The **Mechanism of Action** prediction problem is provided by *Laboratory for Innovation Science at Harvard (LISH)*, and the NIH Common Funds *Library of Integrated Network-Based Cellular Signatures (LINCS)*.

The *goal* of this project is to make drug development more advance and fast by predicting the MoA of drug quickly. Generally researcher do is that they treat sample cell with a drug and finds similarity in *genomic databases* which costs lots of time and money and makes this process very slow. So, the goal of this project is to speed-up this process and make it more efficient.

What is Mechanism of Action (MoA)?
Now a days, drug development has become target based by understanding of the underlying biological mechanism of a disease. So, to describe biological activity of a drug or molecule scientists assigned a label called *Mechanism of Action*.

How the Mechanism of Action of a new drug is being determined?
One approach is to treat a sample of human cells with the drug and then analyze the cellular responses with algorithms that search for similarity to known patterns in large genomic databases, such as libraries of gene expression or cell viability patterns of drugs with known MoAs.

```
#importing libraries
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
import seaborn as sns


'''
downloading dataset using curlwget. curlwget is an extension in chrome browser,
If a dataset is very large which we don't want to download in our system, then usi
we can access them.
To use curlwget use following steps:
1. add curlwget extension to chrome
2. now, go to the url from where you want to download the dataset
3. click on download and immediately cancel the download
4. click on curlwget extension from top-right corner
5. copy the text and pase in notebook followed by exclamation(!) mark
'''
!wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (X

    --2021-03-17 10:13:18--  https://storage.googleapis.com/kaggle-competitions-d
    Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.141.128,
    Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.141.128|
    HTTP request sent, awaiting response... 200 OK
    Length: 67874518 (65M) [application/zip]
    Saving to: ‘lish-moa.zip’
```

```
lish-moa.zip        100%[===================>]  64.73M   131MB/s    in 0.5s

2021-03-17 10:13:19 (131 MB/s) - ‘lish-moa.zip’ saved [67874518/67874518]
```

```
#unzipping the downloaded zip file of dataset
!unzip 'lish-moa.zip'
```

```
Archive:  lish-moa.zip
  inflating: sample_submission.csv
  inflating: test_features.csv
  inflating: train_drug.csv
  inflating: train_features.csv
  inflating: train_targets_nonscored.csv
  inflating: train_targets_scored.csv
```

```
#Reading datasets
df = pd.read_csv('train_features.csv')
tts = pd.read_csv('train_targets_scored.csv')
ttns = pd.read_csv('train_targets_nonscored.csv')
td = pd.read_csv('train_drug.csv')
te_df = pd.read_csv('test_features.csv')
```

```
df.shape
```

```
(23814, 876)
```

```
df.head()
```

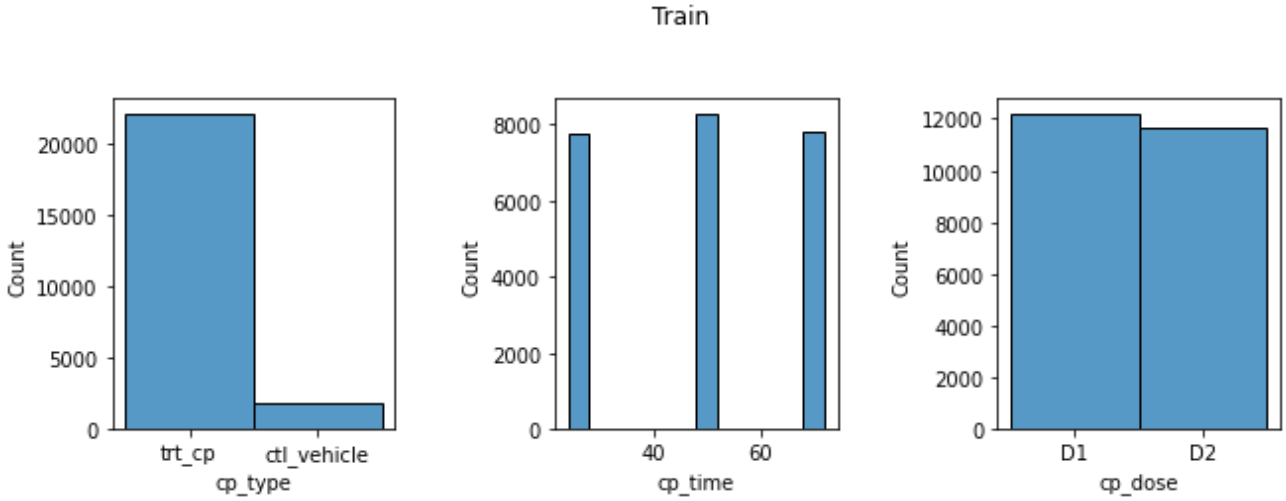
	sig_id	cp_type	cp_time	cp_dose	g-0	g-1	g-2	g-3	g-4
0	id_000644bb2	trt_cp	24	D1	1.0620	0.5577	-0.2479	-0.6208	-0.1944
1	id_000779bfc	trt_cp	72	D1	0.0743	0.4087	0.2991	0.0604	1.0190
2	id_000a6266a	trt_cp	48	D1	0.6280	0.5817	1.5540	-0.0764	-0.0323
3	id_0015fd391	trt_cp	48	D1	-0.5138	-0.2491	-0.2656	0.5288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

