- What is MoA?

> In pharmacology, the term Mechanism of Action (MoA) refers to the specific biochemical interaction through which a drug substance produces its pharmacological effect. A mechanism of action usually includes mention of the specific molecular targets to which the drug binds, such as an enzyme or receptor. Receptor sites have specific affinities for drugs based on the chemical structure of the drug, as well as the specific action that occurs there.

- A short brief about the dataset.

> In this competition, the task is predicting multiple targets of the Mechanism of Action (MoA) responses of different samples. Samples are drugs profiled at different time points and doses, and there are more than 20,000 drugs in dataset. Dataset also consists of various group of features and there are more than two hundred targets of enzymes and receptors.

- Machine Learning Approach

> We are supposed to identify the Mechanism of Action (MoA)of a new drug based on the available information of cell viability and gene expressions and their target MoA. In this problem scientists seek to identify a protein target associated with the disease and develop a molecule that can modulate that protein target. As a shorthand to describe the biological activity of a given molecule, scientists assign a label referred to as mechanism-of-action or MoA for short. Here our target variable is MoA and features used to predict MoA are cell viability and gene expressions. We have been provided the information about human cell responses to drug within a pool of 100 cell types and 772 gene expressions in addition we have access to MoA annotations of more that 20,000 drugs. Each drug can have more than one MoA, so this is an interesting part where we need to perform multi label classification on the data.

- Features
- sig_id is the unique sample id
- Features with g- prefix are gene expression features and there are 772 of them (from g-0 to g-771)
- Features with c- prefix are cell viability features and there are 100 of them (from c-0 to c-99)
- cp_type is a binary categorical feature which indicates the samples are treated with a compound or with a control perturbation (trt_cp or ctl_vehicle)
- cp_time is a categorical feature which indicates the treatment duration (24, 48 or 72 hours)
- cp_dose is a binary categorical feature which indicates the dose is low or high (D1 or D2)

Objective and Metric

- There are two groups of target features; scored target features and non-scored target features. Both of those groups consist of binary MoA targets but only the first group is used for the scoring, so this is a multi-label classification problem.
- This is a multi-label binary classification problem, and metric used for the evaluation is mean columnwise log loss. For every row, a probability that the sample had a positive response for each target, has to be predicted. For N rows and M targets, there will be N×M predictions.

log loss = − 1 M ∑ m = 1 M 1 N ∑ i = 1 N [y i , m log (y ^ i , m) + (1 − y i , m) log (1 − y ^ i , m)]

- N is the number of rows (i=1,…,N)
- M is the number of targets (m=1,…,M)
- y^i,m is the predicted probability of the ith row and mth target
- yi,m is the ground truth of the ith row and mth target (1 for a positive response, 0 otherwise)
- log() is the natural logarithm

In [ ]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from scipy import stats
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
#from xgboost import XGBClassifier
from sklearn import model_selection
from sklearn.utils import class_weight
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import log_loss
from imblearn.under_sampling import RandomUnderSampler
import warnings
warnings.filterwarnings('ignore')
from collections import Counter
import datetime
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
import sklearn.metrics as metrics
```

Reading Data

In [4]:

```
# Reading train dataset
train = pd.read_csv("train_features.csv")
train.head()
```

Out[4]:

	sig_id	cp_type	cp_time	cp_dose	g-0	g-1	g-2	g-3	g-4
0	id_000644bb2	trt_cp	24	D1	1.0620	0.5577	-0.2479	-0.6208	-0.1944
1	id_000779bfc	trt_cp	72	D1	0.0743	0.4087	0.2991	0.0604	1.0190
2	id_000a6266a	trt_cp	48	D1	0.6280	0.5817	1.5540	-0.0764	-0.0323
3	id_0015fd391	trt_cp	48	D1	-0.5138	-0.2491	-0.2656	0.5288	4.0620
4	id_001626bd3	trt_cp	72	D2	-0.3254	-0.4009	0.9700	0.6919	1.4180

5 rows × 876 columns

In [3]:

```
# Reading test dataset
test = pd.read_csv("test_features.csv")
test.head()
```

Out[3]:

	sig_id	cp_type	cp_time	cp_dose	g-0	g-1	g-2	g-3	g
0	id_0004d9e33	trt_cp	24	D1	-0.5458	0.1306	-0.5135	0.4408	1.550
1	id_001897cda	trt_cp	72	D1	-0.1829	0.2320	1.2080	-0.4522	-0.36
2	id_002429b5b	ctl_vehicle	24	D1	0.1852	-0.1404	-0.3911	0.1310	-1.43
3	id_00276f245	trt_cp	24	D2	0.4828	0.1955	0.3825	0.4244	-0.58
4	id_0027f1083	trt_cp	48	D1	-0.3979	-1.2680	1.9130	0.2057	-0.58

5 rows × 876 columns

In [5]:

