Drug Discovery

October 14, 2021

0.1 ChEMBL Database

The ChEMBL Database is a database that contains curated bioactivity data of more than 2 million compounds. It is compiled from more than 76,000 documents, 1.2 million assays and the data spans 13,000 targets and 1,800 cells and 33,000 indications. [Data as of March 25, 2020; ChEMBL version 26].

[1]:
```
import pandas as pd
import numpy as np
from chembl_webresource_client.new_client import new_client

import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
target = new_client.target
target_query = target.search('coronavirus')
targets = pd.DataFrame(target_query)
targets
```

[2]:
```
                                    cross_references  \
0                                                 []
1                                                 []
2                                                 []
3                                                 []
4  [{'xref_id': 'P0C6U8', 'xref_name': None, 'xre…
5                                                 []
6  [{'xref_id': 'P0C6X7', 'xref_name': None, 'xre…
7                                                 []

                                            organism  \
0                                        Coronavirus
1                                    SARS coronavirus
2                                  Feline coronavirus
3                              Human coronavirus 229E
4                                    SARS coronavirus
5  Middle East respiratory syndrome-related coron…
6                                    SARS coronavirus
```

```
7   Severe acute respiratory syndrome coronavirus 2

                                          pref_name  score  \
0                                       Coronavirus   17.0
1                                  SARS coronavirus   15.0
2                                Feline coronavirus   15.0
3                             Human coronavirus 229E   13.0
4                 SARS coronavirus 3C-like proteinase   10.0
5  Middle East respiratory syndrome-related coron…    9.0
6                          Replicase polyprotein 1ab    4.0
7                          Replicase polyprotein 1ab    4.0

   species_group_flag target_chembl_id  \
0               False     CHEMBL613732
1               False     CHEMBL612575
2               False     CHEMBL612744
3               False     CHEMBL613837
4               False       CHEMBL3927
5               False    CHEMBL4296578
6               False       CHEMBL5118
7               False    CHEMBL4523582

                                target_components     target_type   tax_id
0                                                []        ORGANISM    11119
1                                                []        ORGANISM   227859
2                                                []        ORGANISM    12663
3                                                []        ORGANISM    11137
4  [{'accession': 'P0C6U8', 'component_descriptio…  SINGLE PROTEIN   227859
5                                                []        ORGANISM  1335626
6  [{'accession': 'P0C6X7', 'component_descriptio…  SINGLE PROTEIN   227859
7  [{'accession': 'P0DTD1', 'component_descriptio…  SINGLE PROTEIN  2697049
```

From the table above we see the last entry is the protein that corresponds to Covid-19, the virus SARS-Covid2 also known as Severe acute respiratory syndrome coronavirus 2. To get bioactivity data we make a request to the chembl object with the target_chembl_id "CHEMBL4523582"

Here, we will retrieve only bioactivity data for coronavirus 3C-like proteinase (CHEMBL3927) that are reported as IC_{50} values in nM (nanomolar) unit.

[3]:
```
target_chembl_id = 'CHEMBL4523582'
activity = new_client.activity
res = activity.filter(target_chembl_id=target_chembl_id).
 ↪filter(standard_type="IC50")
```

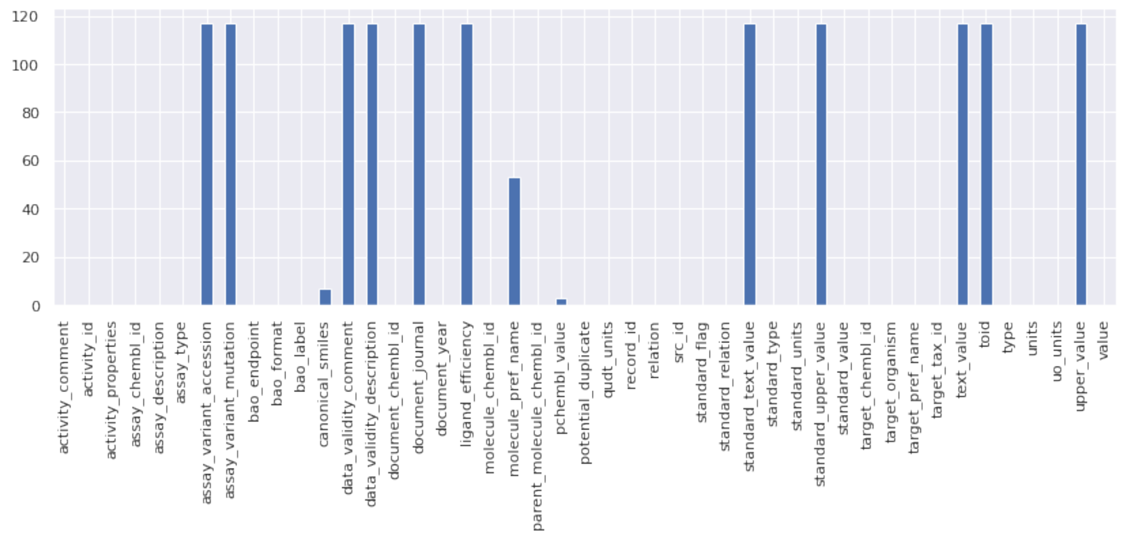
[19]:
```
df = pd.DataFrame(res)
df.head()
```

```
[19]:   activity_comment  activity_id activity_properties assay_chembl_id  \
      0  Dtt Insensitive     19964199                  []   CHEMBL4495583
      1  Dtt Insensitive     19964200                  []   CHEMBL4495583
      2  Dtt Insensitive     19964201                  []   CHEMBL4495583
      3  Dtt Insensitive     19964202                  []   CHEMBL4495583
      4  Dtt Insensitive     19964203                  []   CHEMBL4495583

                                       assay_description assay_type  \
      0  SARS-CoV-2 3CL-Pro protease inhibition IC50 de…          F
      1  SARS-CoV-2 3CL-Pro protease inhibition IC50 de…          F
      2  SARS-CoV-2 3CL-Pro protease inhibition IC50 de…          F
      3  SARS-CoV-2 3CL-Pro protease inhibition IC50 de…          F
      4  SARS-CoV-2 3CL-Pro protease inhibition IC50 de…          F

        assay_variant_accession assay_variant_mutation bao_endpoint   bao_format  \
      0                    None                   None  BAO_0000190  BAO_0000019
      1                    None                   None  BAO_0000190  BAO_0000019
      2                    None                   None  BAO_0000190  BAO_0000019
      3                    None                   None  BAO_0000190  BAO_0000019
      4                    None                   None  BAO_0000190  BAO_0000019

         …                              target_organism  \
      0  …  Severe acute respiratory syndrome coronavirus 2
      1  …  Severe acute respiratory syndrome coronavirus 2
      2  …  Severe acute respiratory syndrome coronavirus 2
      3  …  Severe acute respiratory syndrome coronavirus 2
      4  …  Severe acute respiratory syndrome coronavirus 2

                  target_pref_name target_tax_id text_value  toid  type  units  \
      0  Replicase polyprotein 1ab       2697049       None  None  IC50     uM
      1  Replicase polyprotein 1ab       2697049       None  None  IC50     uM
      2  Replicase polyprotein 1ab       2697049       None  None  IC50     uM
      3  Replicase polyprotein 1ab       2697049       None  None  IC50     uM
      4  Replicase polyprotein 1ab       2697049       None  None  IC50     uM

           uo_units upper_value value
      0  UO_0000065        None  0.39
      1  UO_0000065        None  0.21
      2  UO_0000065        None  0.08
      3  UO_0000065        None  1.58
      4  UO_0000065        None  0.04

      [5 rows x 45 columns]
```