5 rows × 876 columns

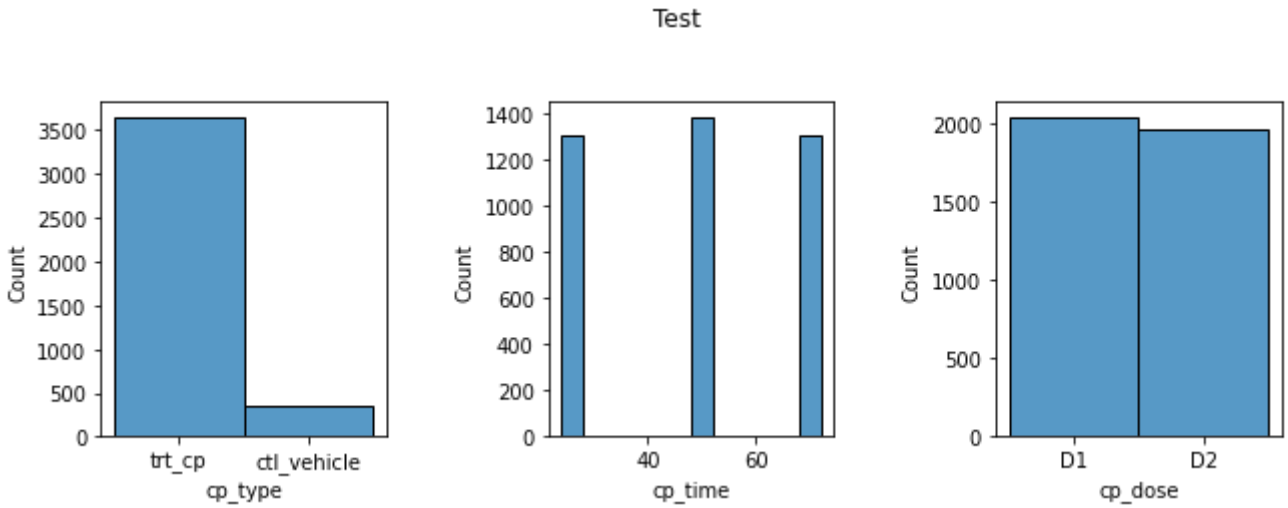
In the datast we have 3 categorical variable cp_type, cp_time and cp_dose so let's visualize their histogram to get understanding about number of unique elements present in these three columns.

```
fig,axes=plt.subplots(1,3, figsize=(10,4))
plt.tight_layout(pad=5)
plt.suptitle('Train')
sns.histplot(df['cp_type'],binwidth=4,ax=axes[0])
```

```
sns.histplot(df['cp_time'], binwidth=4,ax=axes[1])
sns.histplot(df['cp_dose'],binwidth=4,ax=axes[2])
plt.show()
```




```
fig,axes=plt.subplots(1,3, figsize=(10,4))
plt.tight_layout(pad=5)
plt.suptitle('Test')
sns.histplot(te_df['cp_type'],binwidth=4,ax=axes[0])
sns.histplot(te_df['cp_time'], binwidth=4,ax=axes[1])
sns.histplot(te_df['cp_dose'],binwidth=4,ax=axes[2])
plt.show()
```



In both train and test datasets:

cp_type column is imbalanced, trt_cp is dominating over ctl_vehicle.

cp_time column is balanced.

cp_dose column is also balanced.

Here cp_type and cp_dose is nominal data and cp_dose is ordinal data so accordingly we will encode these categorical data.

```
#embedding cp_type, cp_time and cp_dose categorical columns of train dataset
df['cp_type'] = df['cp_type'].map({'trt_cp':0, 'ctl_vehicle':1})
```

```
df['cp_time'] = df['cp_time'].map({24:0, 48:1, 72:2})
df['cp_dose'] = df['cp_dose'].map({'D1':0, 'D2':1})


#embedding cp_type, cp_time and cp_dose categorical columns of test dataset
te_df['cp_type'] = te_df['cp_type'].map({'trt_cp':0, 'ctl_vehicle':1})
te_df['cp_time'] = te_df['cp_time'].map({24:0, 48:1, 72:2})
te_df['cp_dose'] = te_df['cp_dose'].map({'D1':0, 'D2':1})


#checking for duplicate rowsin training dataset
print('Number of duplicate rows in train dataset :',df.duplicated().sum())
print('Number of duplicate sig_id of train dataset :',df['sig_id'].duplicated().su
print('Number of duplicate rows in train dataset except sig_id column:',df.drop('s
```

```
Number of duplicate rows in train dataset : 0
Number of duplicate sig_id of train dataset : 0
Number of duplicate rows in train dataset except sig_id column: 0
```

There is no any duplicate rows in train dataset

```
#Checking for null values in train dataset
print('Number of null values in train dataset :',df.isnull().any().sum())
```

```
Number of null values in train dataset : 0
```

```
#Seperating gene and cell columns
gene_cols = [c for c in df.columns if c.startswith('g-')]
cell_cols = [c for c in df.columns if c.startswith('c-')]


print('Number of gene expression columns :',len(gene_cols))
print('Number of cell viability columns :',len(cell_cols))
```

```
Number of gene expression columns : 772
Number of cell viability columns : 100
```

```
df[cell_cols].describe()
```

	c-0	c-1	c-2	c-3	c-4	c
count	23814.000000	23814.000000	23814.000000	23814.000000	23814.000000	23814.000(