```
#Reading target variable columns
target = pd.read_csv("train_targets_scored.csv")
target.head()
```

Out[5]:

	sig_id	5-alpha_reductase_inhibitor	11-beta-hsd1_inhibitor	acat_inhibitor	acetylchol
0	id_000644bb2	0	0	0	0
1	id_000779bfc	0	0	0	0
2	id_000a6266a	0	0	0	0
3	id_0015fd391	0	0	0	0
4	id_001626bd3	0	0	0	0

5 rows × 207 columns

Exploratory Analysis

Uderstanding shape of test,train and target datasets

In [6]:

```
train.shape
```

Out[6]:

```
(23814, 876)
```

In [7]:

```
test.shape
```

Out[7]:

```
(3982, 876)
```

In [8]:

```
target.shape
```

Out[8]:

```
(23814, 207)
```

In [13]:

```
train.describe()
```

Out[13]:

	cp_time	g-0	g-1	g-2	g-3	
count	23814.000000	23814.000000	23814.000000	23814.000000	23814.000000	2381
mean	48.020156	0.248366	-0.095684	0.152253	0.081971	0.057
std	19.402807	1.393399	0.812363	1.035731	0.950012	1.032
min	24.000000	-5.513000	-5.737000	-9.104000	-5.998000	-6.36
25%	24.000000	-0.473075	-0.562200	-0.437750	-0.429575	-0.47
50%	48.000000	-0.008850	-0.046600	0.075200	0.008050	-0.02
75%	72.000000	0.525700	0.403075	0.663925	0.463400	0.465
max	72.000000	10.000000	5.039000	8.257000	10.000000	10.00

8 rows × 873 columns

In [14]:

```
train.dtypes.value_counts()
```

Out[14]:

```
float64    872
object       3
int64        1
dtype: int64
```

Finding categorical variables in the dataset

In [15]:

```
for col in train.columns:
    if train[col].dtype == "object":
        print(col)
```

```
sig_id
cp_type
cp_dose
```

In [16]:

```
count = 0
for col in train.columns:
    if(col.startswith('g-')):
        count +=1
print(count)
```

```
772
```

In [17]:

```
#Number of features with type gene expression
gene_exp = sum(train.columns.str.startswith('g-'))
gene_exp
```

Out[17]:

```
772
```

In [18]:

```
#Number of features with type cell viability
cell_via = sum(train.columns.str.startswith('c-'))
cell_via
```

Out[18]:

```
100
```

Visualizing Gene and Cell types

> For numerical variables, we specially check the gene expressions. From the distribution plot of gene expression features we can see that most of the samples are normally distributed with mean 0 and values range from -10 to +10.

Gene Expression Features

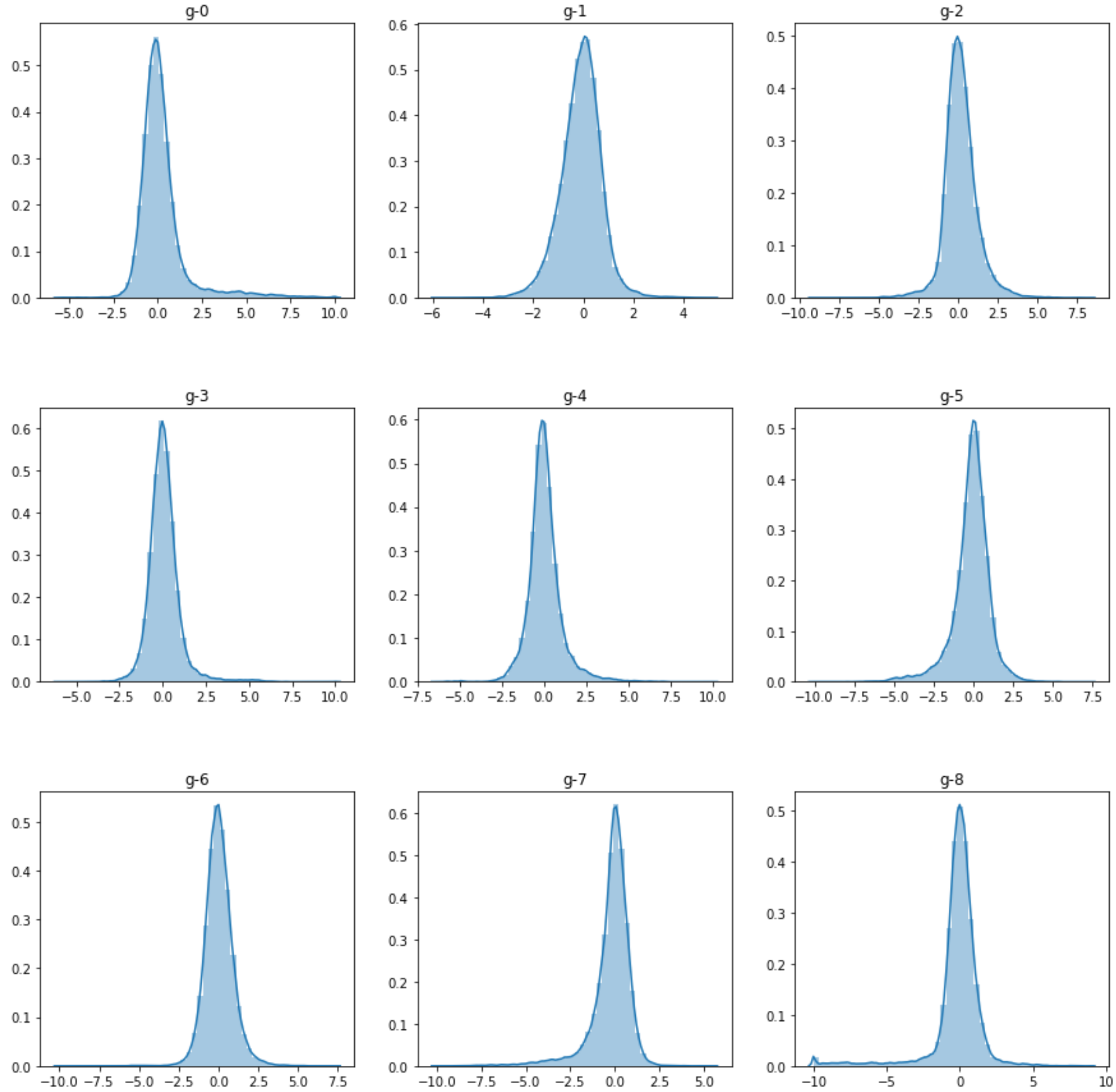
- Gene expression is the amount and type of proteins that are expressed in a cell at any given point in time.
- There are 772 gene expression features and they have g- prefix (g-0 to g-771). Each gene expression feature represents the expression of one particular gene, so there are 772 individual genes are being monitored in this assay.

In [19]:

```
plt.figure(figsize=(15,15))
plt.subplot(3,3,9)
for i in range(9):
    plt.subplot(3,3,i+1)
    sns.distplot(train.iloc[:,i+4])
    plt.title(train.columns[i+4])
    plt.xlabel('')

plt.subplots_adjust(hspace=0.4)
plt.show()
```

```
<ipython-input-19-6b5befb623bf>:4: MatplotlibDeprecationWarning: Adding an
axes using the same arguments as a previous axes currently reuses the earl
ier instance.  In a future version, a new instance will always be created
and returned.  Meanwhile, this warning can be suppressed, and the future b
ehavior ensured, by passing a unique label to each axes instance.
  plt.subplot(3,3,i+1)
```



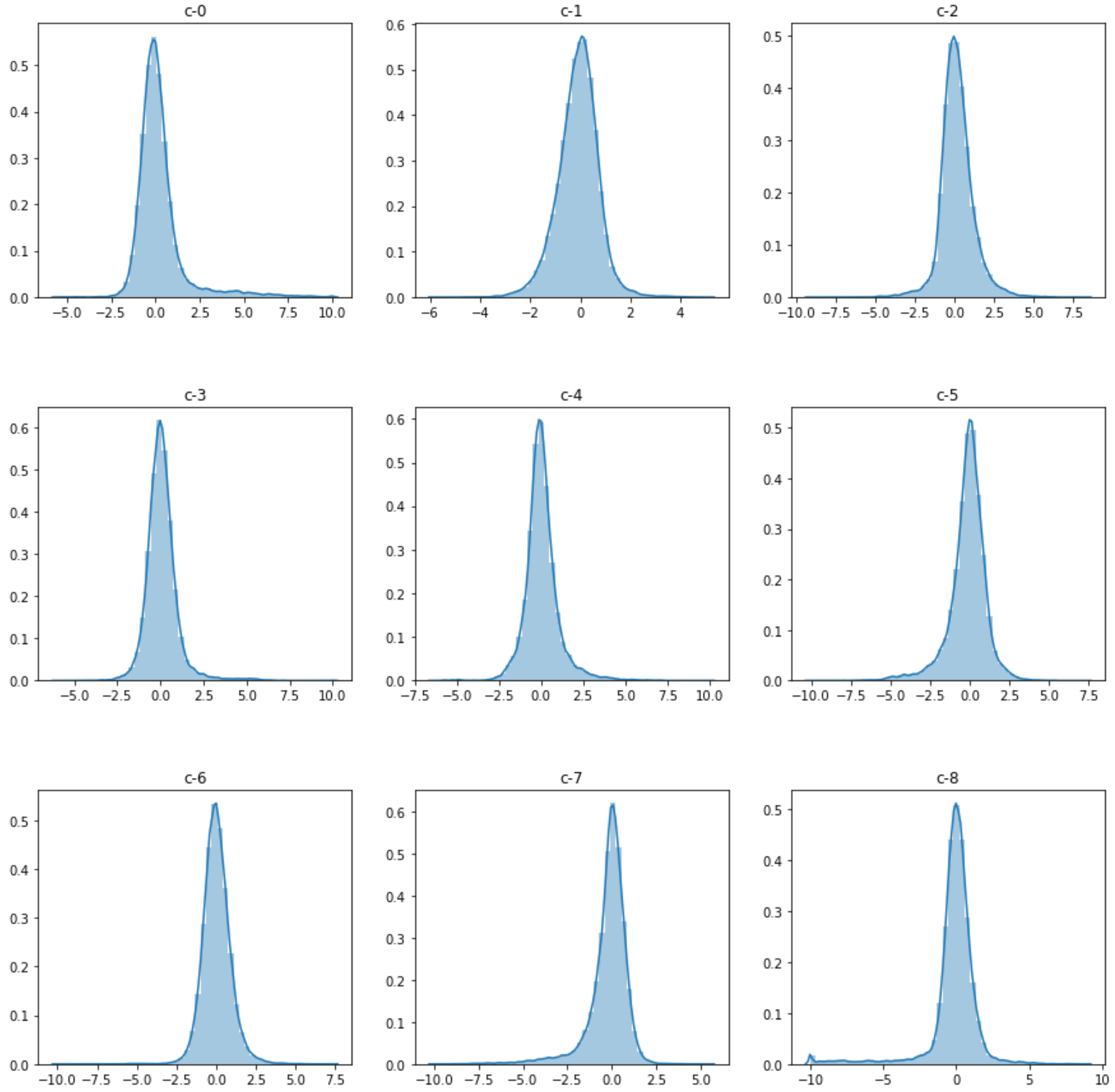
Cell Viability Features

- Cell viability is a measure of the proportion of live, healthy cells within a population. Cell viability assays are used to determine the overall health of cells, optimize culture or experimental conditions.
- There are 100 cell-viability features and they have c- prefix (c-0 to c-99). Each cell-viability feature represents viability of one particular cell line, and all experiments are based on a set of similar cells.

In [13]:

```
plt.figure(figsize=(15,15))
plt.subplot(3,3,9)
for i in range(9):
    plt.subplot(3,3,i+1)
    sns.distplot(train.iloc[:,i+4])
    plt.title(train.columns[i+776])
    plt.xlabel('')

plt.subplots_adjust(hspace=0.4)
plt.show()
```



Categorical Features

- There are three categorical features; cp_type, cp_time and cp_dose. Two of them are binary features and one of them has three unique values, so the cardinality among those features, is very low. All of the categorical features have almost identical distributions in training set.

In [9]:

```
#Check on categorical variables

train['cp_type'].value_counts()
```

Out[9]:

```
trt_cp         21948
ctl_vehicle     1866
Name: cp_type, dtype: int64
```

In [10]:

```
train.cp_time.unique()
```

Out[10]:

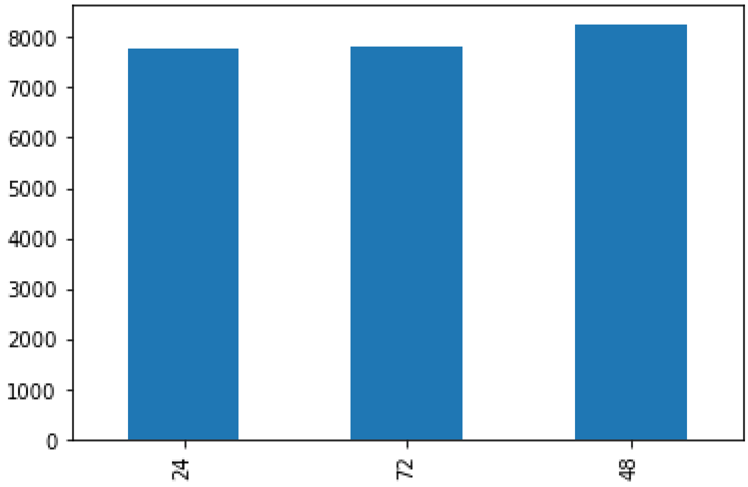
```
array([24, 72, 48], dtype=int64)
```

In [23]:

```
train['cp_time'].value_counts().sort_values().plot(kind = 'bar')
```

Out[23]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24a02e27c70>
```



