Applied_Machine_learning

November 6, 2020

1 Leishmania Drug Discovery

1.0.1 Import libs

[1]:
```
from pytorch_tabnet.tab_model import TabNetRegressor
import torch
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
import numpy as np
import pandas as pd
import rdkit
from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit.Chem import Draw
from rdkit.Chem.Draw import IPythonConsole
from IPython.display import Image
from rdkit.Chem import Descriptors
import matplotlib.pyplot as plt
from rdkit.Chem import DataStructs
from sklearn.model_selection import train_test_split, StratifiedKFold,␣
↪GridSearchCV
import pandas as pd
import numpy as np
np.random.seed(1779177)
import os
#import wget
from pathlib import Path

from matplotlib import pyplot as plt
%matplotlib inline
```

1.0.2 Load Data

[2]:
```
drug_c = pd.read_csv("drugCentral.csv")
endogenous = pd.read_csv("endogenous.csv")
in_trails = pd.read_csv("in-trials.csv")
world = pd.read_csv("world.csv")
data = pd.read_csv("Bioactivities.csv",sep='\t')
```

```
data1 = pd.read_csv("Bioactivities1.csv", sep=';',low_memory=False)
```

[3]:
```
# df1 = pd.read_csv('Bioactivities2.csv',sep='\t',low_memory=False)
```

1.1 Data Cleaning

1.1.1 Remove duplicates

[4]:
```
df1 = data1.dropna(subset=['Smiles', 'Molecule ChEMBL ID']).
 ↪drop_duplicates(subset=['Molecule ChEMBL ID'])
```

1.1.2 Select Type Inhibition only

[5]:
```
df_activity1 = df1.loc[(df1['Standard Type'] == 'Inhibition') & (df1['Standard␣
 ↪Units'] == '%')].dropna(subset=['Standard Value'])
df_activity1['logValue'] = 1 * np.log(df_activity1['Standard Value'])
```

```
/home/moazmohamed/miniconda3/lib/python3.7/site-
packages/pandas/core/series.py:726: RuntimeWarning: divide by zero encountered
in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/home/moazmohamed/miniconda3/lib/python3.7/site-
packages/pandas/core/series.py:726: RuntimeWarning: invalid value encountered in
log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

1.1.3 Log transformation

[6]:
```
df1['logValue'] = df_activity1['logValue']
```

1.1.4 Remove Inf and -Inf from data

[7]:
```
df1 = df1.replace([np.inf, -np.inf], np.nan).dropna(subset=['logValue'],␣
 ↪how="all")
```

1.2 Feature Engineering

1.2.1 Get molecules objects from SMILES notaion

[8]:
```
df1['mol'] = df1['Smiles'].apply(Chem.MolFromSmiles)
```

1.2.2 Add descriptive Features for all the molecules

[9]:
```
df1 = df1.dropna(subset=['mol'])
df1['HeavyAtomCount'] = df1['mol'].apply(Descriptors.HeavyAtomCount)
df1['HAccept'] = df1['mol'].apply(Descriptors.NumHAcceptors)
```

```python
df1['HDonor'] = df1['mol'].apply(Descriptors.NumHDonors)
df1['Heteroatoms'] = df1['mol'].apply(Descriptors.NumHeteroatoms)
df1['RingCount'] = df1['mol'].apply(Descriptors.RingCount)
df1['SaturatedRings'] = df1['mol'].apply(Descriptors.NumSaturatedRings)
df1['AliphaticRings'] = df1['mol'].apply(Descriptors.NumAliphaticRings)
df1['AromaticRings'] = df1['mol'].apply(Descriptors.NumAromaticRings)
df1['Ipc'] = df1['mol'].apply(Descriptors.Ipc)
df1['HallKierAlpha'] = df1['mol'].apply(Descriptors.HallKierAlpha)
df1['NumValenceElectrons'] = df1['mol'].apply(Descriptors.NumValenceElectrons)
df1['MolLogP'] = df1['mol'].apply(Descriptors.MolLogP)
df1['AMW'] = df1['mol'].apply(Descriptors.MolWt)
df1['NumRotatableBonds'] = df1['mol'].apply(Descriptors.NumRotatableBonds)
```

[10]:
```python
X1 = df1[['MolLogP', 'AMW', 'NumRotatableBonds', 'HeavyAtomCount',
       'HAccept', 'HDonor', 'Heteroatoms', 'RingCount', 'SaturatedRings',
       'AliphaticRings', 'AromaticRings', 'Ipc', 'HallKierAlpha',
       'NumValenceElectrons']]
y1 = df1['logValue']
```

1.3 Prepare for Training

1.3.1 Train/valid/test Split and scaling

[11]:
```python
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
scaler_xtrain = RobustScaler()


X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=0.11,␣
↪random_state=1995)

# X_train, X_valid, y_train, y_valid = train_test_split(X1,y1, test_size=0.3,␣
↪random_state=1995)

X_train, X_valid, y_train, y_valid = train_test_split(X_train,y_train,␣
↪test_size=0.2, random_state=1995)


X_train = scaler_xtrain.fit_transform(X_train)
# y_train = scaler_xtrain.transform(y_train.to_numpy().reshape(-1,1))

X_valid = scaler_xtrain.transform(X_valid)
# y_valid = scaler_xtrain.transform(y_valid.to_numpy().reshape(-1,1))

X_test = scaler_xtrain.transform(X_test)
# y_test = scaler_xtrain.transform(y_test.to_numpy().reshape(-1,1))
```

1.4 TabNet: Attentive Interpretable Tabular Learning

1.4.1 Model

[12]:
```
batch_size = 16384
max_epochs = 1000
clf = TabNetRegressor(n_d = 32 , n_a = 32 , n_independent = 4, n_shared =4 ,␣
↪n_steps=4 ,lambda_sparse = 1e-3,
    optimizer_fn=torch.optim.Adam, # Any optimizer works here
                       optimizer_params=dict(lr=2e-2),
                       scheduler_fn=torch.optim.lr_scheduler.ExponentialLR,
                       scheduler_params={"gamma":0.98 # max because default␣
↪eval metric for binary is AUC
#                                          "factor":0.001,
#                                          "patience":10,
                                      },
                       mask_type='entmax', # "sparsemax",entmax
                      )
```