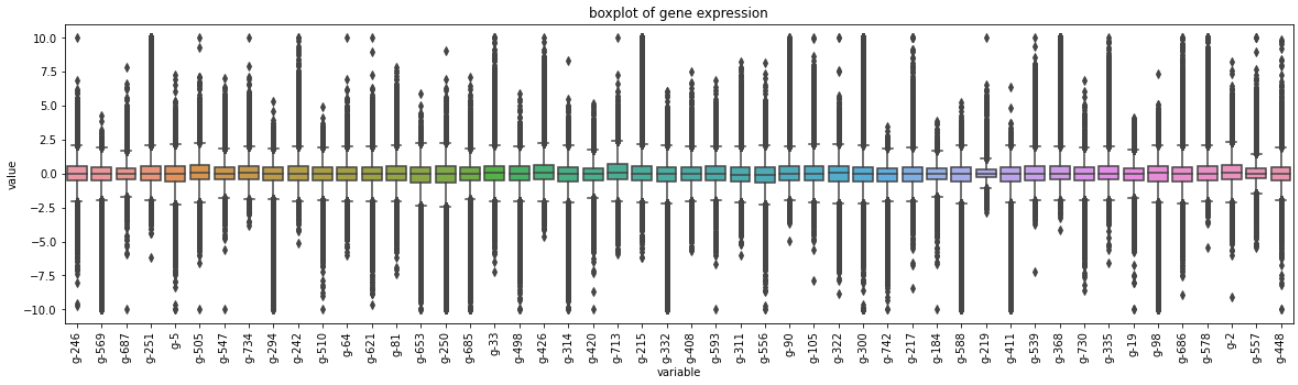
```
df[gene_cols].describe()
```

	g-0	g-1	g-2	g-3	g-4	g
count	23814.000000	23814.000000	23814.000000	23814.000000	23814.000000	23814.000(
mean	0.248366	-0.095684	0.152253	0.081971	0.057347	-0.138
std	1.393399	0.812363	1.035731	0.950012	1.032091	1.179
min	-5.513000	-5.737000	-9.104000	-5.998000	-6.369000	-10.000(
25%	-0.473075	-0.562200	-0.437750	-0.429575	-0.470925	-0.602
50%	-0.008850	-0.046600	0.075200	0.008050	-0.026900	-0.015
75%	0.525700	0.403075	0.663925	0.463400	0.465375	0.510
max	10.000000	5.039000	8.257000	10.000000	10.000000	7.282(

8 rows × 772 columns

Plotting boxplots of 50 random gene columns to see the trend and check for outliers.

```
plt.figure(figsize=(20,5))
sns.boxplot(x="variable", y="value", data=pd.melt(df[gene_cols].iloc[:,random.samp
plt.title('boxplot of gene expression')
plt.xticks(rotation=90)
plt.show()
```



From the boxplot we can see that value of each gene expression lies between 10 to -10. So, we can't classify them as outliers there is high variance in gene expression and we have to normalize it.

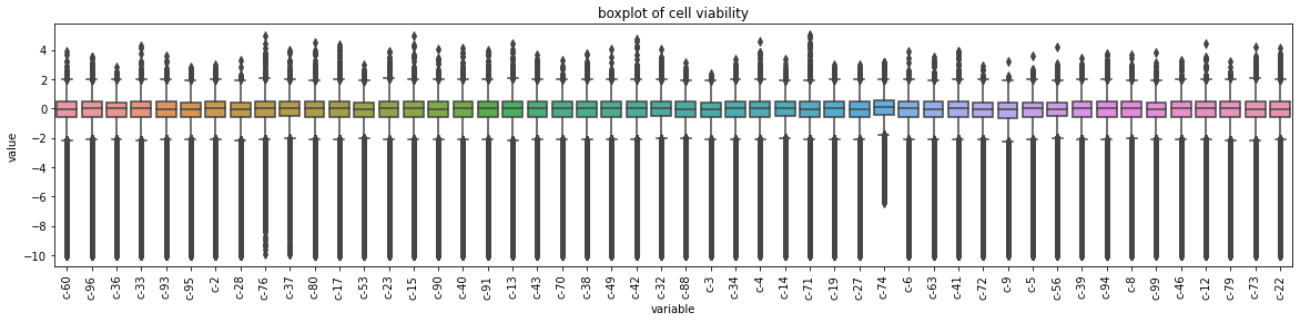
```
'''
Since we have used 50 gene featured to visualize using boxplot, but
outliers can also be in other columns. So to check for outliers we are
using threshold values. If value in a feature increses or decreases from
threshold value then the will be outliers.
'''

count=0
for i in (gene_cols):
  if (max(df[i])-np.mean(df[i])>11 or min(df[i])-np.mean(df[i])<-11):
    print(i, np.mean(df[i]), max(df[i]), min(df[i]))
    count+=1
if count==0:
  print('There is no any data point which is greater than 11 or less than -11')
```

```
There is no any data point which is greater than 11 or less than -11
```