Some columns are completely missing so they will be dropped while others are missing a few.

```
[20]: df.isna().sum().plot(kind='bar', figsize=(14, 4))
```

[20]: <AxesSubplot:>



There are a lot of columns here but we will only look at a few. - SMILES - simplified molecular input line entry system, = double bond, () attached to an atom, C - Carbon O - Oxygen N - Nitrogen - molecule_chembl_id - ID on chembl - standard_value - The target value is the standard value which is the potency of the drug.

The lower the standard value the better the potency of the drug. A drugs affectness is the IC50, the inhibitory concentration at 50%. Idealy we want small standard values so we get to 50% with a lower concertration.

For example, if you had to choose to take 5mL of medication or 5L of medication to have the same affect which would you choose. Also economically smaller doeses are easier to distribute and make

Now we will feature engineer the bioactivity class based on the standard value. The compounds will be

- active: Standard values having values of less than 1000 nM
- inactive: Standard values greater than 10,000 nM
- intermediate: Standard values between 1,000 and 10,000 nM will be referred to as intermediate.

[21]:
```
df['standard_value'] = pd.to_numeric(df['standard_value'])
df['bioactivity_class'] = 'intermediate'
df.loc[df['standard_value'].astype(float) >= 10000, 'bioactivity_class'] =␣
 ↪'inactive'
df.loc[df['standard_value'].astype(float) <= 1000, 'bioactivity_class'] =␣
 ↪'active'
```

Now we will query for the columns we need and drop rows that have missing canonical_smiles for the next step

```
[22]: bioactivity_columns=['molecule_chembl_id', 'canonical_smiles',␣
      ↪'standard_value', 'bioactivity_class']
      df = df[bioactivity_columns].dropna(subset=['canonical_smiles'])
```

```
[23]: df.isna().any()
```

```
[23]: molecule_chembl_id    False
      canonical_smiles      False
      standard_value        False
      bioactivity_class     False
      dtype: bool
```

```
[24]: df.head()
```

```
[24]:   molecule_chembl_id                                    canonical_smiles  \
      0          CHEMBL480   Cc1c(OCC(F)(F)F)ccnc1C[S+]([O-])c1nc2ccccc2[nH]1
      1       CHEMBL178459                          Cc1c(-c2cnccn2)ssc1=S
      2      CHEMBL3545157             O=c1sn(-c2cccc3ccccc23)c(=O)n1Cc1ccccc1
      3       CHEMBL297453  O=C(O[C@@H]1Cc2c(O)cc(O)cc2O[C@@H]1c1cc(O)c(O)…
      4      CHEMBL4303595                             O=C1C=Cc2cc(Br)ccc2C1=O

         standard_value bioactivity_class
      0           390.0            active
      1           210.0            active
      2            80.0            active
      3          1580.0      intermediate
      4            40.0            active
```

0.1.1 Calculate Lipinski descriptors

With the chembl data cleaned we can now calculate Lipinski descriptors based on the canonical_smiles.

Christopher Lipinski, a scientist at Pfizer, came up with a set of rule-of-thumb for evaluating the druglikeness of compounds. Such druglikeness is based on the Absorption, Distribution, Metabolism and Excretion (ADME) that is also known as the pharmacokinetic profile. Lipinski analyzed all orally active FDA-approved drugs in the formulation of what is to be known as the Rule-of-Five or Lipinski's Rule. The rule of five corresponds to how the four descriptors are all within a range that is a multiple of 5.

The Lipinski's Rule stated the following:

- Molecular weight < 500 Dalton
- Octanol-water partition coefficient (LogP) < 5
- Hydrogen bond donors < 5
- Hydrogen bond acceptors < 10

```
[25]: from rdkit import Chem
      from rdkit.Chem import Descriptors, Lipinski
```

Using rdkit we can get the 4 descriptor, moldata will be a column of objects that we can then use to compute the decriptors

[26]:
```
df['moldata'] = df['canonical_smiles'].map(Chem.MolFromSmiles)
df.head()
```