```
Device used : cuda
```

1.4.2 Training

[13]:
```
clf.fit(
    X_train=X_train, y_train=y_train.to_numpy().reshape(-1,1),
    eval_set=[(X_train, y_train.to_numpy().reshape(-1,1)), (X_valid, y_valid.
↪to_numpy().reshape(-1,1))],
    eval_name=['train', 'valid'],
    max_epochs=max_epochs,
    patience=50,
    batch_size=batch_size, virtual_batch_size=128,
    num_workers=5,
    drop_last=False
)
```

```
epoch 0  | loss: 5.74451 | train_mse: 38.25975| valid_mse: 23.19162|  0:00:03s
epoch 1  | loss: 1.68352 | train_mse: 3.58135 | valid_mse: 3.79856 |  0:00:07s
epoch 2  | loss: 1.2839  | train_mse: 1.10532 | valid_mse: 0.84834 |  0:00:11s
epoch 3  | loss: 0.83205 | train_mse: 0.92903 | valid_mse: 0.75198 |  0:00:14s
epoch 4  | loss: 0.97688 | train_mse: 0.72547 | valid_mse: 0.70449 |  0:00:18s
epoch 5  | loss: 0.82935 | train_mse: 0.62694 | valid_mse: 0.62339 |  0:00:21s
epoch 6  | loss: 0.66948 | train_mse: 0.60943 | valid_mse: 0.62355 |  0:00:25s
epoch 7  | loss: 0.58083 | train_mse: 0.51828 | valid_mse: 0.53597 |  0:00:29s
epoch 8  | loss: 0.53304 | train_mse: 0.49107 | valid_mse: 0.50156 |  0:00:32s
epoch 9  | loss: 0.50503 | train_mse: 0.48208 | valid_mse: 0.48703 |  0:00:36s
epoch 10 | loss: 0.48887 | train_mse: 0.48857 | valid_mse: 0.49629 |  0:00:40s
epoch 11 | loss: 0.48268 | train_mse: 0.48    | valid_mse: 0.49562 |  0:00:43s
epoch 12 | loss: 0.47765 | train_mse: 0.47406 | valid_mse: 0.48155 |  0:00:47s
```