Plotting boxplots of 50 random cell columns to see the trend and check for outliers.

```
plt.figure(figsize=(20,4))
sns.boxplot(x="variable", y="value", data=pd.melt(df[cell_cols].iloc[:,random.samp
plt.title('boxplot of cell viability')
plt.xticks(rotation=90)
plt.show()
```



From the boxplot we can see that value of each cell viability lies between 10 to -10. So, we can't classify them as outliers,there is high variance in cell viability and we also have to normalize it.

```
'''
```

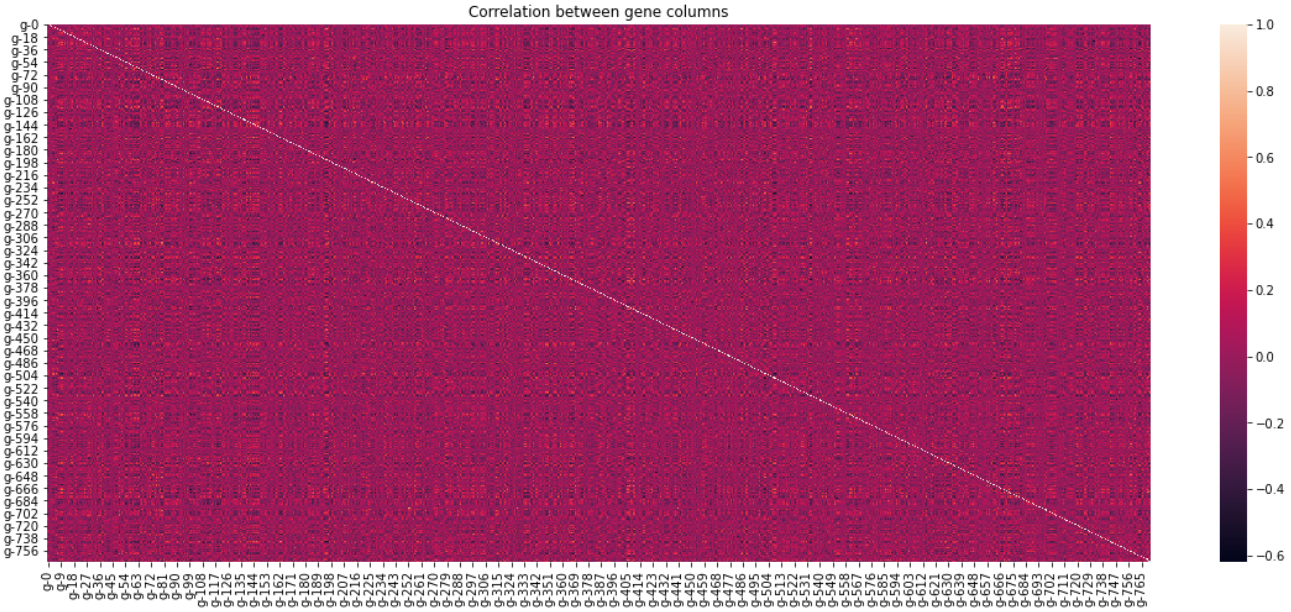
```
Since here also we have used 50 cell featured to visualize using boxplot, but
outliers can also be in other columns. So to check for outliers we are again
using threshold values. If value in a feature increses or decreases from
threshold value then the will be outliers.
'''

count=0
for i in (cell_cols):
  if (max(df[i])-np.mean(df[i])>7 or min(df[i])-np.mean(df[i])<-11):
    print(i, np.mean(df[i]), max(df[i]), min(df[i]))
    count+=1
if count==0:
  print('There is no any data point which is greater than 7 or less than -11')
```

```
There is no any data point which is greater than 7 or less than -11
```