[26]:
```
  molecule_chembl_id                                  canonical_smiles  \
0          CHEMBL480   Cc1c(OCC(F)(F)F)ccnc1C[S+]([O-])c1nc2ccccc2[nH]1
1       CHEMBL178459                           Cc1c(-c2cnccn2)ssc1=S
2      CHEMBL3545157            O=c1sn(-c2cccc3ccccc23)c(=O)n1Cc1ccccc1
3       CHEMBL297453  O=C(O[C@@H]1Cc2c(O)cc(O)cc2O[C@@H]1c1cc(O)c(O)…
4      CHEMBL4303595                           O=C1C=Cc2cc(Br)ccc2C1=O

   standard_value bioactivity_class  \
0           390.0            active
1           210.0            active
2            80.0            active
3          1580.0      intermediate
4            40.0            active

                                        moldata
0  <rdkit.Chem.rdchem.Mol object at 0x7f20c70a7f80>
1  <rdkit.Chem.rdchem.Mol object at 0x7f20c70a7990>
2  <rdkit.Chem.rdchem.Mol object at 0x7f20c70a70d0>
3  <rdkit.Chem.rdchem.Mol object at 0x7f20c70a7df0>
4  <rdkit.Chem.rdchem.Mol object at 0x7f20c70a7620>
```

[27]:
```
df['molwt'] = df['moldata'].map(Descriptors.MolWt)
df['logP'] = df['moldata'].map(Descriptors.MolLogP)
df['numHDonors'] = df['moldata'].map(Descriptors.NumHDonors)
df['numHAcceptors'] = df['moldata'].map(Descriptors.NumHAcceptors)
```

Now that we have the descriptors we no longer need the moldata objects

[28]:
```
lipinski_descriptors = ['molwt', 'logP', 'numHDonors', 'numHAcceptors']
df = df[lipinski_descriptors + ['bioactivity_class', 'standard_value',␣
 ↪'canonical_smiles']]
df.head()
```

[28]:
```
     molwt     logP  numHDonors  numHAcceptors bioactivity_class  \
0  369.368  3.51522           1              4            active
1  226.351  3.30451           0              5            active
2  334.400  3.26220           0              5            active
3  458.375  2.23320           8             11      intermediate
4  237.052  2.22770           0              2            active

   standard_value                                  canonical_smiles
0           390.0  Cc1c(OCC(F)(F)F)ccnc1C[S+]([O-])c1nc2ccccc2[nH]1
```

```
1          210.0                             Cc1c(-c2cnccn2)ssc1=S
2           80.0              O=c1sn(-c2cccc3ccccc23)c(=O)n1Cc1ccccc1
3         1580.0  O=C(O[C@@H]1Cc2c(O)cc(O)cc2O[C@@H]1c1cc(O)c(O)…
4           40.0                           O=C1C=Cc2cc(Br)ccc2C1=O
```

0.1.2 Calculate the molecular finger print

Another set of features we can collect is the finger print of the canonical_smiles using padelpy

[29]:
```
from padelpy import from_smiles
```

[30]:
```
pdf = df['canonical_smiles'].map(lambda x: from_smiles(x, fingerprints=True,␣
 ↪descriptors=False))
pdf.head()
```

[30]:
```
0    {'PubchemFP0': '1', 'PubchemFP1': '1', 'Pubche…
1    {'PubchemFP0': '1', 'PubchemFP1': '0', 'Pubche…
2    {'PubchemFP0': '1', 'PubchemFP1': '1', 'Pubche…
3    {'PubchemFP0': '1', 'PubchemFP1': '1', 'Pubche…
4    {'PubchemFP0': '1', 'PubchemFP1': '0', 'Pubche…
Name: canonical_smiles, dtype: object
```

The padel data frame is a series of list of dict, we convert the data into dataframe

[31]:
```
padel_df = pd.DataFrame(pdf.tolist()).apply(pd.to_numeric)
padel_df.head()
```

[31]:
```
   PubchemFP0  PubchemFP1  PubchemFP2  PubchemFP3  PubchemFP4  PubchemFP5  \
0           1           1           0           0           0           0
1           1           0           0           0           0           0
2           1           1           0           0           0           0
3           1           1           1           0           0           0
4           1           0           0           0           0           0

   PubchemFP6  PubchemFP7  PubchemFP8  PubchemFP9  …  PubchemFP871  \
0           0           0           0           1  …             0
1           0           0           0           1  …             0
2           0           0           0           1  …             0
3           0           0           0           1  …             0
4           0           0           0           1  …             0

   PubchemFP872  PubchemFP873  PubchemFP874  PubchemFP875  PubchemFP876  \
0             0             0             0             0             0
1             0             0             0             0             0
2             0             0             0             0             0
3             0             0             0             0             0
4             0             0             0             0             0
```

```
   PubchemFP877  PubchemFP878  PubchemFP879  PubchemFP880
0             0             0             0             0
1             0             0             0             0
2             0             0             0             0
3             0             0             0             0
4             0             0             0             0

[5 rows x 881 columns]
```

There are 881 columns are the data is very sparse, we can drop columns that have very little variance

[32]:
```
zero_cols = padel_df.sum().loc[lambda x: x<10]
ones_cols = padel_df.sum().loc[lambda x: x>100]
print(f'Columns that are mostly 0s: {zero_cols.shape[0]}')
print(f'Columns that are mostly 1s: {ones_cols.shape[0]}')
```

```
Columns that are mostly 0s: 580
Columns that are mostly 1s: 9
```

[33]:
```
padel_df = padel_df.drop(zero_cols.index.tolist() + ones_cols.index.tolist(),␣
 ↪axis=1)
padel_df.head()
```

[33]:
```
   PubchemFP1  PubchemFP2  PubchemFP12  PubchemFP14  PubchemFP15  PubchemFP16  \
0           1           0            1            1            1            0
1           0           0            0            1            1            0
2           1           0            1            1            1            0
3           1           1            1            0            0            0
4           0           0            0            0            0            0

   PubchemFP18  PubchemFP19  PubchemFP20  PubchemFP21  …  PubchemFP776  \
0            1            1            0            0  …             0
1            0            0            0            0  …             0
2            1            1            0            0  …             0
3            1            1            1            1  …             0
4            1            1            0            0  …             0

   PubchemFP777  PubchemFP797  PubchemFP798  PubchemFP800  PubchemFP803  \
0             0             0             0             0             0
1             0             0             0             0             0
2             0             0             0             1             0
3             1             0             1             0             1
4             0             0             0             0             0

   PubchemFP818  PubchemFP819  PubchemFP821  PubchemFP824
0             0             0             0             0
1             0             0             0             0
```