```
epoch 13 | loss: 0.47425 | train_mse: 0.47792 | valid_mse: 0.48969 |  0:00:50s
epoch 14 | loss: 0.47104 | train_mse: 0.47189 | valid_mse: 0.48468 |  0:00:54s
epoch 15 | loss: 0.47116 | train_mse: 0.47079 | valid_mse: 0.48191 |  0:00:58s
epoch 16 | loss: 0.47007 | train_mse: 0.46905 | valid_mse: 0.4805  |  0:01:01s
epoch 17 | loss: 0.46839 | train_mse: 0.46766 | valid_mse: 0.47946 |  0:01:05s
epoch 18 | loss: 0.46706 | train_mse: 0.46989 | valid_mse: 0.48189 |  0:01:09s
epoch 19 | loss: 0.46604 | train_mse: 0.46456 | valid_mse: 0.47835 |  0:01:12s
epoch 20 | loss: 0.46566 | train_mse: 0.46227 | valid_mse: 0.47595 |  0:01:16s
epoch 21 | loss: 0.46714 | train_mse: 0.4621  | valid_mse: 0.47498 |  0:01:20s
epoch 22 | loss: 0.46555 | train_mse: 0.46431 | valid_mse: 0.47673 |  0:01:23s
epoch 23 | loss: 0.46527 | train_mse: 0.46256 | valid_mse: 0.47763 |  0:01:27s
epoch 24 | loss: 0.46588 | train_mse: 0.46303 | valid_mse: 0.47688 |  0:01:31s
epoch 25 | loss: 0.4663  | train_mse: 0.46177 | valid_mse: 0.47732 |  0:01:34s
epoch 26 | loss: 0.46402 | train_mse: 0.46272 | valid_mse: 0.47431 |  0:01:38s
epoch 27 | loss: 0.46478 | train_mse: 0.46164 | valid_mse: 0.47663 |  0:01:41s
epoch 28 | loss: 0.46461 | train_mse: 0.47426 | valid_mse: 0.47478 |  0:01:45s
epoch 29 | loss: 0.46429 | train_mse: 0.46822 | valid_mse: 0.48775 |  0:01:49s
epoch 30 | loss: 0.46431 | train_mse: 0.45999 | valid_mse: 0.47372 |  0:01:52s
epoch 31 | loss: 0.46308 | train_mse: 0.45964 | valid_mse: 0.47403 |  0:01:56s
epoch 32 | loss: 0.46322 | train_mse: 0.46196 | valid_mse: 0.4813  |  0:02:00s
epoch 33 | loss: 0.46324 | train_mse: 0.46026 | valid_mse: 0.47487 |  0:02:03s
epoch 34 | loss: 0.46263 | train_mse: 0.45936 | valid_mse: 0.47393 |  0:02:07s
epoch 35 | loss: 0.46268 | train_mse: 0.46002 | valid_mse: 0.47471 |  0:02:11s
epoch 36 | loss: 0.46243 | train_mse: 0.45998 | valid_mse: 0.47477 |  0:02:14s
epoch 37 | loss: 0.4637  | train_mse: 0.45997 | valid_mse: 0.47513 |  0:02:18s
epoch 38 | loss: 0.46374 | train_mse: 0.45941 | valid_mse: 0.47422 |  0:02:21s
epoch 39 | loss: 0.46311 | train_mse: 0.45952 | valid_mse: 0.47401 |  0:02:25s
epoch 40 | loss: 0.46337 | train_mse: 0.45954 | valid_mse: 0.47403 |  0:02:29s
epoch 41 | loss: 0.46338 | train_mse: 0.45947 | valid_mse: 0.47405 |  0:02:32s
epoch 42 | loss: 0.46329 | train_mse: 0.45941 | valid_mse: 0.4746  |  0:02:36s
epoch 43 | loss: 0.46225 | train_mse: 0.45987 | valid_mse: 0.47501 |  0:02:40s
epoch 44 | loss: 0.46233 | train_mse: 0.45923 | valid_mse: 0.4742  |  0:02:43s
epoch 45 | loss: 0.46166 | train_mse: 0.459   | valid_mse: 0.4734  |  0:02:47s
epoch 46 | loss: 0.46146 | train_mse: 0.45926 | valid_mse: 0.4739  |  0:02:50s
epoch 47 | loss: 0.46123 | train_mse: 0.45862 | valid_mse: 0.47389 |  0:02:54s
epoch 48 | loss: 0.46135 | train_mse: 0.45963 | valid_mse: 0.47417 |  0:02:58s
epoch 49 | loss: 0.46201 | train_mse: 0.45839 | valid_mse: 0.47302 |  0:03:01s
epoch 50 | loss: 0.46096 | train_mse: 0.45854 | valid_mse: 0.47327 |  0:03:05s
epoch 51 | loss: 0.46136 | train_mse: 0.45862 | valid_mse: 0.4742  |  0:03:08s
epoch 52 | loss: 0.46094 | train_mse: 0.45845 | valid_mse: 0.47429 |  0:03:12s
epoch 53 | loss: 0.46029 | train_mse: 0.45808 | valid_mse: 0.47422 |  0:03:16s
epoch 54 | loss: 0.46094 | train_mse: 0.46181 | valid_mse: 0.47521 |  0:03:19s
epoch 55 | loss: 0.46137 | train_mse: 0.47693 | valid_mse: 0.4993  |  0:03:23s
epoch 56 | loss: 0.46186 | train_mse: 0.50097 | valid_mse: 0.50991 |  0:03:26s
epoch 57 | loss: 0.46159 | train_mse: 0.49026 | valid_mse: 0.50093 |  0:03:30s
epoch 58 | loss: 0.4615  | train_mse: 0.48383 | valid_mse: 0.49984 |  0:03:34s
epoch 59 | loss: 0.46145 | train_mse: 0.4744  | valid_mse: 0.4869  |  0:03:37s
epoch 60 | loss: 0.46138 | train_mse: 0.46828 | valid_mse: 0.48184 |  0:03:41s
```

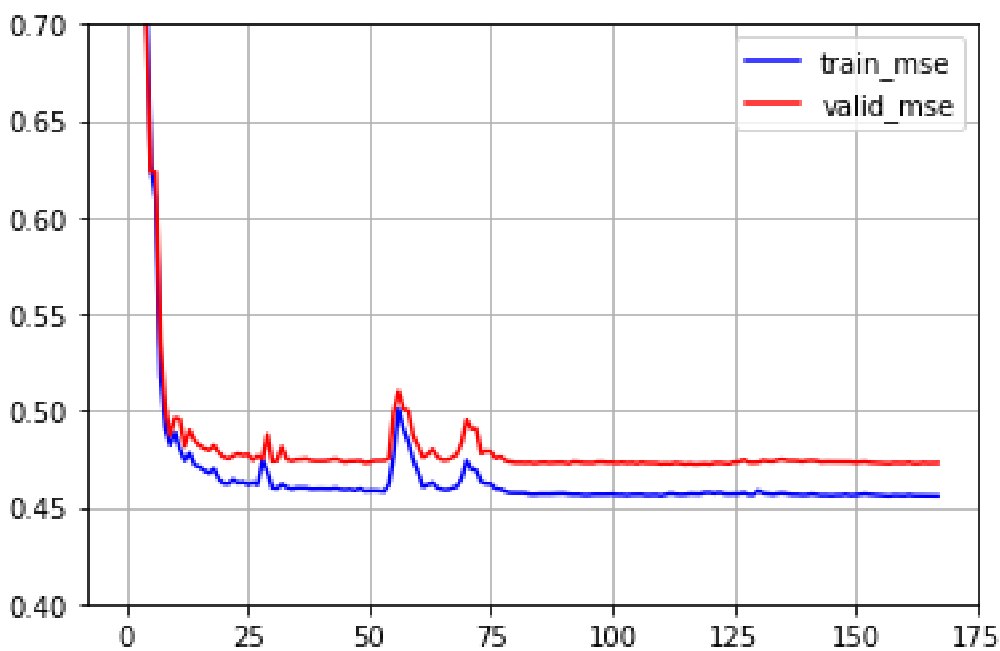
```
epoch 61 | loss: 0.46224 | train_mse: 0.46057 | valid_mse: 0.47586 |  0:03:44s
epoch 62 | loss: 0.46119 | train_mse: 0.46162 | valid_mse: 0.47727 |  0:03:48s
epoch 63 | loss: 0.46086 | train_mse: 0.46265 | valid_mse: 0.47997 |  0:03:52s
epoch 64 | loss: 0.4609  | train_mse: 0.45997 | valid_mse: 0.47598 |  0:03:55s
epoch 65 | loss: 0.46045 | train_mse: 0.45912 | valid_mse: 0.4744  |  0:03:59s
epoch 66 | loss: 0.46113 | train_mse: 0.45891 | valid_mse: 0.4745  |  0:04:02s
epoch 67 | loss: 0.46072 | train_mse: 0.45979 | valid_mse: 0.4756  |  0:04:06s
epoch 68 | loss: 0.46024 | train_mse: 0.46068 | valid_mse: 0.47742 |  0:04:10s
epoch 69 | loss: 0.45998 | train_mse: 0.4651  | valid_mse: 0.48356 |  0:04:13s
epoch 70 | loss: 0.46061 | train_mse: 0.4742  | valid_mse: 0.49535 |  0:04:17s
epoch 71 | loss: 0.46053 | train_mse: 0.46963 | valid_mse: 0.49053 |  0:04:20s
epoch 72 | loss: 0.45978 | train_mse: 0.46915 | valid_mse: 0.49053 |  0:04:24s
epoch 73 | loss: 0.46036 | train_mse: 0.46307 | valid_mse: 0.47783 |  0:04:28s
epoch 74 | loss: 0.45991 | train_mse: 0.46222 | valid_mse: 0.47917 |  0:04:31s
epoch 75 | loss: 0.45997 | train_mse: 0.46204 | valid_mse: 0.4787  |  0:04:35s
epoch 76 | loss: 0.46015 | train_mse: 0.45949 | valid_mse: 0.4753  |  0:04:39s
epoch 77 | loss: 0.45959 | train_mse: 0.45936 | valid_mse: 0.47625 |  0:04:42s
epoch 78 | loss: 0.46015 | train_mse: 0.4581  | valid_mse: 0.47406 |  0:04:46s
epoch 79 | loss: 0.46004 | train_mse: 0.45749 | valid_mse: 0.47358 |  0:04:49s
epoch 80 | loss: 0.45922 | train_mse: 0.45744 | valid_mse: 0.47306 |  0:04:53s
epoch 81 | loss: 0.46022 | train_mse: 0.45735 | valid_mse: 0.4731  |  0:04:57s
epoch 82 | loss: 0.45948 | train_mse: 0.45738 | valid_mse: 0.47297 |  0:05:00s
epoch 83 | loss: 0.45978 | train_mse: 0.45671 | valid_mse: 0.4731  |  0:05:04s
epoch 84 | loss: 0.45971 | train_mse: 0.45665 | valid_mse: 0.47271 |  0:05:08s
epoch 85 | loss: 0.45962 | train_mse: 0.45688 | valid_mse: 0.47308 |  0:05:11s
epoch 86 | loss: 0.45921 | train_mse: 0.45683 | valid_mse: 0.47317 |  0:05:15s
epoch 87 | loss: 0.45958 | train_mse: 0.45687 | valid_mse: 0.47292 |  0:05:18s
epoch 88 | loss: 0.45915 | train_mse: 0.45702 | valid_mse: 0.47303 |  0:05:22s
epoch 89 | loss: 0.45936 | train_mse: 0.4571  | valid_mse: 0.47314 |  0:05:26s
epoch 90 | loss: 0.45973 | train_mse: 0.45717 | valid_mse: 0.47307 |  0:05:29s
epoch 91 | loss: 0.45914 | train_mse: 0.45677 | valid_mse: 0.47264 |  0:05:33s
epoch 92 | loss: 0.45919 | train_mse: 0.4566  | valid_mse: 0.47351 |  0:05:37s
epoch 93 | loss: 0.45853 | train_mse: 0.45666 | valid_mse: 0.47336 |  0:05:40s
epoch 94 | loss: 0.45911 | train_mse: 0.45633 | valid_mse: 0.47297 |  0:05:44s
epoch 95 | loss: 0.45887 | train_mse: 0.45628 | valid_mse: 0.47289 |  0:05:47s
epoch 96 | loss: 0.45805 | train_mse: 0.45657 | valid_mse: 0.47291 |  0:05:51s
epoch 97 | loss: 0.45894 | train_mse: 0.45657 | valid_mse: 0.4734  |  0:05:55s
epoch 98 | loss: 0.459   | train_mse: 0.45655 | valid_mse: 0.47341 |  0:05:58s
epoch 99 | loss: 0.45901 | train_mse: 0.45662 | valid_mse: 0.47312 |  0:06:02s
epoch 100| loss: 0.45889 | train_mse: 0.4568  | valid_mse: 0.47323 |  0:06:05s
epoch 101| loss: 0.45892 | train_mse: 0.45644 | valid_mse: 0.47287 |  0:06:09s
epoch 102| loss: 0.45872 | train_mse: 0.45646 | valid_mse: 0.47321 |  0:06:13s
epoch 103| loss: 0.45844 | train_mse: 0.45621 | valid_mse: 0.47285 |  0:06:16s
epoch 104| loss: 0.45848 | train_mse: 0.45652 | valid_mse: 0.47305 |  0:06:20s
epoch 105| loss: 0.45923 | train_mse: 0.45657 | valid_mse: 0.47264 |  0:06:23s
epoch 106| loss: 0.45858 | train_mse: 0.45658 | valid_mse: 0.47319 |  0:06:27s
epoch 107| loss: 0.45903 | train_mse: 0.45624 | valid_mse: 0.47285 |  0:06:31s
epoch 108| loss: 0.45858 | train_mse: 0.45629 | valid_mse: 0.47288 |  0:06:34s
```

