)N2C(=O)CC2CC(=O)O)cc1.O=C(O)C(F)…
1  O=C(Nc1ccc(C(=O)Nc2cccc3cccc(S(=O)(=O)O)c23)cc…
2  COC(=O)CN1C(=O)S/C(=C\c2ccc(-c3ccc(C(=O)O)cc3)…

   NumericUniProtTargetLabels  Feature_0  Feature_1  Feature_2  Feature_3  \
0                           1        0.0        0.0        0.0        0.0
1                           0        0.0        0.0        0.0        0.0
2                           0        0.0        0.0        0.0        0.0

   Feature_4  Feature_5  Feature_6  Feature_7  …  Feature_1014  \
0        0.0        0.0        0.0        0.0  …           0.0
1        0.0        0.0        0.0        0.0  …           0.0
2        0.0        0.0        0.0        0.0  …           0.0

   Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
0           0.0           0.0           0.0           0.0           1.0
1           0.0           0.0           0.0           0.0           0.0
2           0.0           0.0           0.0           0.0           0.0

   Feature_1020  Feature_1021  Feature_1022  Feature_1023
```

```
0          0.0          0.0          0.0          0.0
1          0.0          0.0          0.0          0.0
2          1.0          0.0          0.0          0.0

[3 rows x 1026 columns]
```

[404]:
```
weight_dict = {0: 1, 1: 1}

rf_model_tuple_P05107_01_ =␣
 ↪GenerateRandomForestModel(df=P05107_df_for_training__,␣
 ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.80      0.89         5
           1       0.90      1.00      0.95         9

    accuracy                           0.93        14
   macro avg       0.95      0.90      0.92        14
weighted avg       0.94      0.93      0.93        14
```

[405]:
```
PRINT(f'The results of Random Forest Multiclass Classifier model\nusing Morgan␣
 ↪Fingerprints features for UniProt P05107 are:')
print_dict_meaningful(rf_model_tuple_P05107_01_[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier model
using Morgan Fingerprints features for UniProt P05107 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.929
precision: 0.936
recall: 0.929
f1_score: 0.926
confusion_matrix: [[4, 1], [0, 9]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[409]:
```
best_rf_model_P05107_ = rf_model_tuple_P05107_01_[0]

best_rf_model_P05107_
```

[409]:
```
RandomForestClassifier(class_weight={0: 1, 1: 1}, n_estimators=50,
                       random_state=42)
```

Save Random Forest Multicalss Classifier Model using Morgan Fingerprint features for P05107

```
[410]: rf_model_filename = os.path.join('trained models/Random Forest Multiclass␣
         ↪Classifier Models', 'rf_model_P05107_.joblib')
       dump(best_rf_model_P05107_, rf_model_filename)

       PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model for P05107 with added Morgan Fingerprints Features

```
[411]: weight_dict = {0: 1, 1: 1}

       xgb_model_tuple_P05107_01_ = GenerateXGBoostModel(df=P05107_df_for_training__,␣
         ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.80      0.89         5
           1       0.90      1.00      0.95         9

    accuracy                           0.93        14
   macro avg       0.95      0.90      0.92        14
weighted avg       0.94      0.93      0.93        14
```

```
[412]: PRINT(f'The results of XGBoost Multiclass Classifier model\nusing Morgan␣
         ↪Fingerprints features for UniProt P05107 are:')
       print_dict_meaningful(xgb_model_tuple_P05107_01_[1])
       PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of XGBoost Multiclass Classifier model
using Morgan Fingerprints features for UniProt P05107 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.929
precision: 0.936
recall: 0.929
f1_score: 0.926
confusion_matrix: [[4, 1], [0, 9]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[415]:
```
best_xgb_model_P05107_ = xgb_model_tuple_P05107_01_[0]

best_xgb_model_P05107_
```

[415]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.01, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
              min_child_weight=1, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=50, n_jobs=None, num_class=2,
              num_parallel_tree=None, …)
```

Save XGBoost Multicalss Classifier Model using Morgan Fingerprint features for P05107

[509]:
```
xgb_model_filename = os.path.join('trained models/XGBoost Multiclass Classifier␣
 ↪Models', 'xgb_model_P05107_.joblib')
dump(best_xgb_model_P05107_, xgb_model_filename)

PRINT('Model Saved.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Pick the Best Model for P05107 Protein Based on the observations above, all four models performed quite well. Therefore, we will randomly select the XGBoost Multi-Class Classifier model that utilizes the generation of Morgan Fingerprints. The reason for this choice is that by utilizing Morgan Fingerprints features, we obtain 1024 new features for our data, while RDKit Descriptors utilize only 8 features (we select those 8 most meaningful features)

[463]:
```
xgb_P05107_morganf_path = 'trained models\\XGBoost Multiclass Classifier␣
 ↪Models\\xgb_model_P05107_.joblib'
xgb_P05107_morganf = load(xgb_P05107_morganf_path)

PRINT('Model Loaded.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Loaded.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1019]:
```
xgb_model_filename = os.path.join('trained models/Best Model of each UniProt',␣
 ↪'final_xgb_P05107.joblib')
dump(xgb_P05107_morganf, xgb_model_filename)
```

```
PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.3.8 Models for P08648 Protein

[478]:
```
P08648_df_for_training, P08648_df_with_uniprotes_col, mapped_label_dict_P08648␣
↪= generate_df_for_training('P08648', 'P08648.csv', 'fifth_df_encoded.csv')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
P08648 model labels -> ['P05556', 'P06756']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished generating DataFrames for UniProt -> P08648.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[479]:
```
P08648_df_for_training.head(3)
```

[479]:
```
                                              SMILES  \
0  O=C(CO[C@@H]1C[C@@H](CNc2ccccn2)N(C(=O)OCc2ccc…
1  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…
2  O=C(N[C@@H](Cc1cccc(OCCNc2ccccn2)c1)C(=O)O)c1c…

   NumericUniProtTargetLabels
0                           0
1                           0
2                           0
```

[480]:
```
P08648_df_with_uniprotes_col.head(3)
```

[480]:
```
                                              SMILES UniProtTargetLabels  \
0  O=C(CO[C@@H]1C[C@@H](CNc2ccccn2)N(C(=O)OCc2ccc…              P05556
1  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…              P05556
2  O=C(N[C@@H](Cc1cccc(OCCNc2ccccn2)c1)C(=O)O)c1c…              P05556

   NumericUniProtTargetLabels
0                           0
1                           0
2                           0
```

[481]:
```
PRINT(f'The mapped labels in ("UniProt": "index_label") format:
↪\n\n{mapped_label_dict_P08648}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The mapped labels in ("UniProt": "index_label") format:
```

```
{'P05556': 0, 'P06756': 1}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Check for Rows with NaN Values

```
[483]: # Identify rows with 'float' values in the 'SMILES' column
float_rows = P08648_df_for_training['SMILES'].apply(lambda x: isinstance(x,␣
 ↪float))


# Display the rows with 'float' values
float_rows_data = P08648_df_for_training[float_rows]

PRINT(float_rows_data)
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    SMILES  NumericUniProtTargetLabels
387    NaN                           0
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[485]: original_rows = P08648_df_for_training.shape[0]

P08648_df_for_training = P08648_df_for_training.dropna()

# Calculate the number of dropped rows
dropped_rows = original_rows - P08648_df_for_training.shape[0]

PRINT(f"{dropped_rows} rows were dropped.")
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1 rows were dropped.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Quick Dataset Analysis

- **Size of the data frame:** 469 - 1 = 468

- **Number of times each protein appears:**
  - P05556: 463 - 1 = 462
  - P0756: 6

Once again, we are confronted with a relatively small dataset comprising only `468` rows. Consequently, opting for deep learning models like the *DeepChem GraphConvModel* might not be the optimal choice. In this scenario, we will adhere to employing *Random Forest* and *XGBoost* multi-class classifiers.

Furthermore, the dataset exhibits a notable imbalance, prompting us to explore potential solutions by assigning different weights to each class. This strategic approach aims to mitigate the impact of class imbalance during the training of our models.

Random Forest Multiclass Classifier Model for P08648 with added RDKitDescriptors Features

```
[486]: P08648_df_for_training_ =␣
         ↪GenerateFeaturesByMoleculeSMILES(df=P08648_df_for_training)
```

```
[487]: P08648_df_for_training_.head(3)
```

```
[487]:                                                       SMILES    MolWt  \
       0  O=C(CO[C@@H]1C[C@@H](CNc2ccccn2)N(C(=O)OCc2ccc…  711.616
       1  O=C(NCc1ccccc1)NC[C@H](NC(=O)[C@@H]1CCCN1S(=O)…  474.539
       2  O=C(N[C@@H](Cc1cccc(OCCNc2ccccn2)c1)C(=O)O)c1c…  474.344

          NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  NumRotatableBonds  \
       0                  268  159.19   4.3268              18                 13
       1                  176  144.91   0.9085              11                  9
       2                  166  100.55   4.3050               9                 10

          HeavyAtomCount  FractionCSP3  NumericUniProtTargetLabels
       0              50      0.343750                           0
       1              33      0.318182                           0
       2              32      0.173913                           0
```

```
[488]: weight_dict = 'balanced'

       rf_model_tuple_P08648_01 =␣
         ↪GenerateRandomForestModel(df=P08648_df_for_training_,␣
         ↪weight_dict=weight_dict)
```

```
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\model_selection\_split.py:700: UserWarning: The least populated
class in y has only 3 members, which is less than n_splits=5.
  warnings.warn(

Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        91
           1       0.00      0.00      0.00         3

    accuracy                           0.97        94
   macro avg       0.48      0.50      0.49        94
weighted avg       0.94      0.97      0.95        94


C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
```

```
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

It appears that even when attempting to assign balanced weights to both classes to address the issue of imbalanced data, challenges persist with the smaller class.

```
[491]: PRINT(f'The results of the best Random Forest Multiclass Classifier model␣
         ↪for\nUniProt P05106 are:')
       print_dict_meaningful(rf_model_tuple_P08648_01[1])
       PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of the best Random Forest Multiclass Classifier model for
UniProt P05106 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.968
precision: 0.937
recall: 0.968
f1_score: 0.952
confusion_matrix: [[91, 0], [3, 0]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Save the Random Forest Multicalss Classifier Model for P08648

```
[508]: rf_model_filename = os.path.join('trained models/Random Forest Multiclass␣
         ↪Classifier Models', 'rf_model_P08648.joblib')
       dump(rf_model_tuple_P08648_01[0], rf_model_filename)

       PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model using RKDitDescriptors features for P08648

[490]:
```
weight_dict = 'balanced'

xgb_model_tuple_P08648_01 = GenerateXGBoostModel(df=P08648_df_for_training_,␣
↪weight_dict=weight_dict)
```

```
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\model_selection\_split.py:700: UserWarning: The least populated
class in y has only 3 members, which is less than n_splits=5.
  warnings.warn(

Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        91
           1       0.00      0.00      0.00         3

    accuracy                           0.97        94
   macro avg       0.48      0.50      0.49        94
weighted avg       0.94      0.97      0.95        94


C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\gavvi\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[492]:
```
PRINT(f'The results of the best Random Forest Multiclass Classifier model␣
↪for\nUniProt P05106 are:')
print_dict_meaningful(xgb_model_tuple_P08648_01[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
The results of the best Random Forest Multiclass Classifier model for
UniProt P05106 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.968
precision: 0.937
recall: 0.968
f1_score: 0.952
confusion_matrix: [[91, 0], [3, 0]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Save XGBoost Multicalss Classifier uaing RKDitDescriptors Model for P08648

[507]:
```
xgb_model_filename = os.path.join('trained models/XGBoost Multiclass Classifier␣
  ↪Models', 'xgb_model_P08648.joblib')
dump(xgb_model_tuple_P08648_01[0], xgb_model_filename)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Random Forest Multiclass Classifier Model for P08648 with added Morgan Fingerprints Features

[493]:
```
P08648_df_for_training__ =␣
  ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P08648_df_for_training,␣
  ↪size=1024, radius=2)
```

Drop new row that contains *Nan* values if existed

[495]:
```
original_rows = P08648_df_for_training__.shape[0]