cp_time

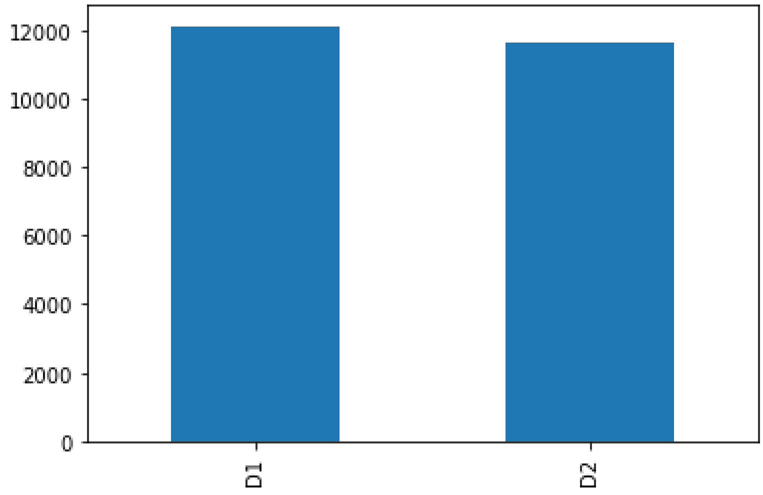
- cp_time is categorical feature in the dataset and it has three unique values; 24, 48 and 72 hours. It indicates the treatment durations of the samples. Sample counts of different cp_time values are very consistent and close to each other in different targets. Sample counts are either extremely close to each other or 48 is slightly higher than the others.

In [24]:

```
train['cp_dose'].value_counts().plot(kind = 'bar')
```

Out[24]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24a02e7e940>
```



cp_dose

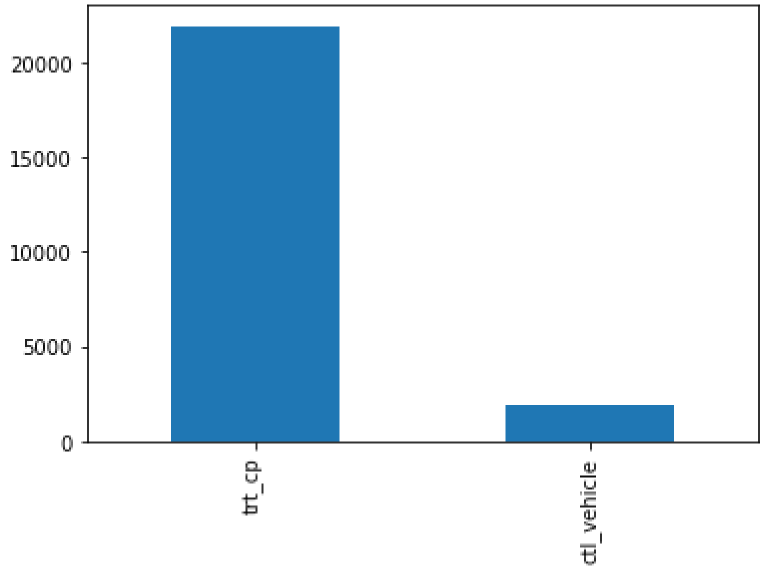
- cp_dose is categorical feature in the dataset and it is also a binary feature. It indicates whether the dose of the samples are either low (D1) or high (D2).

In [25]:

```
train['cp_type'].value_counts().plot(kind = 'bar')
```

Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24a0269fd30>
```



cp_type

- cp_type is categorical feature in the dataset and it is a binary feature. It either means that samples are treated with a compound (trt_cp) or with a control perturbation (ctl_vehicle). Samples treated with control perturbations have no MoAs, thus all of their scored and non-scored target labels are zeros.

Treating Categorical variables for Train and Test Data

In [6]:

```
train = pd.get_dummies(train, columns = ['cp_time'], drop_first=True)
test = pd.get_dummies(test, columns = ['cp_time'], drop_first=True)
```

In [7]:

```
train = pd.get_dummies(train, columns = ['cp_dose'], drop_first=True)
test = pd.get_dummies(test, columns = ['cp_dose'], drop_first=True)
```

In [8]:

```
train = pd.get_dummies(train, columns = ['cp_type'], drop_first=True)
test = pd.get_dummies(test, columns = ['cp_type'], drop_first=True)
```

In [8]:

```
train.head()
```

Out[8]:

	sig_id	g-0	g-1	g-2	g-3	g-4	g-5	g-6	g-7	
0	id_000644bb2	1.0620	0.5577	-0.2479	-0.6208	-0.1944	-1.0120	-1.0220	-0.0326	0
1	id_000779bfc	0.0743	0.4087	0.2991	0.0604	1.0190	0.5207	0.2341	0.3372	-
2	id_000a6266a	0.6280	0.5817	1.5540	-0.0764	-0.0323	1.2390	0.1715	0.2155	0
3	id_0015fd391	-0.5138	-0.2491	-0.2656	0.5288	4.0620	-0.8095	-1.9590	0.1792	-
4	id_001626bd3	-0.3254	-0.4009	0.9700	0.6919	1.4180	-0.8244	-0.2800	-0.1498	-