```
epoch 109| loss: 0.45862 | train_mse: 0.45638 | valid_mse: 0.47279 |  0:06:38s
epoch 110| loss: 0.45913 | train_mse: 0.45602 | valid_mse: 0.47237 |  0:06:41s
epoch 111| loss: 0.45881 | train_mse: 0.45652 | valid_mse: 0.47256 |  0:06:45s
epoch 112| loss: 0.45898 | train_mse: 0.45715 | valid_mse: 0.47242 |  0:06:49s
epoch 113| loss: 0.45839 | train_mse: 0.45673 | valid_mse: 0.47294 |  0:06:52s
epoch 114| loss: 0.45859 | train_mse: 0.45651 | valid_mse: 0.47253 |  0:06:56s
epoch 115| loss: 0.45786 | train_mse: 0.4566  | valid_mse: 0.47226 |  0:06:59s
epoch 116| loss: 0.45824 | train_mse: 0.45699 | valid_mse: 0.47256 |  0:07:03s
epoch 117| loss: 0.45851 | train_mse: 0.45692 | valid_mse: 0.47218 |  0:07:07s
epoch 118| loss: 0.45847 | train_mse: 0.45682 | valid_mse: 0.47241 |  0:07:10s
epoch 119| loss: 0.45774 | train_mse: 0.45761 | valid_mse: 0.4725  |  0:07:14s
epoch 120| loss: 0.45835 | train_mse: 0.45762 | valid_mse: 0.47224 |  0:07:18s
epoch 121| loss: 0.45844 | train_mse: 0.45715 | valid_mse: 0.47286 |  0:07:21s
epoch 122| loss: 0.4581  | train_mse: 0.45769 | valid_mse: 0.47292 |  0:07:25s
epoch 123| loss: 0.45825 | train_mse: 0.45691 | valid_mse: 0.47266 |  0:07:29s
epoch 124| loss: 0.45827 | train_mse: 0.45679 | valid_mse: 0.47246 |  0:07:32s
epoch 125| loss: 0.45836 | train_mse: 0.45684 | valid_mse: 0.47329 |  0:07:36s
epoch 126| loss: 0.45839 | train_mse: 0.45694 | valid_mse: 0.47346 |  0:07:39s
epoch 127| loss: 0.45861 | train_mse: 0.45748 | valid_mse: 0.47462 |  0:07:43s
epoch 128| loss: 0.45832 | train_mse: 0.45655 | valid_mse: 0.47329 |  0:07:47s
epoch 129| loss: 0.45871 | train_mse: 0.45642 | valid_mse: 0.4731  |  0:07:50s
epoch 130| loss: 0.45853 | train_mse: 0.45858 | valid_mse: 0.47325 |  0:07:54s
epoch 131| loss: 0.45841 | train_mse: 0.45727 | valid_mse: 0.47426 |  0:07:57s
epoch 132| loss: 0.4589  | train_mse: 0.45697 | valid_mse: 0.47385 |  0:08:01s
epoch 133| loss: 0.45949 | train_mse: 0.4566  | valid_mse: 0.47369 |  0:08:05s
epoch 134| loss: 0.46025 | train_mse: 0.45706 | valid_mse: 0.47446 |  0:08:08s
epoch 135| loss: 0.46    | train_mse: 0.45731 | valid_mse: 0.47476 |  0:08:12s
epoch 136| loss: 0.45967 | train_mse: 0.45693 | valid_mse: 0.4742  |  0:08:16s
epoch 137| loss: 0.45968 | train_mse: 0.45654 | valid_mse: 0.47415 |  0:08:19s
epoch 138| loss: 0.45974 | train_mse: 0.45656 | valid_mse: 0.47411 |  0:08:23s
epoch 139| loss: 0.45942 | train_mse: 0.4563  | valid_mse: 0.47362 |  0:08:26s
epoch 140| loss: 0.45932 | train_mse: 0.45639 | valid_mse: 0.4741  |  0:08:30s
epoch 141| loss: 0.45978 | train_mse: 0.45686 | valid_mse: 0.47428 |  0:08:34s
epoch 142| loss: 0.45862 | train_mse: 0.45648 | valid_mse: 0.47402 |  0:08:37s
epoch 143| loss: 0.45947 | train_mse: 0.45615 | valid_mse: 0.47341 |  0:08:41s
epoch 144| loss: 0.45964 | train_mse: 0.45623 | valid_mse: 0.47355 |  0:08:44s
epoch 145| loss: 0.4593  | train_mse: 0.45602 | valid_mse: 0.47352 |  0:08:48s
epoch 146| loss: 0.45857 | train_mse: 0.45622 | valid_mse: 0.47352 |  0:08:52s
epoch 147| loss: 0.45918 | train_mse: 0.45623 | valid_mse: 0.47345 |  0:08:55s
epoch 148| loss: 0.45924 | train_mse: 0.45657 | valid_mse: 0.47347 |  0:08:59s
epoch 149| loss: 0.45888 | train_mse: 0.45615 | valid_mse: 0.47346 |  0:09:02s
epoch 150| loss: 0.45868 | train_mse: 0.45629 | valid_mse: 0.47318 |  0:09:06s
epoch 151| loss: 0.45882 | train_mse: 0.45661 | valid_mse: 0.47342 |  0:09:10s
epoch 152| loss: 0.45903 | train_mse: 0.45692 | valid_mse: 0.47355 |  0:09:13s
epoch 153| loss: 0.4587  | train_mse: 0.45655 | valid_mse: 0.47311 |  0:09:17s
epoch 154| loss: 0.45882 | train_mse: 0.45635 | valid_mse: 0.47302 |  0:09:21s
epoch 155| loss: 0.45865 | train_mse: 0.45628 | valid_mse: 0.47285 |  0:09:24s
epoch 156| loss: 0.45828 | train_mse: 0.45581 | valid_mse: 0.47271 |  0:09:28s
```