Visualizing correlation between gene columns

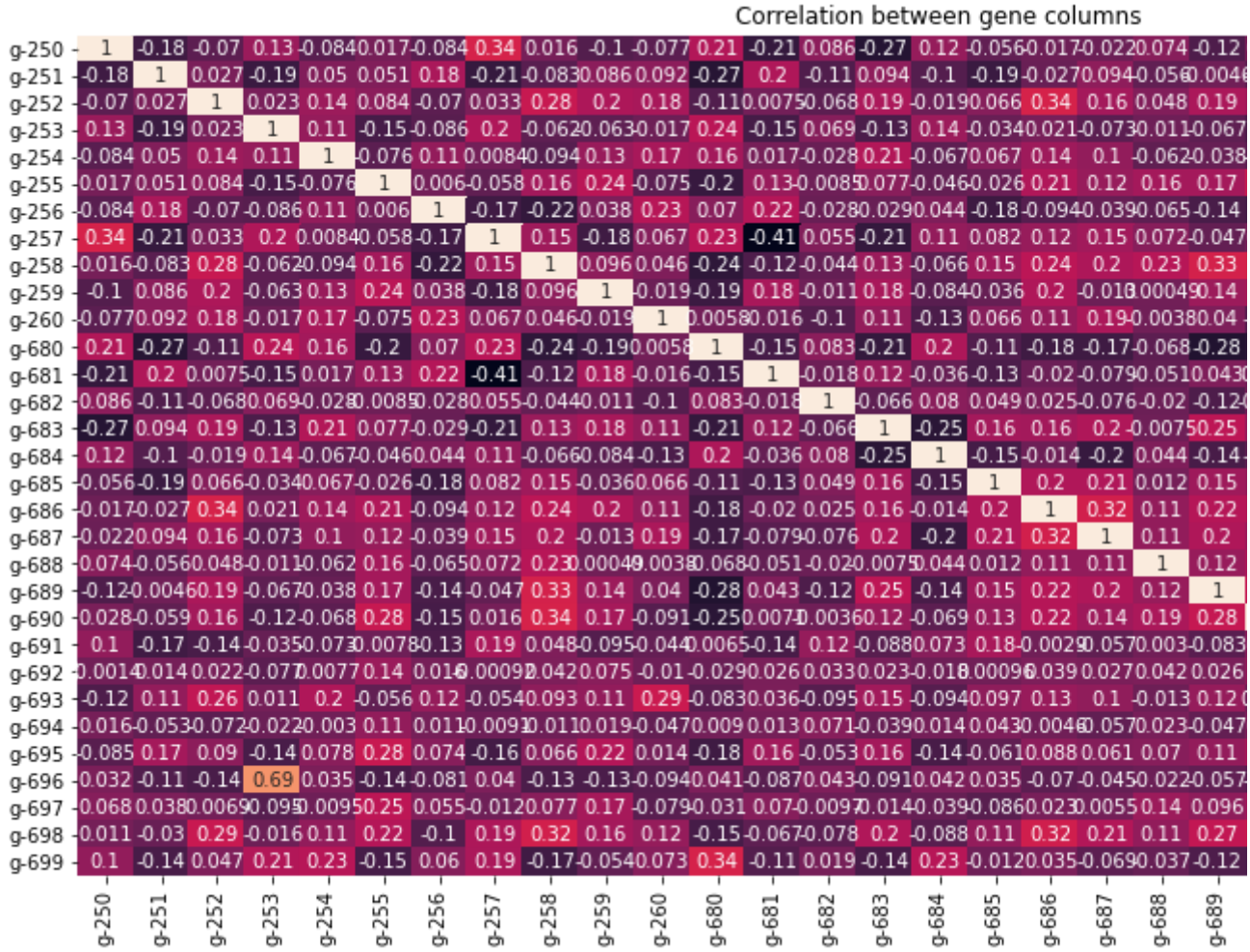
```
plt.figure(figsize=(20,8))
sns.heatmap(df[gene_cols].corr(method='spearman'))
plt.title('Correlation between gene columns')
plt.show()
```



From this heatmap we cannot clearly say that which columns are highly correlated, but we can see that there is no any very dark points in the heatmap and there are some light dots between

g-250 to g-270 and 680 to 700, let's zoom into them.

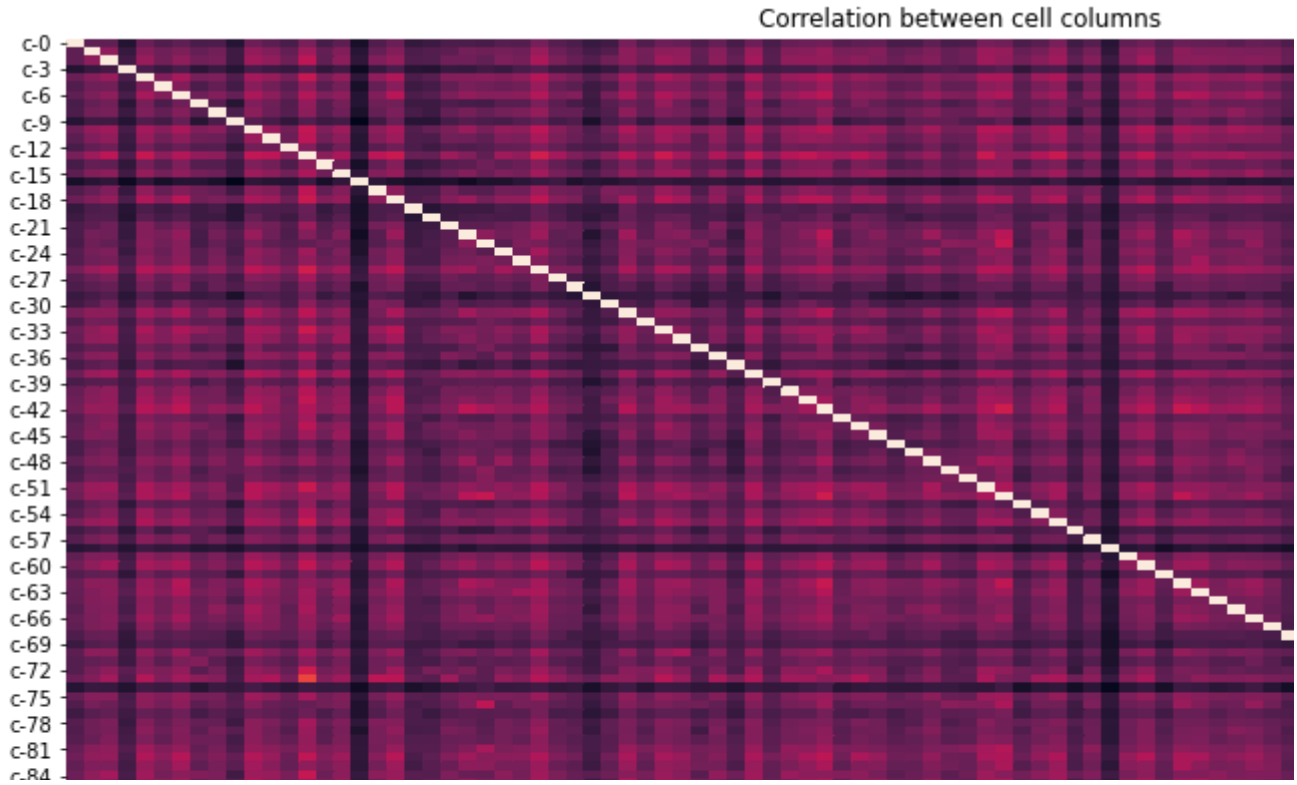
```
plt.figure(figsize=(20,8))
sns.heatmap(df[gene_cols[250:261]+gene_cols[680:700]].corr(method='spearman'), ann
plt.title('Correlation between gene columns')
plt.show()
```