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

[5 rows x 292 columns]
```

We can now merge the padel_df and the bioactivity dataframe and drop column

[34]:
```
df = df.merge(padel_df, left_index=True, right_index=True)
df.head()
```

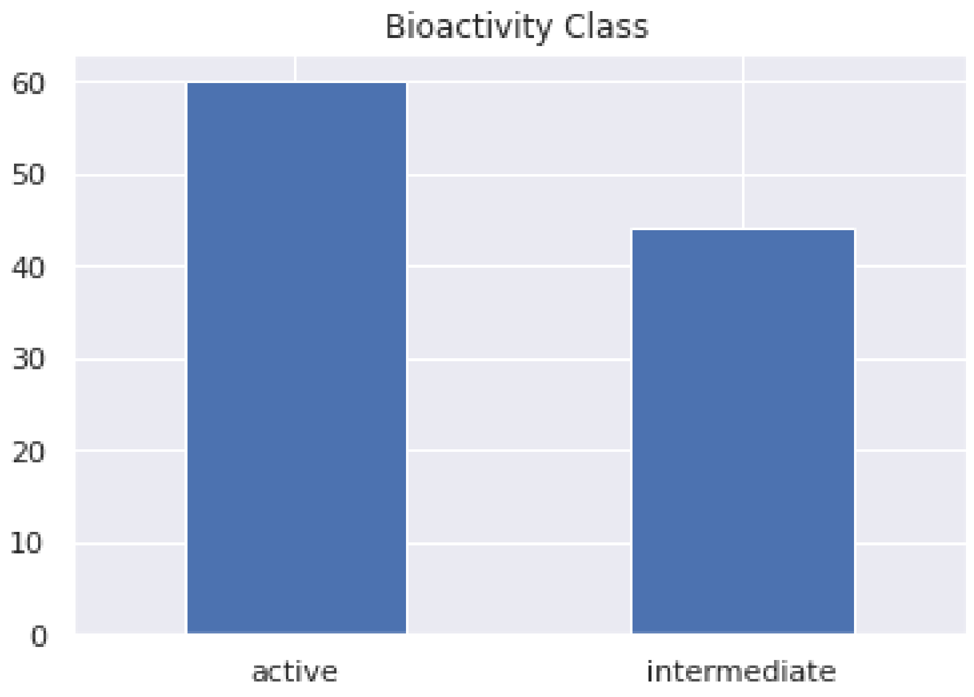
[34]:
```
     molwt     logP  numHDonors  numHAcceptors bioactivity_class  \
0  369.368  3.51522           1              4            active
1  226.351  3.30451           0              5            active
2  334.400  3.26220           0              5            active
3  458.375  2.23320           8             11      intermediate
4  237.052  2.22770           0              2            active

   standard_value                                  canonical_smiles  \
0           390.0   Cc1c(OCC(F)(F)F)ccnc1C[S+]([O-])c1nc2ccccc2[nH]1
1           210.0                               Cc1c(-c2cnccn2)ssc1=S
2            80.0             O=c1sn(-c2cccc3ccccc23)c(=O)n1Cc1ccccc1
3          1580.0  O=C(O[C@@H]1Cc2c(O)cc(O)cc2O[C@@H]1c1cc(O)c(O)…
4            40.0                             O=C1C=Cc2cc(Br)ccc2C1=O

   PubchemFP1  PubchemFP2  PubchemFP12  …  PubchemFP776  PubchemFP777  \
0           1           0            1  …             0             0
1           0           0            0  …             0             0
2           1           0            1  …             0             0
3           1           1            1  …             0             1
4           0           0            0  …             0             0

   PubchemFP797  PubchemFP798  PubchemFP800  PubchemFP803  PubchemFP818  \
0             0             0             0             0             0
1             0             0             0             0             0
2             0             0             1             0             1
3             0             1             0             1             0
4             0             0             0             0             1

   PubchemFP819  PubchemFP821  PubchemFP824
0             0             0             0
1             0             0             0
2             0             1             0
3             1             0             1
4             0             0             0

[5 rows x 299 columns]
```

0.2 Exploratory Data Analysis

We begin with the distribution of the Bioactivity Class

```
[171]: df['bioactivity_class'].value_counts().plot(kind='bar', title='Bioactivity␣
        ↪Class', rot=0)
```

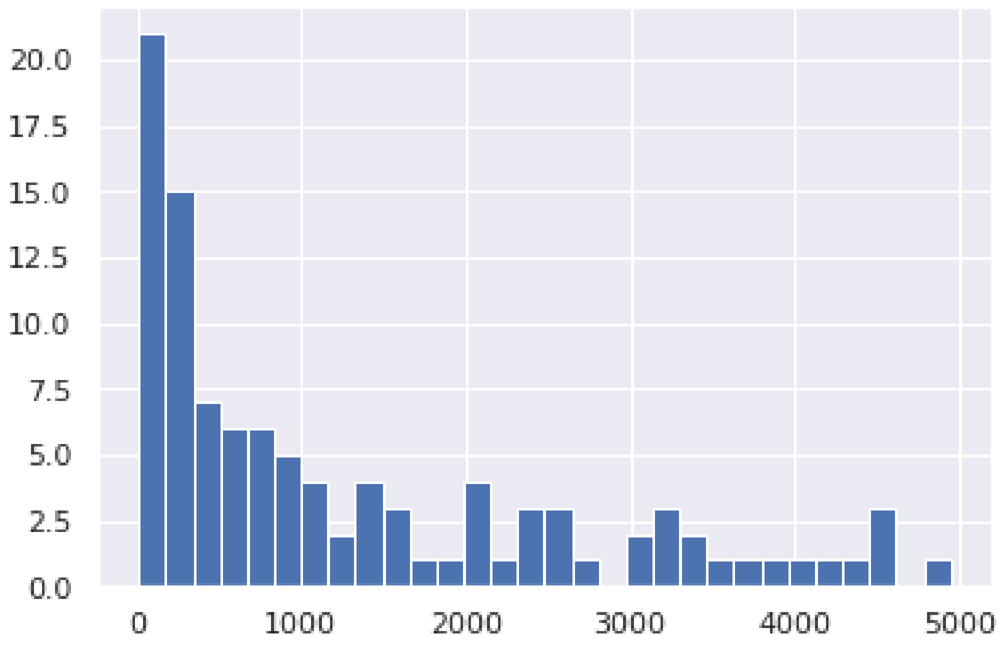
```
[171]: <AxesSubplot:title={'center':'Bioactivity Class'}>
```



There is a good balance between active and intermediate compounds