```
epoch 157| loss: 0.45855 | train_mse: 0.45584 | valid_mse: 0.4726  |  0:09:31s
epoch 158| loss: 0.45915 | train_mse: 0.45617 | valid_mse: 0.47294 |  0:09:35s
epoch 159| loss: 0.45847 | train_mse: 0.45604 | valid_mse: 0.47293 |  0:09:38s
epoch 160| loss: 0.4587  | train_mse: 0.45601 | valid_mse: 0.47282 |  0:09:42s
epoch 161| loss: 0.45812 | train_mse: 0.45634 | valid_mse: 0.47315 |  0:09:46s
epoch 162| loss: 0.45828 | train_mse: 0.45602 | valid_mse: 0.4726  |  0:09:49s
epoch 163| loss: 0.45854 | train_mse: 0.45597 | valid_mse: 0.47266 |  0:09:53s
epoch 164| loss: 0.45849 | train_mse: 0.4558  | valid_mse: 0.47299 |  0:09:56s
epoch 165| loss: 0.45851 | train_mse: 0.45598 | valid_mse: 0.47278 |  0:10:00s
epoch 166| loss: 0.45823 | train_mse: 0.45573 | valid_mse: 0.47278 |  0:10:04s
epoch 167| loss: 0.45787 | train_mse: 0.45586 | valid_mse: 0.47288 |  0:10:07s

Early stopping occured at epoch 167 with best_epoch = 117 and best_valid_mse =
0.47218
Best weights from best epoch are automatically used!
```

[14]:

```python
# plot losses
# plt.plot(clf.history['loss'])
plt.plot(clf.history['train_mse'],'b')
plt.plot(clf.history['valid_mse'],'r')
plt.legend(['train_mse','valid_mse'])
plt.grid(True)
plt.gca().set_ylim(0.4, 0.7)
plt.show()
```



[15]:
```
Checkingpreds = clf.predict(X_test)

y_true = y_test

test_score = mean_squared_error(y_pred=preds, y_true=y_true)

print(f"BEST VALID SCORE  : {clf.best_cost}")
print(f"FINAL TEST SCORE  : {test_score}")
```