Hence there is correlation between g-253 and g-696. Since we have large number of gene feature so these small correlation between some features can be ignored.

visualizing correlation between cell columns

```
plt.figure(figsize=(20,8))
sns.heatmap(df[cell_cols].corr(method='spearman'))
plt.title('Correlation between cell columns')
plt.show()
```

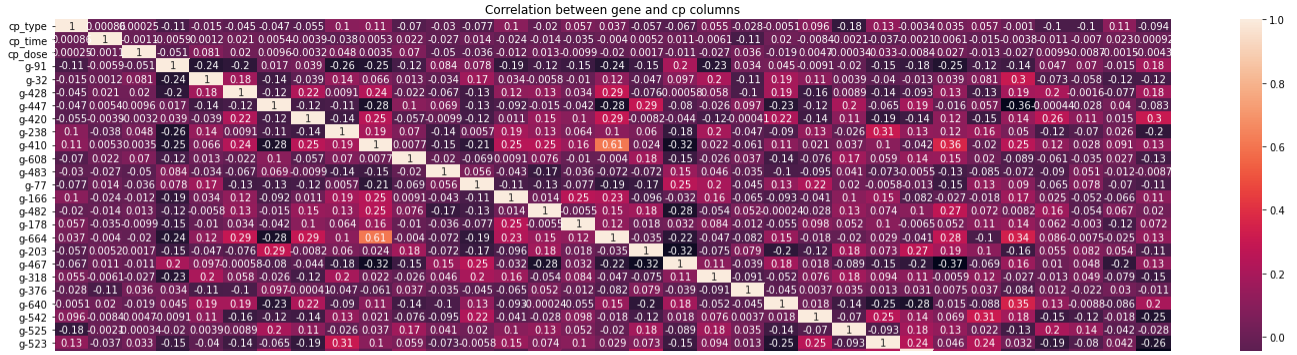


Spearman Rank correlation rank lies between 1 and -1, so in cell columns there is no such high correlation. We can see some dark lines in heatmap such as c-16, c-85, c-74 but their correlation cannot less than 0.2. One interesting thing we can see in this heatmap is that no two columns are negatively correlated in cell columns.

Correlation between categorical columns(cp column) and gene columns

We cannot find correlation between all gene columns with cp columns, so we will randomly select 30 gene columns and check for correlation.

```
plt.figure(figsize=(25,8))
sns.heatmap(df[['cp_type','cp_time','cp_dose']+random.sample(gene_cols,30)].corr(m
plt.title('Correlation between gene and cp columns')
plt.show()
```