```
[172]: df['standard_value'].hist(bins=30)
```

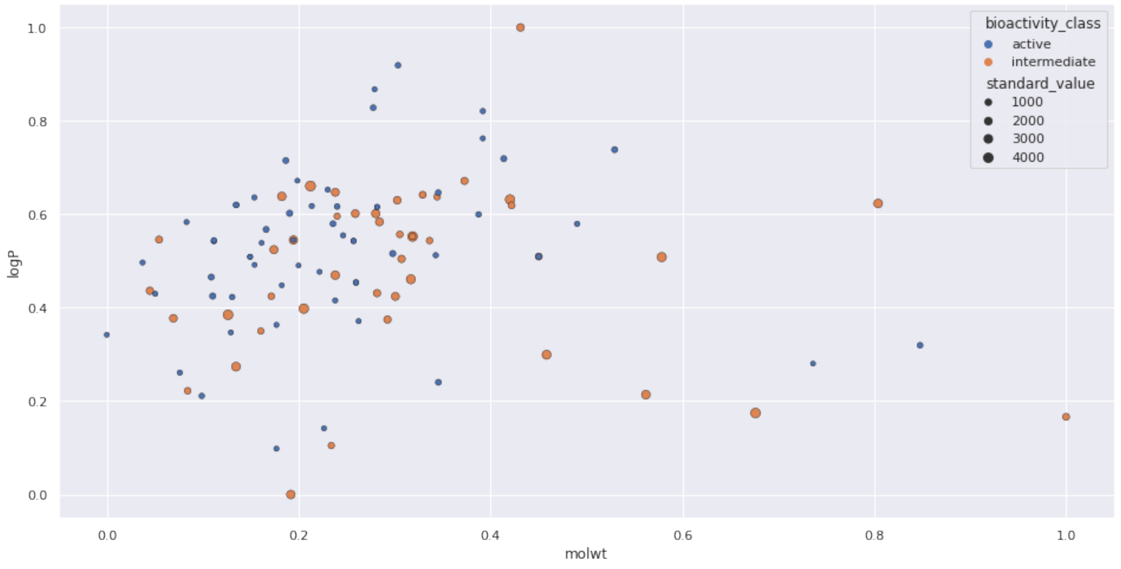
```
[172]: <AxesSubplot:>
```



The distribution of the standard value is right scewed

```
[173]: plt.figure(figsize=(16,8))
       sns.scatterplot(x='molwt', y='logP', data=df, hue='bioactivity_class',␣
        ↪size='standard_value', edgecolor='black')
```

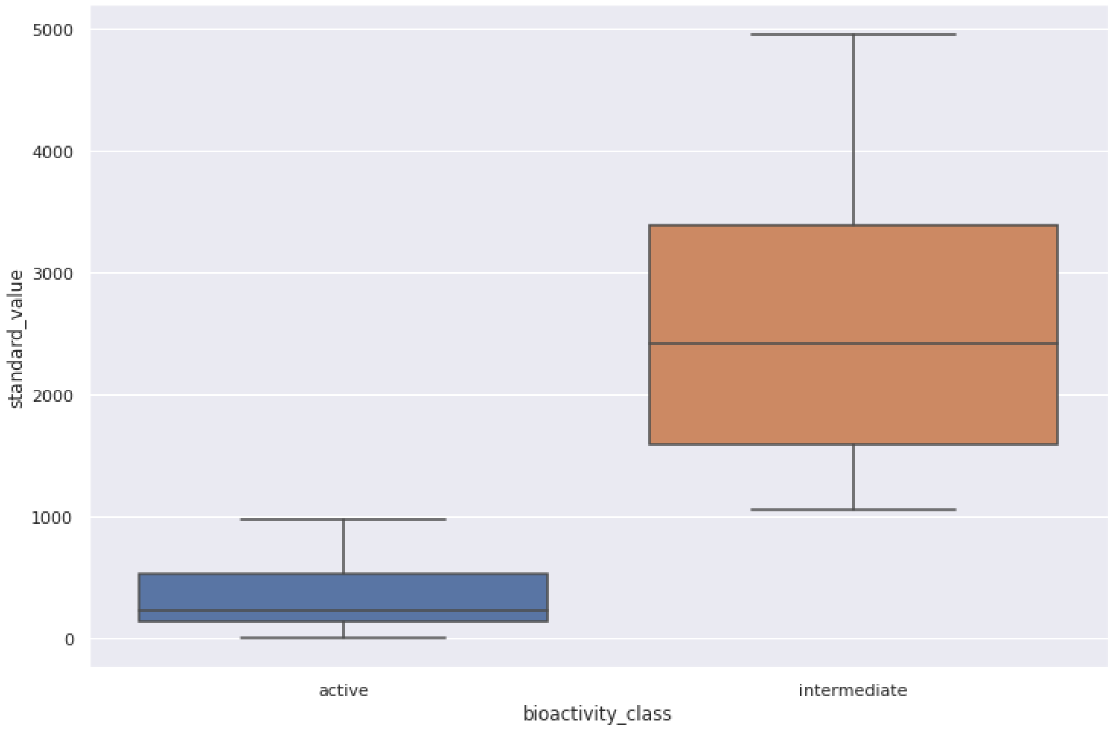
```
[173]: <AxesSubplot:xlabel='molwt', ylabel='logP'>
```



Smaller molwt seem to be linearly coorelated with logP for both active and intermediate compounds, but as values get larger they thin out

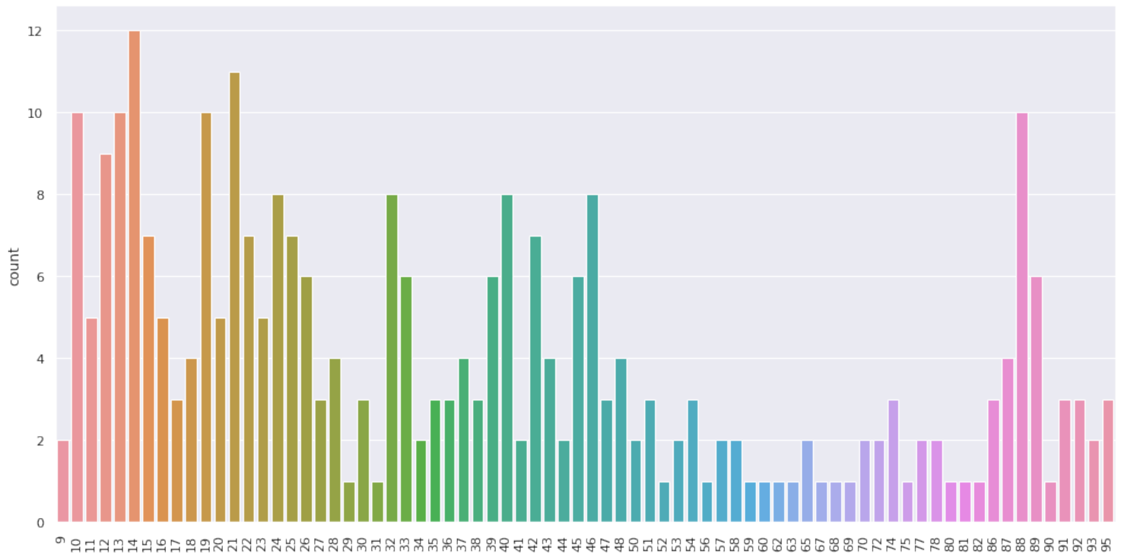
```
[174]: plt.figure(figsize=(12,8))
       sns.boxplot(x='bioactivity_class', y='standard_value', data=df)
```