P08648_df_for_training__ = P08648_df_for_training__.dropna()

# Calculate the number of dropped rows
dropped_rows = original_rows - P08648_df_for_training__.shape[0]

PRINT(f"{dropped_rows} rows were dropped.")
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
2 rows were dropped.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[494]:
```
P05106_df_for_training__.head(5)
```

[494]:
```
                                              SMILES  \
0  COC(=O)c1ccc(COC(=O)N[C@@H](CCC(=O)N2CCN(c3ccc…
```

```
1  COCCOCCOCCOCC(=O)Nc1cc(C[C@H](NS(=O)(=O)c2cccc…
2  Cl.O=C(NC(Cc1ccc2cc(OCCCN3CCNCC3)ccc2c1)C(=O)O…
3                 CC(C)(C)c1nn2c(=O)cc(N3CCNCC3)nc2s1
4  O=C(O)C[C@@H](CC1CCN(C(=O)CCc2ccc3c(n2)NCCC3)C…

   NumericUniProtTargetLabels  Feature_0  Feature_1  Feature_2  Feature_3  \
0                           1        0.0        1.0        0.0        0.0
1                           1        0.0        1.0        0.0        0.0
2                           2        0.0        1.0        0.0        0.0
3                           2        0.0        0.0        0.0        0.0
4                           1        0.0        1.0        0.0        0.0

   Feature_4  Feature_5  Feature_6  Feature_7  …  Feature_1014  \
0        1.0        0.0        0.0        0.0  …           0.0
1        1.0        0.0        0.0        0.0  …           0.0
2        0.0        0.0        0.0        0.0  …           0.0
3        0.0        0.0        0.0        0.0  …           0.0
4        1.0        0.0        0.0        0.0  …           0.0

   Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
0           0.0           0.0           0.0           1.0           0.0
1           0.0           0.0           0.0           0.0           1.0
2           0.0           0.0           0.0           0.0           0.0
3           0.0           0.0           0.0           0.0           0.0
4           0.0           0.0           1.0           0.0           1.0

   Feature_1020  Feature_1021  Feature_1022  Feature_1023
0           0.0           0.0           0.0           0.0
1           0.0           0.0           0.0           0.0
2           0.0           0.0           0.0           0.0
3           0.0           0.0           0.0           0.0
4           0.0           0.0           0.0           0.0

[5 rows x 1026 columns]
```

[496]:
```
weight_dict = 'balanced'

rf_model_tuple_P08648_01_ =␣
↪GenerateRandomForestModel(df=P08648_df_for_training__,␣
↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        93
         1.0       1.00      1.00      1.00         1

    accuracy                           1.00        94
```

```
   macro avg       1.00      1.00      1.00        94
weighted avg       1.00      1.00      1.00        94
```

```
[498]: PRINT(f'The results of Random Forest Multiclass Classifier model\nusing Morgan␣
         ↪Fingerprints features for UniProt P08648 are:')
       print_dict_meaningful(rf_model_tuple_P08648_01_[1])
       PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier model
using Morgan Fingerprints features for UniProt P08648 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 1.000
precision: 1.000
recall: 1.000
f1_score: 1.000
confusion_matrix: [[93, 0], [0, 1]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Save Random Forest Multicalss Classifier Model using Morgan Fingerprint features for P08648

```
[506]: rf_model_filename = os.path.join('trained models/Random Forest Multiclass␣
         ↪Classifier Models', 'rf_model_P08648_.joblib')
       dump(rf_model_tuple_P08648_01_[0], rf_model_filename)

       PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model for P08648 with added Morgan Fingerprints Features

```
[497]: weight_dict = 'balanced'

       xgb_model_tuple_P08648_01_ = GenerateXGBoostModel(df=P08648_df_for_training__,␣
         ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        93
         1.0       1.00      1.00      1.00         1

    accuracy                           1.00        94
   macro avg       1.00      1.00      1.00        94
```

```
weighted avg       1.00      1.00      1.00        94
```

```
[500]: PRINT(f'The results of the best XGBoost Multiclass Classifier model\nusing␣
         ↪Morgan Fingerprints features for UniProt P08648 are:')
       print_dict_meaningful(xgb_model_tuple_P08648_01_[1])
       PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of the best XGBoost Multiclass Classifier model
using Morgan Fingerprints features for UniProt P08648 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 1.000
precision: 1.000
recall: 1.000
f1_score: 1.000
confusion_matrix: [[93, 0], [0, 1]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Save XGBoost Multicalss Classifier Model using Morgan Fingerprint features for P08648

```
[505]: xgb_model_filename = os.path.join('trained models/XGBoost Multiclass Classifier␣
         ↪Models', 'xgb_model_P08648_.joblib')
       dump(xgb_model_tuple_P08648_01_[0], xgb_model_filename)

       PRINT('Model Saved.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Pick the Best Model for P08648 Protein Based on the observations of all models' performances, we have obtained the expected outcomes. To make our selection, we will consider the models utilizing a dataframe with features generated by Morgan Fingerprints.

Moreover, we will choose the Random Forest this time, the reason for that is the huge imbalance of our data, so we want to keep our model simple as possible.

```
[570]: rf_P08648_morganf_path = 'trained models\\Random Forest Multiclass Classifier␣
         ↪Models\\rf_model_P08648_.joblib'
       rf_P08648_morganf = load(rf_P08648_morganf_path)
```

```
[571]: final_model_P08648 = os.path.join('trained models/Best Model of each UniProt',␣
         ↪'final_rf_P08648.joblib')
       dump(rf_P08648_morganf, final_model_P08648)

       PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.3.9 Models for P17301 Protein

[501]:
```
P17301_df_for_training, P17301_df_with_uniprots_col, mapped_label_dict_P17301 =␣
↪generate_df_for_training('P17301', 'P17301.csv', 'sixth_df_encoded.csv')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
P17301 model labels -> ['P05106', 'P05556']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished generating DataFrames for UniProt -> P17301.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[502]:
```
P17301_df_for_training.head(3)
```

[502]:
```
                                              SMILES  \
0  O=C1N[C@H](C(=O)O)Cc2cccc(c2)OC/C=C/COc2ccc1c(…
1     O=C1N[C@H](C(=O)O)Cn2cc(nn2)CCCCOc2ccc(Cl)c1c2
2  CC(C)(C)c1cc(Br)cc([C@H](CC(=O)O)NC(=O)CNC(=O)…

   NumericUniProtTargetLabels
0                           1
1                           1
2                           1
```

[503]:
```
P17301_df_with_uniprots_col.head(3)
```

[503]:
```
                                              SMILES UniProtTargetLabels  \
0  O=C1N[C@H](C(=O)O)Cc2cccc(c2)OC/C=C/COc2ccc1c(…              P05556
1     O=C1N[C@H](C(=O)O)Cn2cc(nn2)CCCCOc2ccc(Cl)c1c2              P05556
2  CC(C)(C)c1cc(Br)cc([C@H](CC(=O)O)NC(=O)CNC(=O)…              P05556

   NumericUniProtTargetLabels
0                           1
1                           1
2                           1
```

[504]:
```
PRINT(f'The mapped labels in ("UniProt": "index_label") format:
↪\n\n{mapped_label_dict_P17301}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The mapped labels in ("UniProt": "index_label") format:

{'P05106': 0, 'P05556': 1}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Quick Dataset Analysis

- **Size of the data frame:** 77

- **Number of occurrences for each protein:**
  - P05106: 20
  - P05556: 57

It appears that the sixth dataset is also characterized by its small size and a slight imbalance between the two classes. Consequently, we will once again refrain from training the *GraphConvModel* and concentrate solely on *Random Forest* and *XGBoost* multiclass classification models. We will employ balanced weights to address the dataset's imbalance and enhance model performance.

Random Forest Multiclass Classifier Model for P17301 with added RDKitDescriptors Features

[529]:
```
P17301_df_for_training_ =␣
↪GenerateFeaturesByMoleculeSMILES(df=P17301_df_for_training)
```

[530]:
```
P17301_df_for_training_.head(3)
```

[530]:
```
                                              SMILES    MolWt  \
0  O=C1N[C@H](C(=O)O)Cc2cccc(c2)OC/C=C/COc2ccc1c(…  387.819
1     O=C1N[C@H](C(=O)O)Cn2cc(nn2)CCCCOc2ccc(Cl)c1c2  364.789
2  CC(C)(C)c1cc(Br)cc([C@H](CC(=O)O)NC(=O)CNC(=O)…  590.475

   NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  NumRotatableBonds  \
0                  140   84.86   3.0931               7                  1
1                  132  106.34   1.5298               9                  1
2                  204  172.38   2.2462              12                  8

   HeavyAtomCount  FractionCSP3  NumericUniProtTargetLabels
0              27      0.200000                           1
1              25      0.375000                           1
2              38      0.384615                           1
```

[531]:
```
weight_dict = 'balanced'

rf_model_tuple_P17301_01 =␣
↪GenerateRandomForestModel(df=P17301_df_for_training_,␣
↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.75      0.86         4
           1       0.92      1.00      0.96        12
```

```
    accuracy                           0.94        16
   macro avg       0.96      0.88      0.91        16
weighted avg       0.94      0.94      0.93        16
```

[535]:
```
PRINT(f'The results of Random Forest Multiclass Classifier model for\nUniProt␣
↪P17301 are:')
print_dict_meaningful(rf_model_tuple_P17301_01[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier model for
UniProt P17301 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.938
precision: 0.942
recall: 0.938
f1_score: 0.934
confusion_matrix: [[3, 1], [0, 12]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Save Random Forest Multicalss Classifier Model for P17301

[536]:
```
rf_model_filename = os.path.join('trained models/Random Forest Multiclass␣
↪Classifier Models', 'rf_model_P17301.joblib')
dump(rf_model_tuple_P17301_01[0], rf_model_filename)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model using RKDitDescriptors features for P17301

[537]:
```
weight_dict = 'balanced'

xgb_model_tuple_P17301_01 = GenerateXGBoostModel(df=P17301_df_for_training_,␣
↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.75      0.86         4
           1       0.92      1.00      0.96        12

    accuracy                           0.94        16
   macro avg       0.96      0.88      0.91        16
```

```
weighted avg       0.94      0.94      0.93        16
```

[538]:
```
PRINT(f'The results of XGBoost Multiclass Classifier model for\nUniProt P17301␣
  ↪are:')
print_dict_meaningful(xgb_model_tuple_P17301_01[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of XGBoost Multiclass Classifier model for
UniProt P17301 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 0.938
precision: 0.942
recall: 0.938
f1_score: 0.934
confusion_matrix: [[3, 1], [0, 12]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Save XGBoost Multicalss Classifier Model for P17301

[539]:
```
xgb_model_filename = os.path.join('trained models/XGBoost Multiclass Classifier␣
  ↪Models', 'xgb_model_P17301.joblib')
dump(xgb_model_tuple_P17301_01[0], xgb_model_filename)

PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Random Forest Multiclass Classifier Model for P17301 with added Morgan Fingerprints Features

[540]:
```
P17301_df_for_training__ =␣
  ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P17301_df_for_training,␣
  ↪size=1024, radius=2)
```

[541]:
```
P17301_df_for_training__
```

[541]:
```
                                              SMILES  \
0   O=C1N[C@H](C(=O)O)Cc2cccc(c2)OC/C=C/COc2ccc1c(…
1      O=C1N[C@H](C(=O)O)Cn2cc(nn2)CCCCOc2ccc(Cl)c1c2
2   CC(C)(C)c1cc(Br)cc([C@H](CC(=O)O)NC(=O)CNC(=O)…
3   Cc1cccc(Cl)c1C(=O)N[C@@H](Cc1ccc(NC(=O)c2c(Cl)…
4   O=C(c1ccccc1)c1ccc([N-]S(=O)(=O)c2cccc(-c3ccc(…
..                                                …
72  O=C1N[C@H](C(=O)O)Cc2ccc(cc2)OC/C=C/COc2cccc(C…
```