```
BEST VALID SCORE  : 0.4721820479649035
FINAL TEST SCORE  : 0.4648419391728419
```

[ ]:
```

```

1.4.3 Prepare Drugs at trials

[16]:
```
in_trails = in_trails.drop_duplicates()
```

[17]:
```
in_trails.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9800 entries, 0 to 9799
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   zinc_id  9800 non-null   object
 1   smiles   9800 non-null   object
dtypes: object(2)
memory usage: 229.7+ KB
```

[18]:
```
in_trails['mol'] = in_trails['smiles'].apply(Chem.MolFromSmiles)
```

[19]:
```
in_trails = in_trails.dropna(subset=['mol'])
in_trails['HeavyAtomCount'] = in_trails['mol'].apply(Descriptors.HeavyAtomCount)
in_trails['HAccept'] = in_trails['mol'].apply(Descriptors.NumHAcceptors)
in_trails['HDonor'] = in_trails['mol'].apply(Descriptors.NumHDonors)
in_trails['Heteroatoms'] = in_trails['mol'].apply(Descriptors.NumHeteroatoms)
in_trails['RingCount'] = in_trails['mol'].apply(Descriptors.RingCount)
in_trails['SaturatedRings'] = in_trails['mol'].apply(Descriptors.
 ↪NumSaturatedRings)
in_trails['AliphaticRings'] = in_trails['mol'].apply(Descriptors.
 ↪NumAliphaticRings)
in_trails['AromaticRings'] = in_trails['mol'].apply(Descriptors.
 ↪NumAromaticRings)
in_trails['Ipc'] = in_trails['mol'].apply(Descriptors.Ipc)
in_trails['HallKierAlpha'] = in_trails['mol'].apply(Descriptors.HallKierAlpha)
```

```
in_trails['NumValenceElectrons'] = in_trails['mol'].apply(Descriptors.
 ↪NumValenceElectrons)
in_trails['MolLogP'] = in_trails['mol'].apply(Descriptors.MolLogP)
in_trails['AMW'] = in_trails['mol'].apply(Descriptors.MolWt)
in_trails['NumRotatableBonds'] = in_trails['mol'].apply(Descriptors.
 ↪NumRotatableBonds)
```

[20]:
```
X_in_trails = in_trails[['MolLogP', 'AMW', 'NumRotatableBonds',␣
 ↪'HeavyAtomCount',
       'HAccept', 'HDonor', 'Heteroatoms', 'RingCount', 'SaturatedRings',
       'AliphaticRings', 'AromaticRings', 'Ipc', 'HallKierAlpha',
       'NumValenceElectrons']]
```

[21]:
```
X_in_trails[['MolLogP', 'AMW', 'NumRotatableBonds', 'HeavyAtomCount',
       'HAccept', 'HDonor', 'Heteroatoms', 'RingCount', 'SaturatedRings',
       'AliphaticRings', 'AromaticRings', 'Ipc', 'HallKierAlpha',
       'NumValenceElectrons']] = scaler_xtrain.
 ↪transform(X_in_trails[['MolLogP', 'AMW', 'NumRotatableBonds',␣
 ↪'HeavyAtomCount',
       'HAccept', 'HDonor', 'Heteroatoms', 'RingCount', 'SaturatedRings',
       'AliphaticRings', 'AromaticRings', 'Ipc', 'HallKierAlpha',
       'NumValenceElectrons']])
```

```
/home/moazmohamed/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
/home/moazmohamed/miniconda3/lib/python3.7/site-
packages/pandas/core/indexing.py:1736: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(loc, value[:, i].tolist())
```

1.4.4 Making predictions

[22]:
```
X_in_trails['pred'] = clf.predict(X_in_trails.to_numpy())



X_in_trails['smiles'] = in_trails['smiles']
X_in_trails['zinc_id'] = in_trails['zinc_id']
```

```
/home/moazmohamed/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
/home/moazmohamed/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
```

[23]:
```python
X_in_trails.sort_values("pred", ascending=False).head(20)
```

[23]:

	MolLogP	AMW	NumRotatableBonds	HeavyAtomCount	HAccept	\
6876	12.650479	11.814498	9.000000	10.00	-0.5	
7543	-7.263342	14.462337	5.666667	8.00	11.0	
6675	-7.263342	14.462337	5.666667	8.00	11.0	
8400	-13.182345	14.640834	4.000000	11.00	13.0	
112	-6.007365	14.332942	4.666667	11.25	9.5	
8444	-6.007365	14.332942	4.666667	11.25	9.5	
649	-5.174520	14.068233	4.000000	11.50	4.5	
5137	-5.104591	13.668876	2.333333	11.25	4.0	
1501	-8.128524	11.692729	6.666667	9.25	5.5	
6763	-9.282639	9.892219	2.666667	7.50	8.0	
6817	-9.282639	9.892219	2.666667	7.50	8.0	
6818	-9.282639	9.892219	2.666667	7.50	8.0	
6634	-9.282639	9.892219	2.666667	7.50	8.0	
4743	2.689809	7.862115	5.333333	6.00	1.5	
583	-4.607244	14.404079	3.666667	11.00	5.0	
584	-4.607244	14.404079	3.666667	11.00	5.0	
563	-4.607244	14.404079	3.666667	11.00	5.0	
562	-4.607244	14.404079	3.666667	11.00	5.0	
2884	-2.772754	14.401552	7.666667	11.50	4.0	
5173	0.005805	9.188672	5.000000	7.75	2.5	

	HDonor	Heteroatoms	RingCount	SaturatedRings	AliphaticRings	\
6876	-0.5	-1.5	-1.0	-1.0	0.0	
7543	3.5	18.0	-0.5	1.0	1.0	
6675	3.5	18.0	-0.5	1.0	1.0	
8400	9.5	12.0	1.5	5.0	5.0	
112	6.5	8.5	0.5	3.0	3.0	

8444	6.5	8.5	0.5	3.0	3.0
649	6.5	8.5	1.0	1.0	1.0
5137	6.0	8.0	1.5	2.0	2.0
1501	7.5	9.0	-1.0	-1.0	-1.0
6763	6.5	9.0	0.5	3.0	3.0
6817	6.5	9.0	0.5	3.0	3.0
6818	6.5	9.0	0.5	3.0	3.0
6634	6.5	9.0	0.5	3.0	3.0
4743	2.5	1.5	0.5	3.0	3.0
583	5.5	9.0	0.0	1.0	1.0
584	5.5	9.0	0.0	1.0	1.0
563	5.5	9.0	0.0	1.0	1.0
562	5.5	9.0	0.0	1.0	1.0
2884	5.5	8.5	0.5	-1.0	-1.0
5173	6.5	6.5	-0.5	-1.0	-1.0

	AromaticRings	Ipc	HallKierAlpha	NumValenceElectrons	\
6876	-2.0	1.554755e+07	-2.548780	11.3	
7543	-2.0	4.653066e+04	1.134146	10.2	
6675	-2.0	4.653066e+04	1.134146	10.2	
8400	-2.0	3.222478e+08	1.036585	13.4	
112	-2.0	2.931798e+08	-1.939024	13.0	
8444	-2.0	2.931798e+08	-1.939024	13.0	
649	1.0	7.380236e+08	-5.975610	12.5	
5137	1.0	6.675579e+08	-5.926829	12.1	
1501	-1.0	1.794391e+06	-4.756098	10.6	
6763	-2.0	2.150610e+05	-0.865854	9.0	
6817	-2.0	2.150610e+05	-0.865854	9.0	
6818	-2.0	2.150610e+05	-0.865854	9.0	
6634	-2.0	2.150610e+05	-0.865854	9.0	
4743	-2.0	2.190338e+04	2.146341	7.6	
583	-1.0	1.379306e+08	-4.353659	12.4	
584	-1.0	1.379306e+08	-4.353659	12.4	
563	-1.0	1.379306e+08	-4.353659	12.4	
562	-1.0	1.379306e+08	-4.353659	12.4	
2884	2.0	5.352659e+08	-6.487805	12.4	
5173	0.0	2.767126e+05	-5.817073	8.3	

	pred	smiles	\
6876	54.421494	COC1=C(OC)C(=O)C(C/C=C(\C)CC/C=C(\C)CC/C=C(\C)…	
7543	14.740351	O=S(=O)(O)OC[C@H]1O[C@@H](O[C@]2(COS(=O)(=O)O)…	
6675	14.740351	O=S(=O)(O)OC[C@H]1O[C@@H](O[C@@]2(COS(=O)(=O)O…	
8400	11.149251	OC[C@H]1O[C@@H](O[C@@H]2[C@@H](O)[C@H](O[C@@H]…	
112	9.677798	CC(/C=C/C=C(\C)C(=O)O[C@@H]1O[C@H](CO[C@@H]2O[…	
8444	9.677798	CC(/C=C/C=C(\C)C(=O)O[C@@H]1O[C@H](CO[C@@H]2O[…	
649	9.270931	CC(=O)N[C@H]1[C@H](NC(=O)C[C@H](N)C(=O)N[C@H]2…	
5137	8.891053	CC(=O)N[C@H]1[C@H](NC(=O)C[C@@H]2NC(=O)[C@H](C…	