```
[174]: <AxesSubplot:xlabel='bioactivity_class', ylabel='standard_value'>
```



Active compounds have a small spread while intermediate have a much IQR

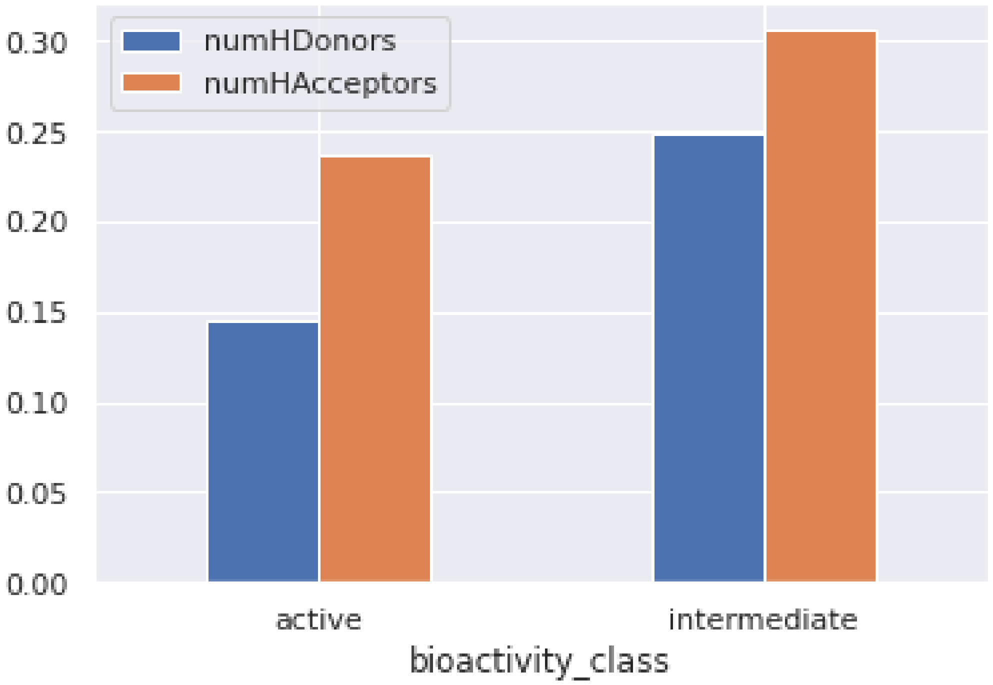
```
[175]: pub_vals = df.filter(like='Pub').sum().sort_values().values
       plt.figure(figsize=(16,8))
       splot = sns.countplot(x=pub_vals)
       splot.tick_params(axis='x', rotation=90)
```



The Pub values don't seem to have a specific distribution

```
[176]: df.groupby('bioactivity_class').agg({'numHDonors': 'mean', 'numHAcceptors':␣
         ↪'mean'}).plot(kind='bar', rot=0)
```

```
[176]: <AxesSubplot:xlabel='bioactivity_class'>
```



Active compounds have less numHDonors and numHAcceptors compared to intermediate

Next let's look at which columns are statistically significant with respect to bioactivity_class. One way to test is the mann whitneyu test, conducted below.

The Mann-Whitney U test is a nonparametric test of the null hypothesis that the distribution underlying sample x is the same as the distribution underlying sample y. It is often used as a test of of difference in location between distributions.

[35]:
```
from scipy.stats import mannwhitneyu
```

[36]:
```
def statistical_significance(x, df, alpha=1e-2):

    adf = df[df['bioactivity_class'].eq('active')][x]
    idf = df[df['bioactivity_class'].eq('intermediate')][x]
    stat, pval = mannwhitneyu(adf, idf)

    h = 'fail to reject H0' if pval > alpha else 'reject H0'
    results = {'Descriptor':x,
               'Statistics':stat,
               'p':pval,
               'alpha':alpha,
               'Interpretation':h}