```
73  O=C(O)CCNC(=O)c1cc(C(=O)Nc2ccc3c(c2)CNCC3)cc([…
74  COc1ccc(C(CC(=O)O)NC(=O)c2cc(C(=O)Nc3ccc4c(c3)…
75  Cc1cc(C)cc(S(=O)(=O)N2CCC[C@H]2C(=O)N[C@@H](CN…
76   O=C1N[C@H](C(=O)O)Cc2ccc(cc2)OCCCCOc2cccc(Cl)c21

    NumericUniProtTargetLabels  Feature_0  Feature_1  Feature_2  Feature_3  \
0                            1        0.0        0.0        0.0        0.0
1                            1        0.0        0.0        0.0        0.0
2                            1        0.0        1.0        0.0        0.0
3                            1        0.0        1.0        0.0        0.0
4                            1        0.0        0.0        0.0        0.0
..                         …          …          …          …          …
72                           1        0.0        0.0        0.0        0.0
73                           0        0.0        0.0        0.0        0.0
74                           0        0.0        1.0        0.0        0.0
75                           1        0.0        1.0        0.0        0.0
76                           1        0.0        0.0        0.0        0.0

    Feature_4  Feature_5  Feature_6  Feature_7  …  Feature_1014  \
0         0.0        0.0        0.0        0.0  …           0.0
1         1.0        0.0        0.0        0.0  …           0.0
2         0.0        0.0        0.0        0.0  …           0.0
3         0.0        0.0        0.0        0.0  …           0.0
4         0.0        0.0        0.0        0.0  …           0.0
..        …          …          …          …  …             …
72        0.0        0.0        0.0        0.0  …           0.0
73        0.0        0.0        0.0        0.0  …           0.0
74        0.0        0.0        0.0        0.0  …           0.0
75        1.0        0.0        0.0        0.0  …           0.0
76        1.0        0.0        0.0        0.0  …           0.0

    Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
0            0.0           0.0           0.0           0.0           1.0
1            0.0           0.0           0.0           0.0           1.0
2            0.0           0.0           0.0           0.0           1.0
3            0.0           0.0           0.0           0.0           0.0
4            0.0           0.0           0.0           0.0           0.0
..           …             …             …             …             …
72           0.0           0.0           0.0           0.0           1.0
73           0.0           0.0           0.0           0.0           0.0
74           0.0           0.0           0.0           0.0           0.0
75           0.0           0.0           0.0           0.0           1.0
76           0.0           0.0           0.0           0.0           1.0

    Feature_1020  Feature_1021  Feature_1022  Feature_1023
0            0.0           0.0           0.0           0.0
1            0.0           0.0           0.0           0.0
```

```
2             0.0           0.0           0.0           0.0
3             0.0           0.0           0.0           0.0
4             0.0           0.0           0.0           0.0
..            …            …            …            …
72            0.0           0.0           0.0           0.0
73            0.0           0.0           0.0           0.0
74            0.0           0.0           0.0           0.0
75            0.0           0.0           0.0           0.0
76            0.0           0.0           0.0           0.0

[77 rows x 1026 columns]
```

[542]:
```
weight_dict = 'balanced'

rf_model_tuple_P17301_01_ =␣
 ↪GenerateRandomForestModel(df=P17301_df_for_training__,␣
 ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00        12

    accuracy                           1.00        16
   macro avg       1.00      1.00      1.00        16
weighted avg       1.00      1.00      1.00        16
```

[546]:
```
PRINT(f'The results of Random Forest Multiclass Classifier model\nusing Morgan␣
 ↪Fingerprints features for UniProt P17301 are:')
print_dict_meaningful(rf_model_tuple_P17301_01_[1])
PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of Random Forest Multiclass Classifier model
using Morgan Fingerprints features for UniProt P17301 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 1.000
precision: 1.000
recall: 1.000
f1_score: 1.000
confusion_matrix: [[4, 0], [0, 12]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Save Random Forest Multicalss Classifier Model using Morgan Fingerprint features for P17301

```
[547]: rf_model_filename = os.path.join('trained models/Random Forest Multiclass␣
         ↪Classifier Models', 'rf_model_P17301_.joblib')
       dump(rf_model_tuple_P17301_01_[0], rf_model_filename)

       PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

XGBoost Multiclass Classifier Model for P17301 with added Morgan Fingerprints Features

```
[543]: weight_dict = 'balanced'

       xgb_model_tuple_P17301_01_ = GenerateXGBoostModel(df=P17301_df_for_training__,␣
         ↪weight_dict=weight_dict)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00        12

    accuracy                           1.00        16
   macro avg       1.00      1.00      1.00        16
weighted avg       1.00      1.00      1.00        16
```

```
[544]: PRINT(f'The results of XGBoost Multiclass Classifier model\nusing Morgan␣
         ↪Fingerprints features for UniProt P17301 are:')
       print_dict_meaningful(xgb_model_tuple_P17301_01_[1])
       PRINT(f'Done.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The results of XGBoost Multiclass Classifier model
using Morgan Fingerprints features for UniProt P17301 are:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
accuracy: 1.000
precision: 1.000
recall: 1.000
f1_score: 1.000
confusion_matrix: [[4, 0], [0, 12]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Done.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Save XGBoost Multicalss Classifier Model using Morgan Fingerprint features for P17301

```
[545]: xgb_model_filename = os.path.join('trained models/XGBoost Multiclass Classifier␣
         ↪Models', 'xgb_model_P17301_.joblib')
       dump(xgb_model_tuple_P17301_01_[0], xgb_model_filename)

       PRINT('Model Saved.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Pick the Best Model for P17301 Protein The result we obtained was as expected for our unbalanced dataset, despite our attempts to generalize it. We will choose the *XGBoost* model with *Morgan Fingerprints* features because it utilizes more hyperparameters during training to better generalize the model. This is necessary for our small and imbalanced dataset.

```
[556]: xgb_P17301_morganf_path = 'trained models\\XGBoost Multiclass Classifier␣
         ↪Models\\xgb_model_P17301_.joblib'
       xgb_P17301_morganf = load(xgb_P17301_morganf_path)
```

```
[557]: final_model_P17301 = os.path.join('trained models/Best Model of each UniProt',␣
         ↪'final_xbg_P17301.joblib')
       dump(xgb_P17301_morganf, final_model_P17301)

       PRINT('Model Saved')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Saved
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.4 Make Prediction On Unseen Dataset for Final Results

Now that we have built, trained, and selected a model for each UniProt target dataset we need, we can generate real-time predictions using our best models.

To achieve this, we begin by extracting sub-datasets from our final dataframe on which we wish to execute predictions. For each sub-dataset, we will perform predictions using its corresponding model that we have built.

Finally, we will combine all the resulting data frames to obtain the final dataframe with the following columns: [SMILES, UniProtTarget, UniProtPartner].

```
[746]: final_df_path = 'data/dataset_for_prediction.csv'
```

```
[747]: f_df
```

```
[747]:                                                    SMILES UniProtTarget
        0     OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…        P13612
        1     C\C=C\[C@@H](CC(=O)O)NC(=O)C[C@@H](CC(C)C)NC(=…        P05556
```

```
2     CN1[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O…        P05106
3      OC(=O)C(CNC(=O)CCCCc1ccc2CCCNc2n1)c3cnc4ccccc4c3        P05106
4     OC(=O)C[C@H](NC(=O)CN1CCC[C@@H](CCC2CCNCC2)C1=…        P05106
…                                                   …          …
4187  COc1cc(\C=C/2\C(=NN(C2=O)c3ccc(cc3)C(=O)O)C)cc…        P05106
4188                      NC(=N)NC(=N)Nc1cccc(Cl)c1Cl        P05106
4189                    NC(=N)NC(=N)Nc1ccc(SC(F)F)cc1        P05106
4190     NCCc1ccc(cc1)C(=O)NCC(=O)N2CCN(CC(=O)O)C(=O)C2        P05106
4191  NC(=N)c1ccc(cc1)C(=O)NCC(=O)N2CCN(CC(=O)O)C(=O)C2        P05106

[4192 rows x 2 columns]
```

[748]:
```python
f_df = pd.read_csv(final_df_path)

PRINT(f'Loaded the final data frame')
f_df.head(10)
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Loaded the final data frame
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[748]:
```
                                              smiles uniprot_id1
0  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…      P13612
1  C\C=C\[C@@H](CC(=O)O)NC(=O)C[C@@H](CC(C)C)NC(=…      P05556
2  CN1[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O…      P05106
3   OC(=O)C(CNC(=O)CCCCc1ccc2CCCNc2n1)c3cnc4ccccc4c3      P05106
4  OC(=O)C[C@H](NC(=O)CN1CCC[C@@H](CCC2CCNCC2)C1=…      P05106
5  N[C@@H](Cc1ccc(O)cc1)C(=O)N[C@H]2CSSC[C@H](NC(…      P05556
6  CC(=O)N1CSC[C@@H]1C(=O)N[C@@H](Cc2ccc(OC(=O)C3…      P05556
7  CC1CCC(C[C@H](NC(=O)[C@@H]2CCC(=O)N2Cc3ccccc3)…      P13612
8  COP(=O)(O)[C@@H](CNC(=O)c1ccc(OCCC2CCNCC2)cc1)…      P05106
9  NC(=N)Nc1cccc(c1)C(=O)Nc2ccc(CC(NS(=O)(=O)c3cc…      P05106
```

[749]:
```python
f_df.rename(columns={'uniprot_id1':'UniProtTarget', 'smiles':'SMILES'},␣
 ↪inplace=True)
```

[750]:
```python
f_df.head(3)
```

[750]:
```
                                              SMILES UniProtTarget
0  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…        P13612
1  C\C=C\[C@@H](CC(=O)O)NC(=O)C[C@@H](CC(C)C)NC(=…        P05556
2  CN1[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O…        P05106
```

[751]:
```python
target_dataframes = {}

# Iterate over unique UniProtTarget values
for target_value in f_df['UniProtTarget'].unique():
    # Filter the dataframe for the current UniProtTarget value
```

```
    target_df = f_df[f_df['UniProtTarget'] == target_value].copy()

    target_dataframes[target_value] = target_df
```

```
[752]: target_dataframes.keys()
```

```
[752]: dict_keys(['P13612', 'P05556', 'P05106', 'P05107', 'P08648', 'P17301'])
```

1.4.1 Predict for P13612

```
[1177]: P13612_label_dict = {0: 'P05556', 1: 'P26010'}
```

```
[1178]: P13612_pred = target_dataframes['P13612'].copy()

        P13612_pred.head(5)
```

```
[1178]:                                                 SMILES UniProtTarget
        0   OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…        P13612
        7   CC1CCC(C[C@H](NC(=O)[C@@H]2CCC(=O)N2Cc3ccccc3)…        P13612
        10  CC(C)CCNC(=O)[C@@H]1OCO[C@H]1C(=O)N[C@@H](Cc2c…        P13612
        14  OC(=O)CN(CC(=O)N[C@@H](Cc1ccc(OCc2c(Cl)cccc2Cl…        P13612
        15  CCC\N=C/1\C(\C(=C1O)O)=N\[C@@H](Cc2ccc(OCc3c(C…        P13612
```

```
[1180]: P13612_pred = P13612_pred.reset_index(drop=True)

        PRINT(f'Reseted the indexes of the data frame in order to avoid issues with␣
         ↪features generation')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Reseted the indexes of the data frame in order to avoid issues with features
generation
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1181]: PRINT(f'Shape:\n\n{P13612_pred.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape:

(1100, 2)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1182]: P13612_pred.drop(['UniProtTarget'],axis=1, inplace=True)
```

```
[1183]: P13612_pred['TempColumnForModelTask'] = 0

        PRINT(f'Added dummy column for modele "tasks" variable in order to compile the␣
         ↪model, later going to remove the column')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Added dummy column for modele "tasks" variable in order to compile the model,
later going to remove the column
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1185]:
```
P13612_pred.head(5)
```