```
1501    8.775519  C[C@H](N)C(=O)N[C@@H](CO)C(=O)N[C@H](C(=O)N[C@…
6763    8.694906  CC(=O)N[C@H]1[C@H](O[C@@H]2[C@@H](C(=O)O)O[C@@…
6817    8.694906  CC(=O)N[C@H]1[C@H](O[C@@H]2[C@@H](C(=O)O)O[C@@…
6818    8.694906  CC(=O)N[C@@H]1[C@H](O[C@@H]2O[C@H](C(=O)O)[C@@…
6634    8.694906  CC(=O)N[C@H]1[C@H](O[C@@H]2[C@@H](C(=O)O)O[C@@…
4743    8.583335  CC(C)[C@@H](CC[C@@H](C)[C@H]1CC[C@H]2[C@H]3[C@…
583     8.579808  CC[C@H](C)[C@@H]1NC(=O)[C@H](Cc2ccc(O)cc2)NC(=…
584     8.579808  CC[C@H](C)[C@@H]1NC(=O)[C@H](Cc2ccc(O)cc2)NC(=…
563     8.579808  CC[C@H](C)[C@@H]1NC(=O)[C@H](Cc2ccc(O)cc2)NC(=…
562     8.579808  CC[C@H](C)[C@@H]1NC(=O)[C@H](Cc2ccc(O)cc2)NC(=…
2884    8.454035  CCCC[C@@H](NC(=O)[C@H](Cc1c[nH]c2ccccc12)NC(=O…
5173    8.420584  C/C(=N\NC(=N)N)c1cc(NC(=O)CCCCCCCCC(=O)Nc2cc(/…