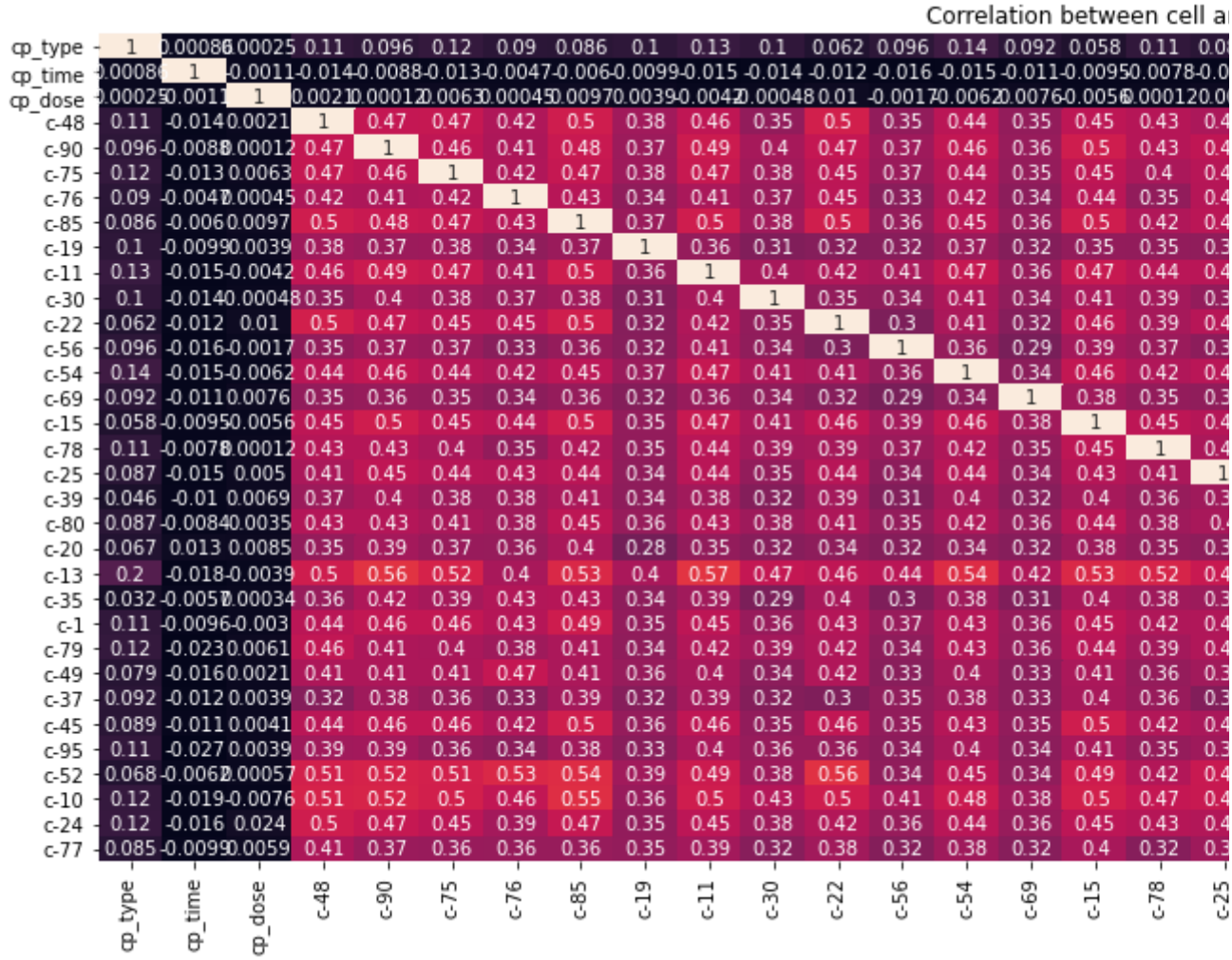


From above heatmap we can clearly see that there is no correlation between cp columns and gene columns.

Correlation between categorical columns(cp column) and cell columns

We cannot find correlation between all cell columns with cp columns, so we will randomly select 30 cell columns and check for correlation.

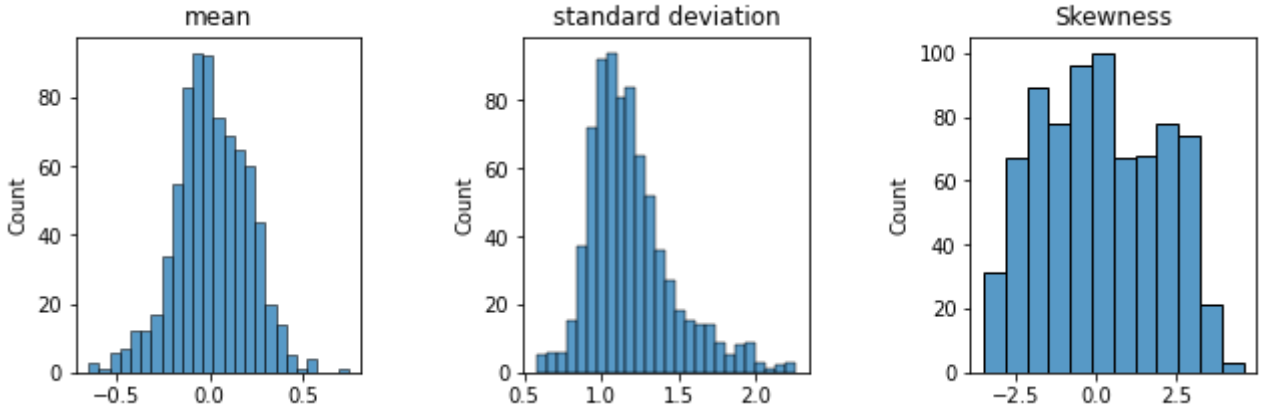
```
plt.figure(figsize=(25,8))
sns.heatmap(df[['cp_type','cp_time','cp_dose']+random.sample(cell_cols,30)].corr(m
plt.title('Correlation between cell and cp columns')
plt.show()
```



From above heatmap we can clearly see that there is no correlation between cp columns and cell columns.