[1185]:
```
                                              SMILES  TempColumnForModelTask
0  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…                       0
1  CC1CCC(C[C@H](NC(=O)[C@@H]2CCC(=O)N2Cc3ccccc3)…                       0
2  CC(C)CCNC(=O)[C@@H]1OCO[C@H]1C(=O)N[C@@H](Cc2c…                       0
3  OC(=O)CN(CC(=O)N[C@@H](Cc1ccc(OCc2c(Cl)cccc2Cl…                       0
4  CCC\N=C/1\C(\C(=C1O)O)=N\[C@@H](Cc2ccc(OCc3c(C…                       0
```

[1187]:
```
gcm_P13612 = dc.models.GraphConvModel(model_dir='models/gcm_P13612', n_tasks=1)
gcm_P13612.restore()

PRINT('Model Loaded')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Model Loaded
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1188]:
```
P13612_gc_pred_csv_path = 'data/csv Files for DeepChem GraphConvModel/
  ↪P13612_pred_gc.csv'
```

[1189]:
```
P13612_pred.to_csv(P13612_gc_pred_csv_path, index=False)
```

[1190]:
```
featurizer = dc.feat.ConvMolFeaturizer()
tasks = ['TempColumnForModelTask']
loader = dc.data.CSVLoader(tasks=tasks,
                           smiles_field='SMILES',
                           featurizer=featurizer)

dataset = loader.featurize(P13612_gc_pred_csv_path)
```

```
smiles_field is deprecated and will be removed in a future version of
DeepChem.Use feature_field instead.
C:\Users\gavvi\anaconda3\Lib\site-packages\deepchem\data\data_loader.py:160:
FutureWarning: featurize() is deprecated and has been renamed to
create_dataset().featurize() will be removed in DeepChem 3.0
  warnings.warn(
```

[1191]:
```
predicted_probs = gcm_P13612.predict(dataset)
```

[1192]:
```
PRINT(predicted_probs[:5])
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[[[0.955466   0.044534  ]]
```

```
 [[0.9868268  0.01317321]]

 [[0.90324336 0.09675666]]

 [[0.9160788  0.08392118]]

 [[0.9037799  0.09622007]]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1193]:
```
predicted_labels = get_class_labels(predicted_probs)

PRINT(f'Converted to probs to labeles using helper function !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Converted to probs to labeles using helper function !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1194]:
```
PRINT(predicted_labels[:10])
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[0 0 0 0 0 0 1 0 0 0]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1196]:
```
predictions = predicted_labels
labeled_predictions = [P13612_label_dict[prediction] for prediction in␣
 ↪predictions]
```

[1197]:
```
P13612_pred['PredictedUniProtPartner'] = labeled_predictions
P13612_pred.drop(['TempColumnForModelTask'], axis=1, inplace=True)

PRINT('Merged the Predictions !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Merged the Predictions !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1198]:
```
P13612_pred
```

[1198]:
```
                                                   SMILES  \
0      OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…
1      CC1CCC(C[C@H](NC(=O)[C@@H]2CCC(=O)N2Cc3ccccc3)…
2      CC(C)CCNC(=O)[C@@H]1OCO[C@H]1C(=O)N[C@@H](Cc2c…
3      OC(=O)CN(CC(=O)N[C@@H](Cc1ccc(OCc2c(Cl)cccc2Cl…
4      CCC\N=C/1\C(\C(=C1O)O)=N\[C@@H](Cc2ccc(OCc3c(C…
…                                                    …
1095   COc1ccccc1c2ccc(C[C@H](NC(=O)C3(CCCC3)c4ccc[n+…
1096   COc1ccccc1c2ccc(C[C@H](NC(=O)C3(CCCC3)c4cncc5c…
1097   COc1ccccc1c2ccc(C[C@H](NC(=O)C3(CC=CC3)c4cccnc…
1098   OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cncc2Cl)cc1)NC…
```

```
1099  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cncc2Cl)cc1)NC…

     PredictedUniProtPartner
0                     P05556
1                     P05556
2                     P05556
3                     P05556
4                     P05556
…                        …
1095                  P26010
1096                  P26010
1097                  P26010
1098                  P05556
1099                  P05556

[1100 rows x 2 columns]
```

Looking at the data frame, we observe that our model has succeeded in predicting the smaller class as well. This achievement is notable given that we trained the model on an unbalanced dataset!

```
[1199]: P13612_pred.to_csv(os.path.join('predictions','P13612_pred.csv'), index=False)

        PRINT('Predictions Saved!')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Predictions Saved!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.4.2 Predict for P05556

```
[1217]: P05556_label_dict = {0: 'O75578', 1: 'P05106', 2: 'P06756', 3:'P08648',
                              4: 'P13612', 5:'P17301', 6: 'P23229', 7: 'P56199',
                             8: 'Q13797'}
```

```
[1218]: P05556_pred = target_dataframes['P05556'].copy()

        P05556_pred.head(5)
```

```
[1218]:                                               SMILES UniProtTarget
        1   C\C=C\[C@@H](CC(=O)O)NC(=O)C[C@@H](CC(C)C)NC(=…        P05556
        5   N[C@@H](Cc1ccc(O)cc1)C(=O)N[C@H]2CSSC[C@H](NC(…        P05556
        6   CC(=O)N1CSC[C@@H]1C(=O)N[C@@H](Cc2ccc(OC(=O)C3…        P05556
        13  COc1cc(CN2CCCC2)cc(OC)c1c3ccc(C[C@H](NC(=O)[C@…        P05556
        18  CC(=O)N1CSC[C@@H]1C(=O)N[C@@H](Cc2ccc(OCc3c(Cl…        P05556
```

```
[1219]: PRINT(f'Shepe:\n\n{P05556_pred.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shepe:
```

```
(948, 2)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1220]:
```
P05556_pred = P05556_pred.reset_index(drop=True)

PRINT(f'Reseted the indexes of the data frame in order to avoid issues with␣
 ↪features generation')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Reseted the indexes of the data frame in order to avoid issues with features
generation
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1221]:
```
P05556_pred.drop(['UniProtTarget'],axis=1, inplace=True)
```

[1224]:
```
# Apply the `calculate_descriptors` method in order to generate 8 new features␣
 ↪for df
P05556_pred['MolecularDescriptors'] = P05556_pred['SMILES'].
 ↪apply(calculate_descriptors)

# Transfer the array at each row under the 'MolecularDescriptors' column into␣
 ↪column with their corresponding names & drop the colunn
P05556_pred[['MolWt', 'NumValenceElectrons', 'TPSA', 'MolLogP',␣
 ↪'NumHeteroatoms', 'NumRotatableBonds', 'HeavyAtomCount', 'FractionCSP3']] =␣
 ↪pd.DataFrame(P05556_pred['MolecularDescriptors'].tolist(), index=P05556_pred.
 ↪index)
P05556_pred.drop(columns=['MolecularDescriptors'], axis=1, inplace=True)

P05556_pred = P05556_pred[['SMILES', 'MolWt', 'NumValenceElectrons', 'TPSA',␣
 ↪'MolLogP', 'NumHeteroatoms', 'NumRotatableBonds', 'HeavyAtomCount',␣
 ↪'FractionCSP3']]
```

[1225]:
```
P05556_pred
```

[1225]:
```
                                                SMILES    MolWt  \
0     C\C=C\[C@@H](CC(=O)O)NC(=O)C[C@@H](CC(C)C)NC(=…  522.646
1     N[C@@H](Cc1ccc(O)cc1)C(=O)N[C@H]2CSSC[C@H](NC(…  538.692
2     CC(=O)N1CSC[C@@H]1C(=O)N[C@@H](Cc2ccc(OC(=O)C3…  500.617
3     COc1cc(CN2CCCC2)cc(OC)c1c3ccc(C[C@H](NC(=O)[C@…  704.673
4     CC(=O)N1CSC[C@@H]1C(=O)N[C@@H](Cc2ccc(OCc3c(Cl…  497.400
..                                                 …        …
943   COC(=O)N1C[C@@H](C[C@H]1CNc2ccccn2)OCC(=O)NC[C…  577.660
944   OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cncc2Cl)cc1)NC…  735.597
945   OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cncc2Cl)cc1)NC…  685.590
946   OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cncc2Cl)cc1)NC…  707.543
947   OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cncc2Cl)cc1)NC…  721.570
```

```
     NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  NumRotatableBonds  \
0                    204  136.63  4.63812               9                 13
1                    198  170.85  1.16640              12                  7
2                    188  113.01  2.84170               9                  7
3                    252  125.48  5.62870              13                 12
4                    170   95.94  3.60570              10                  8
..                     …       …        …               …                 …
943                  218  176.26  1.19256              14                 12
944                  258  172.80  4.18718              17                 10
945                  240  181.59  3.59818              15                 11
946                  246  172.80  3.40698              17                 10
947                  252  172.80  3.79708              17                 10

     HeavyAtomCount  FractionCSP3
0                38      0.379310
1                36      0.583333
2                35      0.615385
3                47      0.411765
4                32      0.318182
..              …           …
943              40      0.461538
944              49      0.343750
945              46      0.322581
946              47      0.300000
947              48      0.322581

[948 rows x 9 columns]
```

[1227]:
```
PRINT(f'Shape after generating features using RKDirDescriptors feature␣
↪generation:\n\n{P05556_pred_.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape after generating features using RKDirDescriptors feature generation:

(948, 1025)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1228]:
```
rf_P05556 = load('trained models/Best Model of each UniProt/final_rf_P05556.
↪joblib')
```

[1229]:
```
rf_P05556
```

[1229]:
```
RandomForestClassifier(class_weight='balanced', max_depth=10,
                       min_samples_leaf=2, min_samples_split=10,
                       n_estimators=200, random_state=42)
```

[1232]:
```
df_for_model_P05556 = P05556_pred.drop(['SMILES'], axis=1)
```

```
[1233]: df_for_model_P05556.head(2)
```

```
[1233]:      MolWt  NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  \
        0  522.646                  204  136.63  4.63812               9
        1  538.692                  198  170.85  1.16640              12

           NumRotatableBonds  HeavyAtomCount  FractionCSP3
        0                 13              38      0.379310
        1                  7              36      0.583333
```

```
[1234]: predictions = rf_P05556.predict(df_for_model_P05556)

        PRINT(f'Finished predicting on unseen data.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished predicting on unseen data.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1236]: PRINT(f'Prediction shape: {predictions.shape}\n\nVisualize few predictions:
          ↪\n\n{predictions[:60]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Prediction shape: (948,)

Visualize few predictions:

[4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 3 4 4 4 4 4 4 4 4 4 4 4 3 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 2 4 4 3 4 4 3 4 4 4 4 4]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

In analyzing our model predictions on unseen data, we observe that the model successfully predicts classes with significantly low and unbalanced distributions in the dataset on which we trained the model. This suggests that our model excels in generalizing to identify even the smaller classes and predicting their protein-protein interactions (PPI)

```
[1237]: labeled_predictions = [P05556_label_dict[prediction] for prediction in␣
          ↪predictions]
```

```
[1240]: PRINT(f'Labeled prediction (UniProt):\n\n{labeled_predictions[:15]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Labeled prediction (UniProt):

['P13612', 'P13612', 'P13612', 'P13612', 'P13612', 'P13612', 'P13612', 'P13612',
'P13612', 'P13612', 'P13612', 'P13612', 'P13612', 'P13612', 'P13612']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1241]: P05556_pred['PredictedUniProtPartner'] = labeled_predictions
```

```
PRINT('Merged the Predictions !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Merged the Predictions !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1242]:
```
P05556_pred
```

[1242]:
```
                                                SMILES    MolWt  \
0    C\C=C\[C@@H](CC(=O)O)NC(=O)C[C@@H](CC(C)C)NC(=…  522.646
1    N[C@@H](Cc1ccc(O)cc1)C(=O)N[C@H]2CSSC[C@H](NC(…  538.692
2    CC(=O)N1CSC[C@@H]1C(=O)N[C@@H](Cc2ccc(OC(=O)C3…  500.617
3    COc1cc(CN2CCCC2)cc(OC)c1c3ccc(C[C@H](NC(=O)[C@…  704.673
4    CC(=O)N1CSC[C@@H]1C(=O)N[C@@H](Cc2ccc(OCc3c(Cl…  497.400
..                                                 …        …
943  COC(=O)N1C[C@@H](C[C@H]1CNc2ccccn2)OCC(=O)NC[C…  577.660
944  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cncc2Cl)cc1)NC…  735.597
945  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cncc2Cl)cc1)NC…  685.590
946  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cncc2Cl)cc1)NC…  707.543
947  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cncc2Cl)cc1)NC…  721.570

     NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  NumRotatableBonds  \
0                    204  136.63  4.63812               9                 13
1                    198  170.85  1.16640              12                  7
2                    188  113.01  2.84170               9                  7
3                    252  125.48  5.62870              13                 12
4                    170   95.94  3.60570              10                  8
..                     …       …        …               …                  …
943                  218  176.26  1.19256              14                 12
944                  258  172.80  4.18718              17                 10
945                  240  181.59  3.59818              15                 11
946                  246  172.80  3.40698              17                 10
947                  252  172.80  3.79708              17                 10

     HeavyAtomCount  FractionCSP3 PredictedUniProtPartner
0                38      0.379310                  P13612
1                36      0.583333                  P13612
2                35      0.615385                  P13612
3                47      0.411765                  P13612
4                32      0.318182                  P13612
..                …           …                       …
943              40      0.461538                  P06756
944              49      0.343750                  P13612
945              46      0.322581                  P13612
946              47      0.300000                  P13612
947              48      0.322581                  P13612

[948 rows x 10 columns]
```

```
[1243]: P05556_pred.to_csv(os.path.join('predictions','P05556_pred.csv'), index=False)

        PRINT('Predictions Saved!')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Predictions Saved!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.4.3 Predict for P05106

```
[1154]: P05106_label_dict = {0: 'P05556', 1: 'P06756', 2: 'P08514', 3: 'P17301', 4:␣
          ↪'26006'}
```

```
[1155]: P05106_pred = target_dataframes['P05106'].copy()


        P05106_pred
```

```
[1155]:                                                   SMILES UniProtTarget
        2     CN1[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O…        P05106
        3      OC(=O)C(CNC(=O)CCCCc1ccc2CCCNc2n1)c3cnc4ccccc4c3        P05106
        4     OC(=O)C[C@H](NC(=O)CN1CCC[C@@H](CCC2CCNCC2)C1=…        P05106
        8     COP(=O)(O)[C@@H](CNC(=O)c1ccc(OCCC2CCNCC2)cc1)…        P05106
        9     NC(=N)Nc1cccc(c1)C(=O)Nc2ccc(CC(NS(=O)(=O)c3cc…        P05106
        …                                                   …           …
        4187  COc1cc(\C=C/2\C(=NN(C2=O)c3ccc(cc3)C(=O)O)C)cc…        P05106
        4188                        NC(=N)NC(=N)Nc1cccc(Cl)c1Cl        P05106
        4189                      NC(=N)NC(=N)Nc1ccc(SC(F)F)cc1        P05106
        4190     NCCc1ccc(cc1)C(=O)NCC(=O)N2CCN(CC(=O)O)C(=O)C2        P05106
        4191  NC(=N)c1ccc(cc1)C(=O)NCC(=O)N2CCN(CC(=O)O)C(=O)C2        P05106

        [1727 rows x 2 columns]
```

```
[1156]: P05106_pred = P05106_pred.reset_index(drop=True)

        PRINT(f'Reseted the indexes of the data frame in order to avoid issues with␣
          ↪features generation')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Reseted the indexes of the data frame in order to avoid issues with features
generation
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1157]: PRINT(f'Shape:\n\n{P05106_pred.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape:
```

```
(1727, 2)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1158]: P05106_pred.drop(['UniProtTarget'],axis=1, inplace=True)
```

```
[1159]: P05106_pred_ =␣
         ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P05106_pred,␣
         ↪size=1024, radius=2)
```

```
[1160]: P05106_pred_
```

```
[1160]:                                                     SMILES  Feature_0  Feature_1  \
        0     CN1[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O…        0.0        0.0
        1      OC(=O)C(CNC(=O)CCCCc1ccc2CCCNc2n1)c3cnc4ccccc4c3        0.0        1.0
        2     OC(=O)C[C@H](NC(=O)CN1CCC[C@@H](CCC2CCNCC2)C1=…        0.0        1.0
        3     COP(=O)(O)[C@@H](CNC(=O)c1ccc(OCCC2CCNCC2)cc1)…        0.0        1.0
        4     NC(=N)Nc1cccc(c1)C(=O)Nc2ccc(CC(NS(=O)(=O)c3cc…        0.0        1.0
        …                                                   …          …          …
        1722  COc1cc(\C=C/2\C(=NN(C2=O)c3ccc(cc3)C(=O)O)C)cc…        0.0        0.0
        1723                       NC(=N)NC(=N)Nc1cccc(Cl)c1Cl        0.0        0.0
        1724                     NC(=N)NC(=N)Nc1ccc(SC(F)F)cc1        0.0        1.0
        1725     NCCc1ccc(cc1)C(=O)NCC(=O)N2CCN(CC(=O)O)C(=O)C2        0.0        1.0
        1726  NC(=N)c1ccc(cc1)C(=O)NCC(=O)N2CCN(CC(=O)O)C(=O)C2        0.0        1.0

              Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  Feature_7  \
        0           0.0        0.0        0.0        0.0        0.0        1.0
        1           0.0        0.0        1.0        0.0        0.0        0.0
        2           0.0        0.0        1.0        1.0        0.0        0.0
        3           0.0        0.0        0.0        0.0        0.0        0.0
        4           0.0        0.0        0.0        0.0        0.0        0.0
        …             …          …          …          …          …          …
        1722        0.0        0.0        0.0        0.0        0.0        0.0
        1723        0.0        0.0        0.0        0.0        0.0        0.0
        1724        0.0        0.0        0.0        0.0        0.0        0.0
        1725        0.0        0.0        0.0        0.0        0.0        0.0
        1726        0.0        0.0        0.0        0.0        0.0        0.0

              Feature_8  …  Feature_1014  Feature_1015  Feature_1016  Feature_1017  \
        0           0.0  …           0.0           0.0           0.0           0.0
        1           0.0  …           0.0           0.0           0.0           1.0
        2           0.0  …           0.0           0.0           0.0           0.0
        3           0.0  …           0.0           0.0           0.0           0.0
        4           0.0  …           0.0           0.0           0.0           0.0
        …             …  …             …             …             …             …
        1722        0.0  …           0.0           0.0           0.0           0.0
        1723        0.0  …           0.0           0.0           0.0           0.0
        1724        0.0  …           0.0           0.0           0.0           0.0
        1725        0.0  …           0.0           0.0           0.0           0.0
```

```
1726        0.0  …          0.0          0.0          0.0          0.0

      Feature_1018  Feature_1019  Feature_1020  Feature_1021  Feature_1022  \
0              0.0           1.0           0.0           0.0           0.0
1              0.0           0.0           0.0           0.0           0.0
2              0.0           1.0           0.0           0.0           0.0
3              0.0           1.0           0.0           0.0           0.0
4              0.0           1.0           0.0           0.0           0.0
…              …            …            …            …            …
1722           0.0           0.0           0.0           0.0           0.0
1723           0.0           0.0           0.0           0.0           0.0
1724           0.0           0.0           0.0           0.0           0.0
1725           0.0           0.0           0.0           0.0           0.0
1726           0.0           0.0           0.0           0.0           0.0

      Feature_1023
0              0.0
1              0.0
2              0.0
3              0.0
4              0.0
…              …
1722           0.0
1723           0.0
1724           0.0
1725           0.0
1726           0.0

[1727 rows x 1025 columns]
```

[1161]:
```
PRINT(f'Shape after generating features using Morgan Fingerprints:
↪\n\n{P05106_pred_.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape after generating features using Morgan Fingerprints:

(1727, 1025)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1162]:
```
xgb_P05106 = load('trained models/Best Model of each UniProt/final_xgb_P05106.
↪joblib')
```

[1163]:
```
xgb_P05106
```

[1163]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
```

```
                gamma=0.2, grow_policy=None, importance_type=None,
                interaction_constraints=None, learning_rate=0.01, max_bin=None,
                max_cat_threshold=None, max_cat_to_onehot=None,
                max_delta_step=None, max_depth=5, max_leaves=None,
                min_child_weight=5, missing=nan, monotone_constraints=None,
                multi_strategy=None, n_estimators=100, n_jobs=None, num_class=5,
                num_parallel_tree=None, …)
```

```
[1164]: df_for_model_P05106 = P05106_pred_.drop(['SMILES'], axis=1)
```

```
[1165]: df_for_model_P05106.head(2)
```

```
[1165]:    Feature_0  Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  \
        0        0.0        0.0        0.0        0.0        0.0        0.0
        1        0.0        1.0        0.0        0.0        1.0        0.0

           Feature_6  Feature_7  Feature_8  Feature_9  …  Feature_1014  \
        0        0.0        1.0        0.0        0.0  …           0.0
        1        0.0        0.0        0.0        0.0  …           0.0

           Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
        0           0.0           0.0           0.0           0.0           1.0
        1           0.0           0.0           1.0           0.0           0.0

           Feature_1020  Feature_1021  Feature_1022  Feature_1023
        0           0.0           0.0           0.0           0.0
        1           0.0           0.0           0.0           0.0

        [2 rows x 1024 columns]
```

```
[1166]: predictions = xgb_P05106.predict(df_for_model_P05106)

        PRINT(f'Finished predicting on unseen data.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished predicting on unseen data.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1167]: PRINT(f'Prediction shape: {predictions.shape}\n\nVisualize few predictions:
          ↪\n\n{predictions[:30]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Prediction shape: (1727,)

Visualize few predictions:

[2 1 2 2 1 1 1 2 2 1 2 1 1 1 2 2 1 1 1 1 1 2 1 1 2 2 1 1 2 1]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1168]: labeled_predictions = [P05106_label_dict[prediction] for prediction in␣
         ↪predictions]
```

```
[1172]: PRINT(f'Labeled predictions (UniProt):\n\n{labeled_predictions[:15]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Labeled predictions (UniProt):

['P08514', 'P06756', 'P08514', 'P08514', 'P06756', 'P06756', 'P06756', 'P08514',
'P08514', 'P06756', 'P08514', 'P06756', 'P06756', 'P06756', 'P08514']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1173]: P05106_pred['PredictedUniProtPartner'] = labeled_predictions

        PRINT('Merged the Predictions !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Merged the Predictions !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1174]: P05106_pred
```

```
[1174]:
                                                 SMILES  \
0     CN1[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O…
1      OC(=O)C(CNC(=O)CCCCc1ccc2CCCNc2n1)c3cnc4ccccc4c3
2     OC(=O)C[C@H](NC(=O)CN1CCC[C@@H](CCC2CCNCC2)C1=…
3     COP(=O)(O)[C@@H](CNC(=O)c1ccc(OCCC2CCNCC2)cc1)…
4     NC(=N)Nc1cccc(c1)C(=O)Nc2ccc(CC(NS(=O)(=O)c3cc…
…                                                   …
1722  COc1cc(\C=C/2\C(=NN(C2=O)c3ccc(cc3)C(=O)O)C)cc…
1723                             NC(=N)NC(=N)Nc1cccc(Cl)c1Cl
1724                           NC(=N)NC(=N)Nc1ccc(SC(F)F)cc1
1725     NCCc1ccc(cc1)C(=O)NCC(=O)N2CCN(CC(=O)O)C(=O)C2
1726  NC(=N)c1ccc(cc1)C(=O)NCC(=O)N2CCN(CC(=O)O)C(=O)C2

     PredictedUniProtPartner
0                     P08514
1                     P06756
2                     P08514
3                     P08514
4                     P06756
…                      …
1722                  P06756
1723                  P06756
1724                  P06756
1725                  P08514
1726                  P08514
```

```
[1727 rows x 2 columns]
```

```
[1175]: P05106_pred.to_csv(os.path.join('predictions','P05106_pred.csv'), index=False)

        PRINT('Predictions Saved!')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Predictions Saved!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.4.4 Predict for P05107