5 rows × 877 columns

Understanding Target Variables

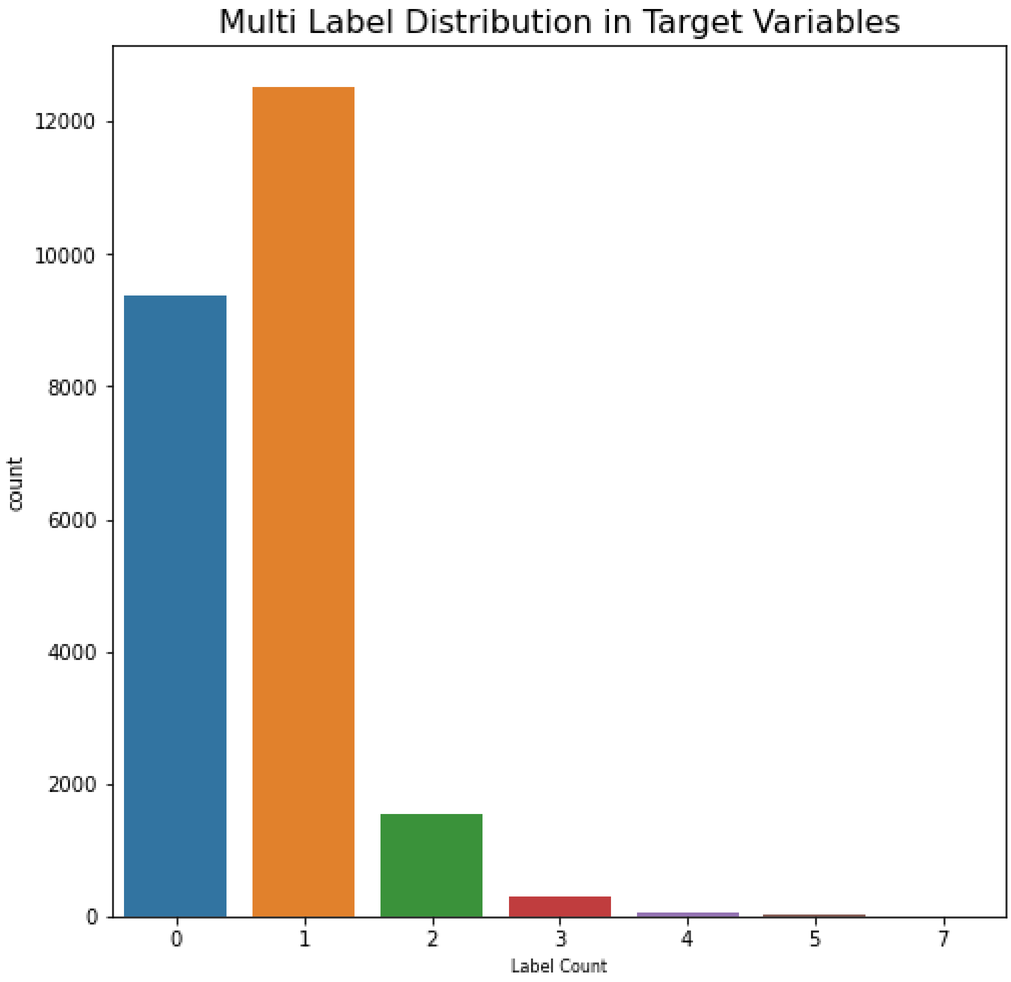
- Target features are categorized into two groups; scored and non-scored target features, and features in both of those groups are binary. The competition score is based on the scored target features but non-scored group can still be used for model evaluation, data analysis and feature engineering.
- It is a multi-label classification problem but one sample can be classified to multiple targets or none of the targets as well. Most of the time, samples are classified to 0 or 1 target, but a small part of the training set samples are classified to 2, 3, 4, 5 and 7 different targets at the same time. Classified targets distributions are not very similar for scored targets and non-scored targets since there is a huge discrepancy of 0 and 1 classified targets.

In [29]:

```
plt.figure(figsize=(8,8))
sns.countplot(target.iloc[:,1:].sum(axis=1))
plt.title("Multi Label Distribution in Target Variables",fontsize=16)
plt.xlabel("Label Count",fontsize=8)
```