    return results
```

[37]:
```
pd.concat([pd.DataFrame(statistical_significance(i, df), index=[0]) for i in␣
 ↪lipinski_descriptors + ['standard_value']]).sort_values('Interpretation')
```

[37]:
```
       Descriptor  Statistics             p  alpha     Interpretation
0           molwt       976.5  1.200940e-02   0.01  fail to reject H0
0            logP      1170.5  1.634514e-01   0.01  fail to reject H0
0   numHAcceptors      1076.5  5.325991e-02   0.01  fail to reject H0
0      numHDonors       883.5  1.664367e-03   0.01          reject H0
0  standard_value         0.0  1.936102e-18   0.01          reject H0
```

Taking a look at standard values, the actives and inactives displayed statistically significant difference, which is to be expected since we define the bioactivity threshold with it.

Of the 4 Lipinski's descriptors (MW, LogP, NumHDonors and NumHAcceptors), only numHDonors shows statistically significant difference between actives and inactives.

0.3 Modeling

We will train 2 random forests, one with all the features and another with only significant Lipinski's descriptors

[38]:
```
import numpy as np
np.random.seed(42)
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
```

[39]:
```
df.head()
```

[39]:
```
    molwt     logP  numHDonors  numHAcceptors bioactivity_class  \
0  369.368  3.51522           1              4            active
1  226.351  3.30451           0              5            active
2  334.400  3.26220           0              5            active
3  458.375  2.23320           8             11      intermediate
4  237.052  2.22770           0              2            active

   standard_value                                   canonical_smiles  \
0           390.0   Cc1c(OCC(F)(F)F)ccnc1C[S+]([O-])c1nc2ccccc2[nH]1
1           210.0                              Cc1c(-c2cnccn2)ssc1=S
2            80.0              O=c1sn(-c2cccc3ccccc23)c(=O)n1Cc1ccccc1
3          1580.0  O=C(O[C@@H]1Cc2c(O)cc(O)cc2O[C@@H]1c1cc(O)c(O)…
4            40.0                                O=C1C=Cc2cc(Br)ccc2C1=O

   PubchemFP1  PubchemFP2  PubchemFP12  …  PubchemFP776  PubchemFP777  \
0           1           0            1  …             0             0
1           0           0            0  …             0             0
2           1           0            1  …             0             0
3           1           1            1  …             0             1
4           0           0            0  …             0             0

   PubchemFP797  PubchemFP798  PubchemFP800  PubchemFP803  PubchemFP818  \
0             0             0             0             0             0
1             0             0             0             0             0
2             0             0             1             0             1
3             0             1             0             1             0
4             0             0             0             0             1

   PubchemFP819  PubchemFP821  PubchemFP824
0             0             0             0
1             0             0             0
2             0             1             0
3             1             0             1
4             0             0             0

[5 rows x 299 columns]
```

[165]:
```
def train_model(X, y, trees):
    np.random.seed(42)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    model = RandomForestClassifier(n_estimators=trees)
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    return score, X_test, y_test, model
```

[166]:
```
non_features = ['standard_value','bioactivity_class', 'canonical_smiles']
```

[167]:
```
X, y = df.drop(non_features, axis=1), df['bioactivity_class']
print(f'Training X shape {X.shape}')
print(f'Training y shape {y.shape}')
```

```
Training X shape (104, 296)
Training y shape (104,)
```

[168]:
```
score, X_test, y_test, model = train_model(X, y, 1000)
print(f'Training score: {score}')
```

```
Training score: 0.5238095238095238
```

[169]:
```
y_pred = model.predict(X_test)
```

[170]:
```
eval_df = pd.DataFrame({'y_true': y_test, 'y_pred': y_pred})
print(classification_report(y_true=eval_df['y_true'], y_pred=eval_df['y_pred']))
```

```
              precision    recall  f1-score   support

      active       0.67      0.67      0.67        15
intermediate       0.17      0.17      0.17         6

    accuracy                           0.52        21
   macro avg       0.42      0.42      0.42        21
weighted avg       0.52      0.52      0.52        21
```

[51]:
```
insignificant_cols = ['numHAcceptors', 'logP', 'molwt']
X, y = df.drop(non_features + insignificant_cols, axis=1),␣
 ↪df['bioactivity_class']
print(f'Training X shape {X.shape}')
print(f'Training y shape {y.shape}')
```

```
Training X shape (104, 293)
Training y shape (104,)
```

[52]:
```
score, X_test, y_test, model = train_model(X, y, 500)
print(f'Training score: {score}')
```

```
Training score: 0.5238095238095238
```

[53]:
```
y_pred = model.predict(X_test)
```

[54]:
```
eval_df = pd.DataFrame({'y_true': y_test, 'y_pred': y_pred})
print(classification_report(y_true=eval_df['y_true'], y_pred=eval_df['y_pred']))
```

```
              precision    recall  f1-score   support

      active       0.69      0.60      0.64        15
intermediate       0.25      0.33      0.29         6

    accuracy                           0.52        21
   macro avg       0.47      0.47      0.46        21
weighted avg       0.57      0.52      0.54        21
```

[71]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
```

[112]:
```
df[lipinski_descriptors] = df[lipinski_descriptors].apply(lambda x: (x - np.
 ↪min(x)) / (np.max(x) - np.min(x)))
```

[134]:
```
X = torch.Tensor(df.drop(non_features, axis=1).values)
nfeatures = X.shape[1]
y = pd.get_dummies(df['bioactivity_class'])
y = torch.Tensor(y.values)

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.15)
```

[140]:
```
y_val.shape
```

[140]:
```
torch.Size([16, 2])
```

[144]:
```
def build_model(nfeatures, device='cpu'):
    dense = 32

    model = nn.Sequential(
        nn.Linear(nfeatures, dense),
        nn.ReLU(inplace=True),
        nn.Linear(dense, dense//2),
        nn.ReLU(inplace=True),
        nn.Linear(dense//2, 2),
        nn.Sigmoid()
        ).to(device)

    return model

model = build_model(nfeatures)
model
```

[144]:
```
Sequential(
  (0): Linear(in_features=296, out_features=32, bias=True)
```

```
    (1): ReLU(inplace=True)
    (2): Linear(in_features=32, out_features=16, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=16, out_features=2, bias=True)
    (5): Sigmoid()
  )
```