```
[1128]: P05107_label_dict = {0: 'P11215', 1: 'P20701'}
```

```
[1129]: P05107_pred = target_dataframes['P05107'].copy()

        P05107_pred.head(5)
```

```
[1129]:                                                SMILES UniProtTarget
        28   OC(=O)[C@@H]1CCCN1c2cc(ccn2)c3ccc(Sc4ccc5OCCOc…        P05107
        47   CN([C@@H]1CCN(C1)c2cc(ccn2)c3ccc(Sc4ccc5OCCOc5…        P05107
        57   CC(C)c1ccccc1Sc2ccc(cc2C(F)(F)F)c3cc(ncn3)N4CC…        P05107
        84   CNC(=O)[C@H](Cc1ccc2ccccc2c1)N3CC(=O)N(Cc4ccc(…        P05107
        107       OC(=O)[C@H](Cc1cccc2ccccc12)NC(=O)c3ccccc3Br        P05107
```

```
[1130]: PRINT(f'Shepe:\n\n{P05107_pred.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shepe:

(339, 2)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1131]: P05107_pred = P05107_pred.reset_index(drop=True)

        PRINT(f'Reseted the indexes of the data frame in order to avoid issues with␣
         ↪features generation')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Reseted the indexes of the data frame in order to avoid issues with features
generation
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1132]: P05107_pred.drop(['UniProtTarget'],axis=1, inplace=True)
```

```
[1133]: P05107_pred_ =␣
         ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P05107_pred,␣
         ↪size=1024, radius=2)
```

```
[1134]: P05107_pred_
```

```
[1134]:                                                  SMILES  Feature_0  Feature_1  \
        0    OC(=O)[C@@H]1CCCN1c2cc(ccn2)c3ccc(Sc4ccc5OCCOc…        0.0        1.0
        1    CN([C@@H]1CCN(C1)c2cc(ccn2)c3ccc(Sc4ccc5OCCOc5…        0.0        1.0
        2    CC(C)c1ccccc1Sc2ccc(cc2C(F)(F)F)c3cc(ncn3)N4CC…        0.0        1.0
        3    CNC(=O)[C@H](Cc1ccc2ccccc2c1)N3CC(=O)N(Cc4ccc(…        0.0        1.0
        4          OC(=O)[C@H](Cc1cccc2ccccc12)NC(=O)c3ccccc3Br        0.0        1.0
        ..                                                 …        …          …
        334  CCOC(=O)CN1CCN(CC1)c2cc(ncn2)c3ccc(Sc4ccccc4C(…        0.0        1.0
        335  COc1ccccc1Sc2ccc(cc2C(F)(F)F)c3ccnc(c3)N4CC[C@…        0.0        1.0
        336  CNC(=O)[C@@H](Cc1ccc2ccccc2c1)N3CCC(=O)N(Cc4cn…        0.0        1.0
        337  CNC(=O)[C@H](Cc1ccc2ccccc2c1)N3CCC(=O)N(CCc4cn…        0.0        1.0
        338  CNC(=O)[C@H](Cc1ccc2ccccc2c1)N3CCC(=O)N(Cc4ccc…        0.0        1.0

             Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  Feature_7  \
        0          0.0        0.0        1.0        0.0        0.0        0.0
        1          0.0        0.0        0.0        0.0        0.0        0.0
        2          0.0        0.0        0.0        0.0        0.0        0.0
        3          0.0        0.0        0.0        1.0        0.0        0.0
        4          0.0        0.0        0.0        0.0        0.0        0.0
        ..           …          …          …          …          …          …
        334        0.0        0.0        0.0        0.0        0.0        0.0
        335        0.0        0.0        0.0        0.0        0.0        0.0
        336        0.0        0.0        0.0        1.0        0.0        0.0
        337        0.0        0.0        0.0        1.0        0.0        0.0
        338        0.0        0.0        0.0        1.0        0.0        0.0

             Feature_8  …  Feature_1014  Feature_1015  Feature_1016  Feature_1017  \
        0          0.0  …           0.0           0.0           0.0           0.0
        1          0.0  …           0.0           0.0           0.0           1.0
        2          0.0  …           1.0           0.0           0.0           0.0
        3          0.0  …           0.0           1.0           0.0           0.0
        4          0.0  …           0.0           0.0           0.0           0.0
        ..           …  …             …             …             …             …
        334        0.0  …           0.0           0.0           0.0           0.0
        335        0.0  …           0.0           0.0           0.0           1.0
        336        0.0  …           0.0           1.0           0.0           0.0
        337        0.0  …           0.0           1.0           0.0           0.0
        338        0.0  …           0.0           1.0           0.0           0.0

             Feature_1018  Feature_1019  Feature_1020  Feature_1021  Feature_1022  \
        0             0.0           1.0           0.0           0.0           0.0
        1             0.0           1.0           0.0           0.0           0.0
        2             0.0           0.0           0.0           0.0           0.0
        3             0.0           1.0           0.0           0.0           0.0
        4             0.0           0.0           0.0           0.0           0.0
```

```
..             …           …           …           …           …
334            0.0            0.0            0.0            0.0            0.0
335            0.0            1.0            0.0            0.0            0.0
336            0.0            1.0            0.0            0.0            0.0
337            0.0            1.0            0.0            0.0            0.0
338            0.0            1.0            0.0            0.0            0.0

     Feature_1023
0             0.0
1             0.0
2             0.0
3             0.0
4             0.0
..            …
334           0.0
335           0.0
336           0.0
337           0.0
338           0.0

[339 rows x 1025 columns]
```

```
[1135]: PRINT(f'Shape after generating features using Morgan Fingerprints:
          ↪\n\n{P05107_pred_.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape after generating features using Morgan Fingerprints:

(339, 1025)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1136]: P05107_pred_.dropna(axis=0, inplace=True)
```

```
[1137]: PRINT(f'Shape after dropping:\n\n{P05107_pred_.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape after dropping:

(339, 1025)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1138]: xgb_P05107 = load('trained models/Best Model of each UniProt/final_xgb_P05107.
          ↪joblib')
```

```
[1139]: xgb_P05107
```

```
[1139]: XGBClassifier(base_score=None, booster=None, callbacks=None,
                      colsample_bylevel=None, colsample_bynode=None,
```

```
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.01, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
              min_child_weight=1, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=50, n_jobs=None, num_class=2,
              num_parallel_tree=None, …)
```

[1140]:
```
df_for_model_P05107 = P05107_pred_.drop(['SMILES'], axis=1)
```

[1141]:
```
df_for_model_P05107.head(2)
```

[1141]:
```
   Feature_0  Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  \
0        0.0        1.0        0.0        0.0        1.0        0.0
1        0.0        1.0        0.0        0.0        0.0        0.0

   Feature_6  Feature_7  Feature_8  Feature_9  …  Feature_1014  \
0        0.0        0.0        0.0        1.0  …           0.0
1        0.0        0.0        0.0        1.0  …           0.0

   Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
0           0.0           0.0           0.0           0.0           1.0
1           0.0           0.0           1.0           0.0           1.0

   Feature_1020  Feature_1021  Feature_1022  Feature_1023
0           0.0           0.0           0.0           0.0
1           0.0           0.0           0.0           0.0

[2 rows x 1024 columns]
```

[1142]:
```
predictions = xgb_P05107.predict(df_for_model_P05107)

PRINT(f'Finished predicting on unseen data.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished predicting on unseen data.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1143]:
```
PRINT(f'Prediction shape: {predictions.shape}\n\nVisualize few predictions:
  ↪\n\n{predictions[:50]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Prediction shape: (339,)

Visualize few predictions:
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1
 1 1 1 1 0 0 0 1 1 1 1 1 1]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

As we can see, our model indeed identifies both classes in the predictions! This is a positive outcome, demonstrating the model's ability to accurately discern between the specified classes.

[1144]:
```
labeled_predictions = [P05107_label_dict[prediction] for prediction in␣
↪predictions]
```

[1147]:
```
PRINT(f'Labeled prediction (UniProt):\n\n{labeled_predictions[:15]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Labeled prediction (UniProt):

['P20701', 'P20701', 'P20701', 'P20701', 'P20701', 'P20701', 'P20701', 'P20701',
'P20701', 'P20701', 'P20701', 'P20701', 'P20701', 'P20701', 'P20701']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1148]:
```
P05107_pred['PredictedUniProtPartner'] = labeled_predictions

PRINT('Merged the Predictions !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Merged the Predictions !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1152]:
```
P05107_pred
```

[1152]:
```
                                                SMILES PredictedUniProtPartner
0     OC(=O)[C@@H]1CCCN1c2cc(ccn2)c3ccc(Sc4ccc5OCCOc…                  P20701
1     CN([C@@H]1CCN(C1)c2cc(ccn2)c3ccc(Sc4ccc5OCCOc5…                  P20701
2     CC(C)c1ccccc1Sc2ccc(cc2C(F)(F)F)c3cc(ncn3)N4CC…                  P20701
3     CNC(=O)[C@H](Cc1ccc2ccccc2c1)N3CC(=O)N(Cc4ccc(…                  P20701
4          OC(=O)[C@H](Cc1cccc2ccccc12)NC(=O)c3ccccc3Br                  P20701
..                                                 …                      …
334   CCOC(=O)CN1CCN(CC1)c2cc(ncn2)c3ccc(Sc4ccccc4C(…                  P20701
335   COc1ccccc1Sc2ccc(cc2C(F)(F)F)c3ccnc(c3)N4CC[C@…                  P20701
336   CNC(=O)[C@@H](Cc1ccc2ccccc2c1)N3CCC(=O)N(Cc4cn…                  P20701
337   CNC(=O)[C@H](Cc1ccc2ccccc2c1)N3CCC(=O)N(CCc4cn…                  P20701
338   CNC(=O)[C@H](Cc1ccc2ccccc2c1)N3CCC(=O)N(Cc4ccc…                  P20701

[339 rows x 2 columns]
```

[1153]:
```
P05107_pred.to_csv(os.path.join('predictions','P05107_pred.csv'), index=False)