Out[29]:

```
Text(0.5, 0, 'Label Count')
```

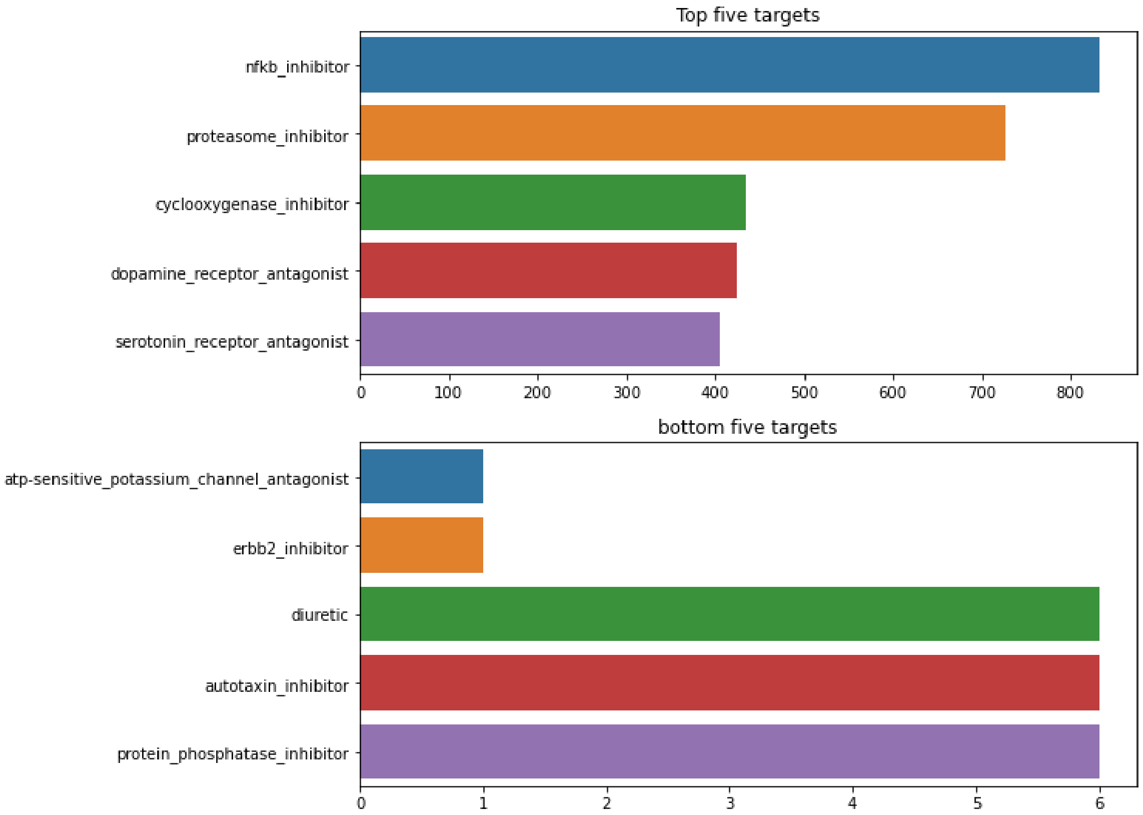


> All the target columns are binary in nature, indicating whether a cell type responds to the drug or not. Since, this is a multi-label classification problem drug samples can have multiple MoA i.e. more than one target variable class can be active. We draw the bar plot to show multi label distribution in target variables.

- Scored Target Features
- The most commmonly classified scored targets are nfkb inhibitor, proteasome inhibitor, cyclooxygenase inhibitor, dopamine receptor antagonist, serotonin receptor antagonist and dna_inhibitor, and there are more than 400 samples classified to each of them. The most rarely classified scored targets are atp-sensitive potassium channel antagonist and erbb2 inhibitor, and there is only one sample classified to each of them. A similar classification distribution is expected in test set.
- There are lots of scored targets classified with the same number of times which suggests there might be a relationship between them.

In [30]:

```
#Visualising top 5 targets and bottom 5 targets
target1  = target.drop(['sig_id'] , axis =1)
top_targets = pd.Series(target1.sum()).sort_values(ascending=False)[:5]
bottom_targets = pd.Series(target1.sum()).sort_values()[:5]
fig, axs = plt.subplots(figsize=(9,9) , nrows=2)
sns.barplot(top_targets.values , top_targets.index , ax = axs[0] ).set(title = "Top five targets")
sns.barplot(bottom_targets.values , bottom_targets.index, ax = axs[1] ).set(title = "bottom five targets")
plt.show()
```



Cleaning Data

In [33]:

```
train.columns[train.isnull().any()]
```

Out[33]:

```
Index([], dtype='object')
```

In [34]:

```
test.columns[test.isnull().any()]
```

Out[34]:

```
Index([], dtype='object')
```

Merging target into single column

In [9]:

```
target['total_cells_reacted'] = target.sum(axis=1)

target['cell_reaction'] = np.minimum(1,target['total_cells_reacted'])
```

In [11]:

```
target.head()
```

Out[11]:

	sig_id	5-alpha_reductase_inhibitor	11-beta-hsd1_inhibitor	acat_inhibitor	acetylchol
0	id_000644bb2	0	0	0	0
1	id_000779bfc	0	0	0	0
2	id_000a6266a	0	0	0	0
3	id_0015fd391	0	0	0	0
4	id_001626bd3	0	0	0	0

5 rows × 209 columns

Defining the target variable

In [10]:

```
# cell reaction as target
target['cell_reaction'].value_counts()
```

Out[10]:

```
1    14447
0     9367
Name: cell_reaction, dtype: int64
```

t-test to check on all features and check if p-value <0.05

- The null hypothesis states that there is no relationship between the two variables being studied (one variable does not affect the other).
- The alternative hypothesis states that the independent variable did affect the dependent variable. ### How do you know if a p-value is statistically significant?
- The level of statistical significance is often expressed as a p-value between 0 and 1. The smaller the p-value, the stronger the evidence that we should reject the null hypothesis.
- A p-value less than 0.05 (typically ≤ 0.05) is statistically significant. It indicates strong evidence against the null hypothesis, Therefore, we reject the null hypothesis, and accept the alternative hypothesis.