Plotting mean, standard deviation and skewness of gene columns

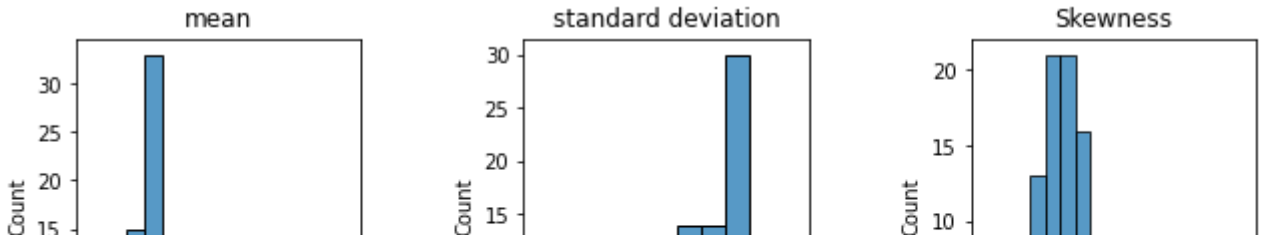
```
fig,axes=plt.subplots(1,3, figsize=(10,4))
plt.tight_layout(pad=5)
sns.histplot(np.mean(df[gene_cols]),ax=axes[0])
axes[0].set_title('mean')
sns.histplot(np.std(df[gene_cols]),ax=axes[1])
axes[1].set_title('standard deviation')
sns.histplot(df[gene_cols].skew(),ax=axes[2])
axes[2].set_title('Skewness')
plt.show()
```



Mean of gene columns lies between 1 and -1 and there are large number of columns whose means are near zero, standard deviation of the gene colums are from 0.5 to 2.5. Skewness is greater than 2.5 and less than -2.5, hence gene data is highly skewed.

Plotting mean, standard deviation and skewness of cell columns

```
fig,axes=plt.subplots(1,3, figsize=(10,4))
plt.tight_layout(pad=5)
sns.histplot(np.mean(df[cell_cols]),ax=axes[0])
axes[0].set_title('mean')
sns.histplot(np.std(df[cell_cols]),ax=axes[1])
axes[1].set_title('standard deviation')
sns.histplot(df[cell_cols].skew(),ax=axes[2])
axes[2].set_title('Skewness')
plt.show()
```



Mean of gene columns lies between 0 and -0.6 and there are large number of columns whose means are near -0.5, standard deviation of the gene colums are from 1 to 2.5. Skewness is less than -2.0, hence gene data is highly left skewed.

Checking train_drug dataset

```
td['drug_id'].value_counts()
```

```
cacb2b860    1866
87d714366     718
9f80f3f77     246
8b87a7a83     203
5628cb3ee     202
              ...
122c63321       1
5bbc89d67       1
7226e2204       1
828099693       1
42feaa183       1
Name: drug_id, Length: 3289, dtype: int64
```

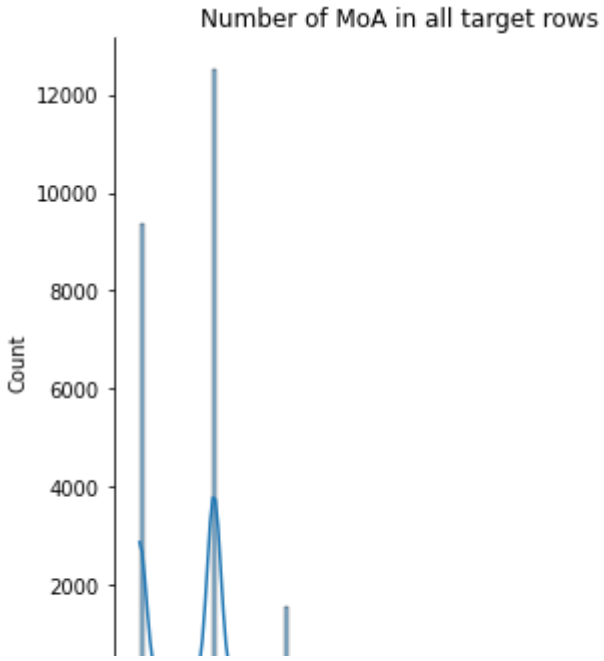
From above analysis we can say that some drugs are used many times to test on sample cells.

Checking for count of MoAs in each row of target scored dataset

```
tts.sum(axis=1).value_counts()
```

```
1    12532
0     9367
2     1538
3      303
4       55
5       13
7        6
dtype: int64
```

```
sns.displot(np.sum(tts.drop('sig_id', axis=1), axis=1), kde=True)
plt.xlabel('Number of MoAs')
plt.title('Number of MoA in all target rows')
plt.show()
```



From above graph we can see that maximum samples have either 0 or 1 MoAs and there are very less samples which have more than 1 MoAs.

```
#getting which MoA occurs most and which least
pd.DataFrame(np.sum(tts.drop('sig_id', axis=1), axis=0)).sort_values(by=[0], ascen
```

	0
nfkb_inhibitor	832
proteasome_inhibitor	726
cyclooxygenase_inhibitor	435
dopamine_receptor_antagonist	424
serotonin_receptor_antagonist	404
...	...
elastase_inhibitor	6
steroid	6
atm_kinase_inhibitor	6
erbb2_inhibitor	1
atp-sensitive_potassium_channel_antagonist	1

206 rows × 1 columns

nfkb_inhibitor, proteasome_inhibitor, cyclooxygenase_inhibitor, dopamine_receptor_antagonist, serotonin_receptor_antagonist occurs most whereas elastase_inhibitor, steroid, atm_kinase_inhibitor, erbb2_inhibitor, atp-sensitive_potassium_channel_antagonist occurs least.