[162]:
```
def torch_model(model, criterion, optimizer, num_epochs=3, device='cpu'):

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch+1, num_epochs))
        print('-' * 10)

        running_loss = 0.0
        running_corrects = 0

        for inputs, labels in zip(X_train, y_train):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            loss = criterion(outputs, labels)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        for inputs, labels in zip(X_val, y_val):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            loss = criterion(outputs, labels)
            pred = torch.argmax(outputs)

            running_loss += loss.item() * inputs.size(0)
            running_corrects += (pred == torch.argmax(labels))

        epoch_loss = running_loss / len(X_val)
        epoch_acc = running_corrects.double() / len(X_val)

        print('Epoch: {} loss: {:.4f}, acc: {:.4f}'.format(epoch, epoch_loss,␣
 ↪epoch_acc))
    return model
```

[163]:
```
#criterion = nn.BCEWithLogitsLoss()
criterion = nn.BCELoss()
```

```
optimizer = optim.Adam(model.parameters())
model_trained = torch_model(model, criterion, optimizer, num_epochs=50)
```

```
Epoch 1/50
----------
Epoch: 0 loss: 635.2976, acc: 0.5625
Epoch 2/50
----------
Epoch: 1 loss: 663.2454, acc: 0.5625
Epoch 3/50
----------
Epoch: 2 loss: 647.4379, acc: 0.5625
Epoch 4/50
----------
Epoch: 3 loss: 652.7914, acc: 0.5625
Epoch 5/50
----------
Epoch: 4 loss: 658.7648, acc: 0.5625
Epoch 6/50
----------
Epoch: 5 loss: 681.4570, acc: 0.5625
Epoch 7/50
----------
Epoch: 6 loss: 1462.4322, acc: 0.5625
Epoch 8/50
----------
Epoch: 7 loss: 658.0520, acc: 0.5625
Epoch 9/50
----------
Epoch: 8 loss: 691.6519, acc: 0.5625
Epoch 10/50
----------
Epoch: 9 loss: 650.4499, acc: 0.5000
Epoch 11/50
----------
Epoch: 10 loss: 657.0429, acc: 0.5000
Epoch 12/50
----------
Epoch: 11 loss: 1453.1509, acc: 0.5000
Epoch 13/50
----------
Epoch: 12 loss: 1469.5395, acc: 0.5625
Epoch 14/50
----------
Epoch: 13 loss: 1480.2909, acc: 0.5625
Epoch 15/50
----------
Epoch: 14 loss: 1492.9968, acc: 0.5625
```

```
Epoch 16/50
----------
Epoch: 15 loss: 1498.7855, acc: 0.5000
Epoch 17/50
----------
Epoch: 16 loss: 1519.7517, acc: 0.5000
Epoch 18/50
----------
Epoch: 17 loss: 1527.2460, acc: 0.5000
Epoch 19/50
----------
Epoch: 18 loss: 1525.1408, acc: 0.5000
Epoch 20/50
----------
Epoch: 19 loss: 1532.9584, acc: 0.5625
Epoch 21/50
----------
Epoch: 20 loss: 1514.4528, acc: 0.5625
Epoch 22/50
----------
Epoch: 21 loss: 1522.6337, acc: 0.5625
Epoch 23/50
----------
Epoch: 22 loss: 1526.3275, acc: 0.5625
Epoch 24/50
----------
Epoch: 23 loss: 1521.1605, acc: 0.5000
Epoch 25/50
----------
Epoch: 24 loss: 1530.6089, acc: 0.5625
Epoch 26/50
----------
Epoch: 25 loss: 1545.9663, acc: 0.5625
Epoch 27/50
----------
Epoch: 26 loss: 1547.2604, acc: 0.5000
Epoch 28/50
----------
Epoch: 27 loss: 1545.6268, acc: 0.5000
Epoch 29/50
----------
Epoch: 28 loss: 1548.3075, acc: 0.5000
Epoch 30/50
----------
Epoch: 29 loss: 1536.9853, acc: 0.5000
Epoch 31/50
----------
Epoch: 30 loss: 1550.5007, acc: 0.5000
```

```
Epoch 32/50
----------
Epoch: 31 loss: 1563.2408, acc: 0.5000
Epoch 33/50
----------
Epoch: 32 loss: 1559.7170, acc: 0.5000
Epoch 34/50
----------
Epoch: 33 loss: 1532.8507, acc: 0.5625
Epoch 35/50
----------
Epoch: 34 loss: 1543.4288, acc: 0.5000
Epoch 36/50
----------
Epoch: 35 loss: 1559.2325, acc: 0.5000
Epoch 37/50
----------
Epoch: 36 loss: 1516.1327, acc: 0.5000
Epoch 38/50
----------
Epoch: 37 loss: 1541.6872, acc: 0.5625
Epoch 39/50
----------
Epoch: 38 loss: 1547.2140, acc: 0.5625
Epoch 40/50
----------
Epoch: 39 loss: 1563.9856, acc: 0.5625
Epoch 41/50
----------
Epoch: 40 loss: 1562.7542, acc: 0.5625
Epoch 42/50
----------
Epoch: 41 loss: 1564.1597, acc: 0.5625
Epoch 43/50
----------
Epoch: 42 loss: 1574.9132, acc: 0.5625
Epoch 44/50
----------
Epoch: 43 loss: 1578.1881, acc: 0.5625
Epoch 45/50
----------
Epoch: 44 loss: 1582.6948, acc: 0.5625
Epoch 46/50
----------
Epoch: 45 loss: 1589.7124, acc: 0.5625
Epoch 47/50
----------
Epoch: 46 loss: 1592.7140, acc: 0.5625
```

```
Epoch 48/50
----------
Epoch: 47 loss: 1602.0390, acc: 0.5625
Epoch 49/50
----------
Epoch: 48 loss: 1604.5203, acc: 0.5625
Epoch 50/50
----------
Epoch: 49 loss: 1609.6037, acc: 0.5625
```

```
[ ]:
```