               zinc_id
6876  ZINC000085427689
7543  ZINC000196037215
6675  ZINC000196037206
8400  ZINC000299818012
112   ZINC000936070151
8444  ZINC000245224178
649   ZINC000255990532
5137  ZINC000169676912
1501  ZINC000169345692
6763  ZINC000096014305
6817  ZINC000096014304
6818  ZINC000096014307
6634  ZINC000096014306
4743  ZINC000049833385
583   ZINC000256015222
584   ZINC000256015224
563   ZINC000256015225
562   ZINC000256015223
2884  ZINC000195761836
5173  ZINC000072266997
```

[44]:
```python
X_in_trails['smiles'].iloc[112]
```

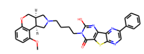
[44]:
```
'CC(/C=C/C=C(\\C)C(=O)O[C@@H]1O[C@H](CO[C@@H]2O[C@H](CO)[C@@H](O)[C@H](O)[C@H]2O
)[C@@H](O)[C@H](O)[C@H]1O)=C\\C=C/C=C(C)/C=C/C=C(\\C)C(=O)O[C@@H]1O[C@H](CO[C@@H
]2O[C@H](CO)[C@@H](O)[C@H](O)[C@H]2O)[C@@H](O)[C@H](O)[C@H]1O'
```

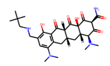
[24]:
```python
best_predicted = X_in_trails[["smiles"]].values[:20,0]
best_predicted_mols = [Chem.MolFromSmiles(x) for x in best_predicted]
```

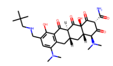
1.4.5 Checking Top 20 compounds

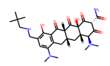
[25]:
```
rdkit.Chem.Draw.MolsToGridImage(best_predicted_mols, molsPerRow=2, maxMols=100,␣
↪subImgSize=(400, 400))
```

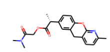
[25]:

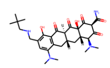


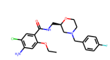


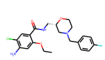


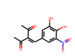


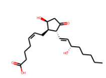


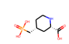


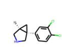


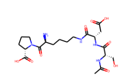


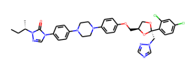


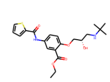


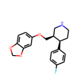


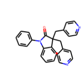


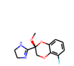


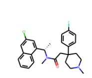












1.5 AutoDock Vina Validation

1.5.1 Binding Affinty of -4.74!!

4-(Benzylideneamino)benzenesulfonamide

1.5.2 Prepare Random Approved Selected Drugs

[26]:
```
endogenous = endogenous.drop_duplicates()
```

[27]:
```
endogenous.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 81519 entries, 0 to 81518
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   zinc_id  81519 non-null  object
 1   smiles   81519 non-null  object
dtypes: object(2)
memory usage: 1.9+ MB
```

[28]:
```
endogenous['mol'] = endogenous['smiles'].apply(Chem.MolFromSmiles)
endogenous = endogenous.dropna(subset=['mol'])
endogenous['HeavyAtomCount'] = endogenous['mol'].apply(Descriptors.
 ↪HeavyAtomCount)
endogenous['HAccept'] = endogenous['mol'].apply(Descriptors.NumHAcceptors)
endogenous['HDonor'] = endogenous['mol'].apply(Descriptors.NumHDonors)
endogenous['Heteroatoms'] = endogenous['mol'].apply(Descriptors.NumHeteroatoms)
endogenous['RingCount'] = endogenous['mol'].apply(Descriptors.RingCount)
endogenous['SaturatedRings'] = endogenous['mol'].apply(Descriptors.
 ↪NumSaturatedRings)
endogenous['AliphaticRings'] = endogenous['mol'].apply(Descriptors.
 ↪NumAliphaticRings)
endogenous['AromaticRings'] = endogenous['mol'].apply(Descriptors.
 ↪NumAromaticRings)
endogenous['Ipc'] = endogenous['mol'].apply(Descriptors.Ipc)
endogenous['HallKierAlpha'] = endogenous['mol'].apply(Descriptors.HallKierAlpha)
endogenous['NumValenceElectrons'] = endogenous['mol'].apply(Descriptors.
 ↪NumValenceElectrons)
endogenous['MolLogP'] = endogenous['mol'].apply(Descriptors.MolLogP)
endogenous['AMW'] = endogenous['mol'].apply(Descriptors.MolWt)
endogenous['NumRotatableBonds'] = endogenous['mol'].apply(Descriptors.
 ↪NumRotatableBonds)
```

```
RDKit WARNING: [08:26:11] Conflicting single bond directions around double bond
at index 1.
RDKit WARNING: [08:26:11]   BondStereo set to STEREONONE and single bond
directions set to NONE.
RDKit WARNING: [08:26:11] Conflicting single bond directions around double bond
at index 1.
RDKit WARNING: [08:26:11]   BondStereo set to STEREONONE and single bond
directions set to NONE.
```

[29]:
```
X_endogenous = endogenous[['MolLogP', 'AMW', 'NumRotatableBonds',␣
↪'HeavyAtomCount',
       'HAccept', 'HDonor', 'Heteroatoms', 'RingCount', 'SaturatedRings',
       'AliphaticRings', 'AromaticRings', 'Ipc', 'HallKierAlpha',
       'NumValenceElectrons']]ZINC000261496575
```

[30]:
```
X_endogenous[['MolLogP', 'AMW', 'NumRotatableBonds', 'HeavyAtomCount',
       'HAccept', 'HDonor', 'Heteroatoms', 'RingCount', 'SaturatedRings',
       'AliphaticRings', 'AromaticRings', 'Ipc', 'HallKierAlpha',
       'NumValenceElectrons']] = scaler_xtrain.
↪transform(X_endogenous[['MolLogP', 'AMW', 'NumRotatableBonds',␣
↪'HeavyAtomCount',
       'HAccept', 'HDonor', 'Heteroatoms', 'RingCount', 'SaturatedRings',
       'AliphaticRings', 'AromaticRings', 'Ipc', 'HallKierAlpha',
       'NumValenceElectrons']])
```

```
/home/moazmohamed/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
/home/moazmohamed/miniconda3/lib/python3.7/site-
packages/pandas/core/indexing.py:1736: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(loc, value[:, i].tolist())
```

1.5.3 Making predictions

[31]:
```
X_endogenous['pred'] = clf.predict(X_endogenousZINC000261496575.to_numpy())
```

```
X_endogenous['smiles'] = endogenous['smiles']
X_endogenous['zinc_id'] = endogenous['zinc_id']
```

```
/home/moazmohamed/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
/home/moazmohamed/miniconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
```

[1]:
```
X_endogenous.sort_values("pred", ascending=False).head(100)
```

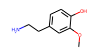
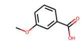
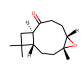
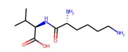
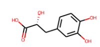
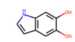
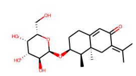
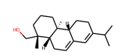
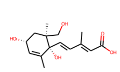
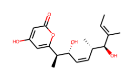
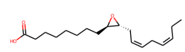
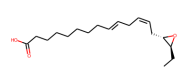
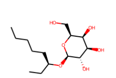
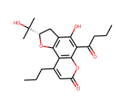
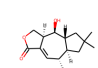
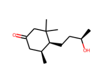
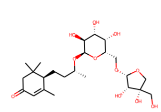
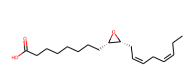
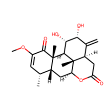
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-6b84e642d225> in <module>
----> 1 X_endogenous.sort_values("pred", ascending=False).head(100)

NameError: name 'X_endogenous' is not defined
```

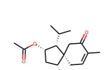
[33]:
```
best_predicted = X_endogenous[["smiles"]].values[:20,0]
best_predicted_mols = [Chem.MolFromSmiles(x) for x in best_predicted]
```

1.5.4 Checking Top 20 compounds

[34]:
```
rdkit.Chem.Draw.MolsToGridImage(best_predicted_mols, molsPerRow=2, maxMols=100,␣
↪subImgSize=(400, 400))
```

[34]:



```
[ ]:
```