In [63]:

```
count = 0
x =[]
for col in train.columns:
    if col in ['sig_id', 'cp_type', 'cp_time', 'cp_dose']:
        continue
    if stats.ttest_ind(train[col], test[col]).pvalue < 0.05:
        print(col, stats.ttest_ind(train[col], test[col]).pvalue)
        x.append(col)
        count += 1
```

```
g-0 0.032543038217202086
g-1 0.019393109454476025
g-3 0.046721309178349664
g-22 6.587624616736739e-05
g-37 0.010911040292799237
g-48 0.030801708455925808
g-50 0.03531252898048682
g-52 0.028092336066332588
g-60 0.003985876456805001
g-72 0.03734120596180097
g-98 0.002596791303651995
g-100 0.008351686077946879
g-101 3.938347554051903e-05
g-105 0.0429096270660344
g-110 0.007002341005554218
g-119 0.0026784043039369777
g-120 0.04036243989718826
g-121 0.018277391157227284
g-134 0.024952122240263698
g-135 5.2762961694835665e-05
g-136 0.014781889784802717
g-139 0.032409758453034926
g-140 0.029888796018104586
g-145 0.04082269273028868
g-150 0.02741917010243746
g-152 0.029540491011946193
g-158 0.006359327029617653
g-165 0.015874095698013277
g-166 0.044642334463927895
g-168 0.04202225455856443
g-169 0.03499856643946845
g-174 0.018682872175197398
g-186 0.008947447524514207
g-193 0.027903058327269626
g-206 0.031924304036266206
g-211 0.0011897547708701196
g-215 0.023199568324813253
g-224 0.03730567820677584
g-235 0.03499601003997585
g-237 0.023174582508069416
g-240 0.04288696473409333
g-248 0.031831549496090336
g-257 0.0006119796151287299
g-267 0.04980450859091937
g-273 0.006297806993275883
g-284 0.04655082243178294
g-298 0.015033343937168856
g-322 0.01744507921578457
g-330 0.011666755951211137
g-332 0.0021244787311601766
g-343 0.03324481374310012
g-346 0.01355309132510231
g-354 0.0016517824313901726
g-360 0.009620025505946095
g-362 0.0324817698501945
g-365 0.012689368176776432
g-367 0.023754819659996564
g-378 0.04656608215292139
g-380 0.0017078489673201087
g-386 0.009849666373976277
g-390 0.03601825397540972
```

```
g-392 0.00014622041338638775
g-396 0.011558383816503901
g-400 0.00234417466908514
g-406 0.03060881366633618
g-410 0.003942306812742894
g-413 0.008357431453948386
g-418 0.002296074513896323
g-433 0.0028952219485856396
g-437 0.03037716243553023
g-439 0.012787195107598747
g-440 0.0009790120173915739
g-445 0.03942676587611877
g-451 0.03521936791875377
g-461 0.026308390115405485
g-473 0.017296065430318728
g-485 0.04687073341670321
g-486 0.013919900233126406
g-489 0.0010059622726513917
g-491 0.0029324527830305452
g-492 0.00048123771140560774
g-498 0.012101192831340432
g-503 0.0015522760171656617
g-518 0.0160779468086828
g-526 0.006223261683449866
g-528 0.01243903801888239
g-554 0.005526846061505236
g-568 0.03981859366092907
g-574 0.0009977307203055188
g-577 0.017459489508530397
g-586 0.024064208550529632
g-600 0.002201740921036897
g-619 0.016276470302135262
g-628 0.0013193596507959106
g-655 0.01974989210697146
g-656 0.0001817267948382975
g-664 0.005051744552255386
g-672 0.03407566901525629
g-675 0.011914459074866945
g-683 0.03929606627209026
g-700 0.0037311306453646255
g-702 0.0010125897831886554
g-708 0.012301558853143693
g-709 0.0398328066319844
g-719 0.04875188668684708
g-723 0.029252768384844995
g-725 0.026614623630592568
g-743 0.040185681769053454
g-744 0.036928324629596374
g-745 0.018719801905843145
g-757 0.04915580418862598
g-761 0.005939948820549039
g-767 0.011931339152979633
g-770 0.010495387864363378
g-771 0.02066501430630159
c-9 0.00676768179495497
c-10 0.027463106489330136
c-18 0.027953170579559945
c-22 0.012676157706292834
c-49 0.04890393979396845
c-70 0.03525132105881509
c-97 0.03324866397008827
```

```
c-98 0.03673769845220116
cp_type_trt_cp 0.01292041444027295
```

In [64]:

```
count
```

Out[64]:

```
124
```

- As the above listed 124 features have p-value < 0.05, hence we can say that those features are highly dependent on target variable.