PRINT('Predictions Saved!')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Predictions Saved!
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.4.5 Predict for P08648

```
[1118]: P08648_label_dict = {0: 'P05556', 1: 'P06756'}
```

```
[1104]: P08648_pred = target_dataframes['P08648'].copy()

        P08648_pred.head(5)
```

```
[1104]:                                                SMILES UniProtTarget
        245  Cc1cc(C)c(c(C)c1)S(=O)(=O)N[C@@H](CNC(=O)C2=NO…        P08648
        289  OC(=O)[C@H](Cc1cccc(OCCNc2ccccn2)c1)NC(=O)c3c(…        P08648
        339  CCCN(C(=O)CC(=O)O)C1=C(C)C[C@H](N([C@@H](C)c2c…        P08648
        361  CC1(CCCCC1)C(=O)N[C@@H](CNC(=O)CO[C@@H]2C[C@@H…        P08648
        364  OC(=O)[C@H](Cc1cccc(OCCCCNc2ccccn2)c1)NC(=O)c3…        P08648
```

```
[1105]: P08648_pred = P08648_pred.reset_index(drop=True)

        PRINT(f'Reseted the indexes of the data frame in order to avoid issues with␣
          ↪features generation')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Reseted the indexes of the data frame in order to avoid issues with features
generation
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1106]: PRINT(f'Visualize data frame shape: {P08648_pred.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Visualize data frame shape: (76, 2)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1107]: P08648_pred.drop(['UniProtTarget'],axis=1, inplace=True)
```

```
[1108]: P08648_pred_ =␣
          ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P08648_pred,␣
          ↪size=1024, radius=2)
```

```
[1109]: P08648_pred_
```

```
[1109]:                                               SMILES  Feature_0  Feature_1  \
        0   Cc1cc(C)c(c(C)c1)S(=O)(=O)N[C@@H](CNC(=O)C2=NO…        0.0        1.0
        1   OC(=O)[C@H](Cc1cccc(OCCNc2ccccn2)c1)NC(=O)c3c(…        0.0        1.0
        2   CCCN(C(=O)CC(=O)O)C1=C(C)C[C@H](N([C@@H](C)c2c…        0.0        1.0
        3   CC1(CCCCC1)C(=O)N[C@@H](CNC(=O)CO[C@@H]2C[C@@H…        0.0        1.0
        4   OC(=O)[C@H](Cc1cccc(OCCCCNc2ccccn2)c1)NC(=O)c3…        0.0        1.0
        ..                                               …          …          …
        71  NC(=N)NCCCNC(=O)[C@@H]1CCC(=O)N2C[C@H](CCC(=O)…        0.0        0.0
```

```
72  NC(=N)NCCCNC(=O)[C@H]1CCC(=O)N2C[C@H](CCC(=O)O…        0.0        0.0
73  Cc1cc(C)c(C(=O)N[C@@H](CNC(=O)CO[C@@H]2C[C@@H]…        0.0        1.0
74  Cc1cc(C)c(c(C)c1)S(=O)(=O)N[C@@H](CNC(=O)CO[C@…        0.0        1.0
75  OC(=O)[C@H](CNC(=O)CO[C@@H]1C[C@@H](CNc2ccccn2…        0.0        1.0

    Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  Feature_7  \
0         0.0        0.0        0.0        0.0        0.0        0.0
1         0.0        0.0        0.0        0.0        0.0        0.0
2         0.0        0.0        0.0        0.0        0.0        0.0
3         1.0        0.0        1.0        0.0        0.0        0.0
4         0.0        0.0        0.0        0.0        0.0        0.0
..        …        …        …        …        …        …
71        0.0        0.0        0.0        0.0        0.0        0.0
72        0.0        0.0        0.0        0.0        0.0        0.0
73        0.0        0.0        0.0        0.0        0.0        0.0
74        0.0        0.0        0.0        0.0        0.0        0.0
75        0.0        0.0        0.0        0.0        0.0        0.0

    Feature_8  …  Feature_1014  Feature_1015  Feature_1016  Feature_1017  \
0         0.0  …           0.0           0.0           0.0           0.0
1         0.0  …           0.0           0.0           0.0           0.0
2         0.0  …           0.0           0.0           0.0           0.0
3         0.0  …           0.0           0.0           0.0           0.0
4         0.0  …           0.0           0.0           0.0           0.0
..        …  …           …           …           …           …
71        0.0  …           0.0           0.0           0.0           0.0
72        0.0  …           0.0           0.0           0.0           0.0
73        0.0  …           0.0           0.0           0.0           0.0
74        0.0  …           0.0           0.0           0.0           0.0
75        0.0  …           0.0           0.0           0.0           0.0

    Feature_1018  Feature_1019  Feature_1020  Feature_1021  Feature_1022  \
0            1.0           1.0           0.0           1.0           0.0
1            0.0           0.0           0.0           0.0           0.0
2            0.0           1.0           0.0           0.0           0.0
3            1.0           1.0           0.0           0.0           0.0
4            0.0           0.0           0.0           0.0           0.0
..           …           …           …           …           …
71           0.0           1.0           1.0           0.0           0.0
72           0.0           1.0           1.0           0.0           0.0
73           1.0           1.0           0.0           0.0           0.0
74           0.0           1.0           0.0           1.0           0.0
75           1.0           1.0           0.0           0.0           0.0

    Feature_1023
0            0.0
1            0.0
```

```
2              0.0
3              0.0
4              0.0
..             …
71             0.0
72             0.0
73             0.0
74             0.0
75             0.0

[76 rows x 1025 columns]
```

[1110]:
```
PRINT(f'Shape after generating features using Morgan Fingerprints:
 ↪\n\n{P08648_pred_.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape after generating features using Morgan Fingerprints:

(76, 1025)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1111]:
```
rf_P08648 = load('trained models/Best Model of each UniProt/final_rf_P08648.
 ↪joblib')
```

[1112]:
```
rf_P08648
```

[1112]:
```
RandomForestClassifier(class_weight='balanced', min_samples_split=15,
                       n_estimators=50, random_state=42)
```

[1113]:
```
df_for_model_P08648 = P08648_pred_.drop(['SMILES'], axis=1)
```

[1114]:
```
df_for_model_P08648.head(2)
```

[1114]:
```
   Feature_0  Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  \
0        0.0        1.0        0.0        0.0        0.0        0.0
1        0.0        1.0        0.0        0.0        0.0        0.0

   Feature_6  Feature_7  Feature_8  Feature_9  …  Feature_1014  \
0        0.0        0.0        0.0        0.0  …           0.0
1        0.0        0.0        0.0        0.0  …           0.0

   Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
0           0.0           0.0           0.0           1.0           1.0
1           0.0           0.0           0.0           0.0           0.0

   Feature_1020  Feature_1021  Feature_1022  Feature_1023
0           0.0           1.0           0.0           0.0
1           0.0           0.0           0.0           0.0
```

```
[2 rows x 1024 columns]
```

[1115]:
```
predictions = rf_P08648.predict(df_for_model_P08648)

PRINT(f'Finished predicting on unseen data.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished predicting on unseen data.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1116]:
```
PRINT(f'Prediction shape: {predictions.shape}\n\nVisualize few predictions:
  ↪\n\n{predictions[:30]}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Prediction shape: (76,)

Visualize few predictions:

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1117]:
```
predictions = [int(value) for value in predictions]

PRINT(predictions)
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

As expected, although we attempted to balance the model using techniques such as assigning weight to the smaller class, our dataset was severely unbalanced for generalization with only two classes:

- Number of times P05556 appears -> 463
- Number of times P0756 appears -> 6

As a result, the model performed as anticipated on our small and unbalanced dataset.

[1119]:
```
labeled_predictions = [P08648_label_dict[prediction] for prediction in␣
  ↪predictions]
```

[1123]:
```
P08648_pred['PredictedUniProtPartner'] = labeled_predictions

PRINT('Merged the Predictions !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Merged the Predictions !
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[1124]:
```
P08648_pred
```

[1124]:
```
                                               SMILES PredictedUniProtPartner
0   Cc1cc(C)c(c(C)c1)S(=O)(=O)N[C@@H](CNC(=O)C2=NO…                  P05556
1   OC(=O)[C@H](Cc1cccc(OCCNc2ccccn2)c1)NC(=O)c3c(…                  P05556
2   CCCN(C(=O)CC(=O)O)C1=C(C)C[C@H](N([C@@H](C)c2c…                  P05556
3   CC1(CCCCC1)C(=O)N[C@@H](CNC(=O)CO[C@@H]2C[C@@H…                  P05556
4   OC(=O)[C@H](Cc1cccc(OCCCCNc2ccccn2)c1)NC(=O)c3…                  P05556
..                                                ...                     ...
71  NC(=N)NCCCNC(=O)[C@@H]1CCC(=O)N2C[C@H](CCC(=O)…                  P05556
72  NC(=N)NCCCNC(=O)[C@H]1CCC(=O)N2C[C@H](CCC(=O)O…                  P05556
73  Cc1cc(C)c(C(=O)N[C@@H](CNC(=O)CO[C@@H]2C[C@@H]…                  P05556
74  Cc1cc(C)c(c(C)c1)S(=O)(=O)N[C@@H](CNC(=O)CO[C@…                  P05556
75  OC(=O)[C@H](CNC(=O)CO[C@@H]1C[C@@H](CNc2ccccn2…                  P05556

[76 rows x 2 columns]
```

[1256]:
```
P08648_pred.to_csv(os.path.join('predictions','P08648_pred.csv'), index=False)

PRINT('Predictions Saved!')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Predictions Saved!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.4.6 Predict for P17301

[1087]:
```
P17301_label_dict = {0: 'P05106', 1: 'P05556'}
```

[1088]:
```
P17301_pred = target_dataframes['P17301'].copy()

P17301_pred
```

[1088]:
```
                                                 SMILES UniProtTarget
1149     Cc1ccccc1S(=O)(=O)N[C@@H](CNC(=O)c2cocc2)C(=O)O        P17301
3327   CCc1cc(O)c2c(O)c3C(=O)c4c(O)cccc4C(=O)c3cc2c1C…        P17301
```

[1089]:
```
P17301_pred = P17301_pred.reset_index(drop=True)

PRINT(f'Reseted the indexes of the data frame in order to avoid issues with␣
  ↪features generation')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Reseted the indexes of the data frame in order to avoid issues with features
generation
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1090]: PRINT(f'Shape:\n\n{P17301_pred.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape:

(2, 2)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1091]: P17301_pred.drop(['UniProtTarget'],axis=1, inplace=True)
```

```
[1092]: P17301_pred_ =␣
  ↪GenerateMorganFingerprintsFeaturesByMoleculeSMILES(df=P17301_pred,␣
  ↪size=1024, radius=2)
```

```
[1093]: P17301_pred_
```

```
[1093]:                                               SMILES  Feature_0  Feature_1  \
0     Cc1ccccc1S(=O)(=O)N[C@@H](CNC(=O)c2cocc2)C(=O)O        0.0        1.0
1   CCc1cc(O)c2c(O)c3C(=O)c4c(O)cccc4C(=O)c3cc2c1C…        0.0        0.0

   Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  Feature_7  \
0        0.0        0.0        0.0        0.0        0.0        0.0
1        0.0        0.0        0.0        0.0        0.0        0.0

   Feature_8  …  Feature_1014  Feature_1015  Feature_1016  Feature_1017  \
0        0.0  …           0.0           0.0           0.0           0.0
1        0.0  …           0.0           0.0           0.0           0.0

   Feature_1018  Feature_1019  Feature_1020  Feature_1021  Feature_1022  \
0           0.0           0.0           0.0           0.0           0.0
1           0.0           0.0           0.0           0.0           0.0

   Feature_1023
0           0.0
1           0.0

[2 rows x 1025 columns]
```

```
[1094]: PRINT(f'Shape after generating features using Morgan Fingerprints:
  ↪\n\n{P17301_pred_.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shape after generating features using Morgan Fingerprints:

(2, 1025)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1095]: xgb_P17301 = load('trained models/Best Model of each UniProt/final_xgb_P17301.
  ↪joblib')
```

```
[1096]: xgb_P17301
```

```
[1096]: XGBClassifier(base_score=None, booster=None, callbacks=None,
                      colsample_bylevel=None, colsample_bynode=None,
                      colsample_bytree=0.8, device=None, early_stopping_rounds=None,
                      enable_categorical=False, eval_metric=None, feature_types=None,
                      gamma=0, grow_policy=None, importance_type=None,
                      interaction_constraints=None, learning_rate=0.1, max_bin=None,
                      max_cat_threshold=None, max_cat_to_onehot=None,
                      max_delta_step=None, max_depth=3, max_leaves=None,
                      min_child_weight=5, missing=nan, monotone_constraints=None,
                      multi_strategy=None, n_estimators=50, n_jobs=None, num_class=2,
                      num_parallel_tree=None, …)
```

```
[1097]: df_for_model_P17301 = P17301_pred_.drop(['SMILES'], axis=1)

        df_for_model_P17301.head(2)
```

```
[1097]:    Feature_0  Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  \
        0        0.0        1.0        0.0        0.0        0.0        0.0
        1        0.0        0.0        0.0        0.0        0.0        0.0

           Feature_6  Feature_7  Feature_8  Feature_9  …  Feature_1014  \
        0        0.0        0.0        0.0        0.0  …           0.0
        1        0.0        0.0        0.0        0.0  …           0.0

           Feature_1015  Feature_1016  Feature_1017  Feature_1018  Feature_1019  \
        0           0.0           0.0           0.0           0.0           0.0
        1           0.0           0.0           0.0           0.0           0.0

           Feature_1020  Feature_1021  Feature_1022  Feature_1023
        0           0.0           0.0           0.0           0.0
        1           0.0           0.0           0.0           0.0

        [2 rows x 1024 columns]
```

```
[1098]: predictions = xgb_P17301.predict(df_for_model_P17301)

        PRINT(f'Finished predicting on unseen data.')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Finished predicting on unseen data.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1099]: PRINT(f'Prediction shape: {predictions.shape}\n\nVisualize predictions:
          ↪\n\n{predictions}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Prediction shape: (2,)

Visualize predictions:

[1 1]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1100]: labeled_predictions = [P17301_label_dict[prediction] for prediction in␣
          ↪predictions]
```

```
[1101]: P17301_pred['PredictedUniProtPartner'] = labeled_predictions

        PRINT('Merged the Predictions !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Merged the Predictions !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[1102]: P17301_pred
```

```
[1102]:                                              SMILES PredictedUniProtPartner
         0     Cc1ccccc1S(=O)(=O)N[C@@H](CNC(=O)c2cocc2)C(=O)O                P05556
         1   CCc1cc(O)c2c(O)c3C(=O)c4c(O)cccc4C(=O)c3cc2c1C…                P05556
```

```
[1127]: P17301_pred.to_csv(os.path.join('predictions','P17301_pred.csv'), index=False)

        PRINT('Predictions Saved!')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Predictions Saved!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.5 Putting it all together

After generating a sub-data frame for each unique UniProt we built a model for, and predicting the interactions for every single dataset associated with a given molecule SMILE and the UniProt target of its partner, we can finally combine all the datasets into a single comprehensive data frame.

```
[1244]: prediction_dir = 'predictions'
```

```
[1245]: P13612_df = pd.read_csv(os.path.join(prediction_dir, 'P13612_pred.csv'))
```

```
[1246]: P13612_df['UniProtTarget'] = 'P13612'
```

```
[1247]: P13612_df.head(3)
```

```
[1247]:                                               SMILES PredictedUniProtPartner  \
        0  OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…                  P05556
        1  CC1CCC(C[C@H](NC(=O)[C@@H]2CCC(=O)N2Cc3ccccc3)…                  P05556
        2  CC(C)CCNC(=O)[C@@H]1OCO[C@H]1C(=O)N[C@@H](Cc2c…                  P05556

          UniProtTarget
        0        P13612
        1        P13612
        2        P13612
```

```
[1248]: P05556_df = pd.read_csv(os.path.join(prediction_dir, 'P05556_pred.csv'))
        P05556_df['UniProtTarget'] = 'P05556'
        P05556_df.head(3)
```

```
[1248]:                                               SMILES    MolWt  \
        0  C\C=C\[C@@H](CC(=O)O)NC(=O)C[C@@H](CC(C)C)NC(=…  522.646
        1  N[C@@H](Cc1ccc(O)cc1)C(=O)N[C@H]2CSSC[C@H](NC(…  538.692
        2  CC(=O)N1CSC[C@@H]1C(=O)N[C@@H](Cc2ccc(OC(=O)C3…  500.617

           NumValenceElectrons    TPSA  MolLogP  NumHeteroatoms  NumRotatableBonds  \
        0                  204  136.63  4.63812               9                 13
        1                  198  170.85  1.16640              12                  7
        2                  188  113.01  2.84170               9                  7

           HeavyAtomCount  FractionCSP3 PredictedUniProtPartner UniProtTarget
        0              38      0.379310                  P13612        P05556
        1              36      0.583333                  P13612        P05556
        2              35      0.615385                  P13612        P05556
```

```
[1249]: P05556_df.drop(['MolWt','NumValenceElectrons','TPSA',␣
         ↪'MolLogP','NumHeteroatoms', 'NumRotatableBonds', 'HeavyAtomCount',␣
         ↪'FractionCSP3'], axis=1, inplace=True)
```

```
[1250]: P05556_df.head(3)
```

```
[1250]:                                               SMILES PredictedUniProtPartner  \
        0  C\C=C\[C@@H](CC(=O)O)NC(=O)C[C@@H](CC(C)C)NC(=…                  P13612
        1  N[C@@H](Cc1ccc(O)cc1)C(=O)N[C@H]2CSSC[C@H](NC(…                  P13612
        2  CC(=O)N1CSC[C@@H]1C(=O)N[C@@H](Cc2ccc(OC(=O)C3…                  P13612

          UniProtTarget
        0        P05556
        1        P05556
        2        P05556
```

```
[1251]: P05107_df = pd.read_csv(os.path.join(prediction_dir, 'P05107_pred.csv'))
        P05107_df['UniProtTarget'] = 'P05107'
        P05107_df.head(3)
```

[1251]:
```
                                              SMILES PredictedUniProtPartner  \
0  OC(=O)[C@@H]1CCCN1c2cc(ccn2)c3ccc(Sc4ccc5OCCOc…                  P20701
1  CN([C@@H]1CCN(C1)c2cc(ccn2)c3ccc(Sc4ccc5OCCOc5…                  P20701
2  CC(C)c1ccccc1Sc2ccc(cc2C(F)(F)F)c3cc(ncn3)N4CC…                  P20701

  UniProtTarget
0        P05107
1        P05107
2        P05107
```

[1252]:
```python
P05106_df = pd.read_csv(os.path.join(prediction_dir, 'P05106_pred.csv'))
P05106_df['UniProtTarget'] = 'P05106'
P05106_df.head(3)
```

[1252]:
```
                                              SMILES PredictedUniProtPartner  \
0  CN1[C@@H](CCCN=C(N)N)C(=O)NCC(=O)N[C@@H](CC(=O…                  P08514
1   OC(=O)C(CNC(=O)CCCCc1ccc2CCCNc2n1)c3cnc4ccccc4c3                  P06756
2  OC(=O)C[C@H](NC(=O)CN1CCC[C@@H](CCC2CCNCC2)C1=…                  P08514

  UniProtTarget
0        P05106
1        P05106
2        P05106
```

[1257]:
```python
P08648_df = pd.read_csv(os.path.join(prediction_dir, 'P08648_pred.csv'))
P08648_df['UniProtTarget'] = 'P08648'
P08648_df.head(3)
```

[1257]:
```
                                              SMILES PredictedUniProtPartner  \
0  Cc1cc(C)c(c(C)c1)S(=O)(=O)N[C@@H](CNC(=O)C2=NO…                  P05556
1  OC(=O)[C@H](Cc1cccc(OCCNc2ccccn2)c1)NC(=O)c3c(…                  P05556
2  CCCN(C(=O)CC(=O)O)C1=C(C)C[C@H](N([C@@H](C)c2c…                  P05556

  UniProtTarget
0        P08648
1        P08648
2        P08648
```

[1254]:
```python
P17301_df = pd.read_csv(os.path.join(prediction_dir, 'P17301_pred.csv'))
P17301_df['UniProtTarget'] = 'P17301'
P17301_df.head(3)
```

[1254]:
```
                                              SMILES PredictedUniProtPartner  \
0    Cc1ccccc1S(=O)(=O)N[C@@H](CNC(=O)c2cocc2)C(=O)O                  P05556
1  CCc1cc(O)c2c(O)c3C(=O)c4c(O)cccc4C(=O)c3cc2c1C…                  P05556

  UniProtTarget
0        P17301
```

```
1        P17301
```

1.5.1 Combine all Data Frames into One Whole Data Frame

[1258]:
```
df_list = [P13612_df, P05556_df, P05107_df, P05106_df, P08648_df, P17301_df]
```

[1259]:
```
combined_df = pd.concat(df_list, ignore_index=True)
```

[1260]:
```
combined_df
```

[1260]:
```
                                                  SMILES  \
0     OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…
1     CC1CCC(C[C@H](NC(=O)[C@@H]2CCC(=O)N2Cc3ccccc3)…
2     CC(C)CCNC(=O)[C@@H]1OCO[C@H]1C(=O)N[C@@H](Cc2c…
3     OC(=O)CN(CC(=O)N[C@@H](Cc1ccc(OCc2c(Cl)cccc2Cl…
4     CCC\N=C/1\C(\C(=C1O)O)=N\[C@@H](Cc2ccc(OCc3c(C…
…                                                  …
4187  Cc1cc(C)c(C(=O)N[C@@H](CNC(=O)CO[C@@H]2C[C@@H]…
4188  Cc1cc(C)c(c(C)c1)S(=O)(=O)N[C@@H](CNC(=O)CO[C@…
4189  OC(=O)[C@H](CNC(=O)CO[C@@H]1C[C@@H](CNc2ccccn2…
4190    Cc1ccccc1S(=O)(=O)N[C@@H](CNC(=O)c2cocc2)C(=O)O
4191  CCc1cc(O)c2c(O)c3C(=O)c4c(O)cccc4C(=O)c3cc2c1C…

     PredictedUniProtPartner UniProtTarget
0                     P05556        P13612
1                     P05556        P13612
2                     P05556        P13612
3                     P05556        P13612
4                     P05556        P13612
…                        …             …
4187                  P05556        P08648
4188                  P05556        P08648
4189                  P05556        P08648
4190                  P05556        P17301
4191                  P05556        P17301

[4192 rows x 3 columns]
```

[1261]:
```
new_order = ['SMILES', 'UniProtTarget', 'PredictedUniProtPartner']
```

[1262]:
```
combined_df = combined_df[new_order]
```

[1266]:
```
combined_df
```

[1266]:
```
                                                  SMILES UniProtTarget  \
0     OC(=O)[C@H](Cc1ccc(NC(=O)c2c(Cl)cccc2Cl)cc1)NC…        P13612
1     CC1CCC(C[C@H](NC(=O)[C@@H]2CCC(=O)N2Cc3ccccc3)…        P13612
2     CC(C)CCNC(=O)[C@@H]1OCO[C@H]1C(=O)N[C@@H](Cc2c…        P13612
3     OC(=O)CN(CC(=O)N[C@@H](Cc1ccc(OCc2c(Cl)cccc2Cl…        P13612
4     CCC\N=C/1\C(\C(=C1O)O)=N\[C@@H](Cc2ccc(OCc3c(C…        P13612
…                                                   …            …
4187  Cc1cc(C)c(C(=O)N[C@@H](CNC(=O)CO[C@@H]2C[C@@H]…        P08648
4188  Cc1cc(C)c(c(C)c1)S(=O)(=O)N[C@@H](CNC(=O)CO[C@…        P08648
4189  OC(=O)[C@H](CNC(=O)CO[C@@H]1C[C@@H](CNc2ccccn2…        P08648
4190    Cc1cccccc1S(=O)(=O)N[C@@H](CNC(=O)c2cocc2)C(=O)O        P17301
4191  CCc1cc(O)c2c(O)c3C(=O)c4c(O)cccc4C(=O)c3cc2c1C…        P17301

     PredictedUniProtPartner
0                     P05556
1                     P05556
2                     P05556
3                     P05556
4                     P05556
…                       …
4187                  P05556
4188                  P05556
4189                  P05556
4190                  P05556
4191                  P05556

[4192 rows x 3 columns]
```

[1267]:
```python
combined_df.to_csv('prediction_df.csv', index=False)

PRINT('SAVED & DONE !')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
SAVED & DONE !
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.5.2 Verify Data Frame Shape

[1274]:
```python
old_df = pd.read_csv(os.path.join('data','dataset_for_prediction.csv'))
```

[1275]:
```python
PRINT(f'Shapes check:\n\n{old_df.shape}\n\nvs.\n\n{combined_df.shape}')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Shapes check:

(4192, 2)

vs.

(4192, 3)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Everything seems fine with the shapes; the additional column is a result of appending the predicted

UniProt partner column to our combined dataframe.

```
[1279]: PRINT(f'-------------------------------------------------------------')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-------------------------------------------------------------
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
[ ]:
```