CASE Study-1 - Don't not Overfit ||

Data Source : https://www.kaggle.com/c/dont-overfit-ii/data (https://www.kaggle.com/c/dont-overfit-ii/data)

The primary goal of this case study to avoid overfitting problem of given small amount of training sample

We have 250 samples data and 300 features and 1 id and 1 class label(0/1) We have 19750 features and 300 features and 1 id,

So, with small amount of training data set, We must to do task carefully to avoid overfitting.

Import Necessary Libraries

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Read the training data with the help of pandas library

In [ ]:

```
train_data = pd.read_csv('train.csv')
train_data.head()
```

Out[ ]:

	id	target	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	0	1.0	-1.067	-1.114	-0.616	0.376	1.090	0.467	-0.422	0.460	-0.443	-0.338	0.416	-2.177	-0.326	0.340	1.174	-0.
1	1	0.0	-0.831	0.271	1.716	1.096	1.731	-0.197	1.904	-0.265	0.557	1.202	0.542	0.424	-1.572	-0.968	-1.483	0.5
2	2	0.0	0.099	1.390	-0.732	-1.065	0.005	-0.081	-1.450	0.317	-0.624	-0.017	-0.665	1.905	0.376	-1.373	1.587	1.4
3	3	1.0	-0.989	-0.916	-1.343	0.145	0.543	0.636	1.127	0.189	-0.118	-0.638	0.760	-0.360	-2.048	-0.996	-0.361	0.9
4	4	0.0	0.811	-1.509	0.522	-0.360	-0.220	-0.959	0.334	-0.566	-0.656	-0.499	-0.653	-0.058	-0.046	0.654	-0.697	-1.

5 rows × 302 columns

In [ ]:

```
train_data.columns
```

Out[ ]:

```
Index(['id', 'target', '0', '1', '2', '3', '4', '5', '6', '7',
       ...
       '290', '291', '292', '293', '294', '295', '296', '297', '298', '299'],
      dtype='object', length=302)
```

In [ ]:

```
train_data.shape
```

Out[ ]:

```
(250, 302)
```

In [ ]:

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Columns: 302 entries, id to 299
dtypes: float64(301), int64(1)
memory usage: 590.0 KB
```

In [ ]:

```
train_data.describe()
```

Out[ ]:

	id	target	0	1	2	3	4	5	6	
count	250.000000	250.000000	250.000000	250.000000	250.000000	250.000000	250.000000	250.000000	250.000000	250.000
mean	124.500000	0.268000	-0.098064	0.001208	0.090680	-0.122248	0.011500	-0.116624	0.006932	0.10098
std	72.312977	0.443806	0.996063	0.955117	0.968065	0.933001	0.945662	1.081705	1.014091	1.02804
min	0.000000	0.000000	-3.181000	-3.041000	-2.967000	-2.898000	-2.837000	-3.831000	-2.873000	-2.48900
25%	62.250000	0.000000	-0.756250	-0.624750	-0.515750	-0.695500	-0.678000	-0.758500	-0.646250	-0.58900
50%	124.500000	0.000000	-0.064500	-0.008000	0.067500	-0.090000	0.028000	-0.073500	-0.076500	0.10450
75%	186.750000	1.000000	0.647750	0.493250	0.716000	0.436250	0.625250	0.554250	0.676500	0.71700
max	249.000000	1.000000	2.347000	3.138000	2.609000	2.590000	2.413000	2.687000	2.793000	3.76600

8 rows × 302 columns

Observation

Train data has 300 independent features(0-299) and one dependent feature(target) along with unique id. All the values of feature are continuous and have same distribution from distribution table.

In [ ]:

```
print(train_data["target"].value_counts())
```

```
0.0    183
1.0     67
Name: target, dtype: int64
```

We found 183 count for target value 0 and 67 for target value 1. This is imbalanced dataset(Not highly imbalanced but decent).

In [ ]:

```
nul=0
for col in train_data.columns:
  if(train_data[col].isnull().any()):
    print(col,"has null value")
    nul=1
if(nul==0):
  print("There is no null value")
```

```
There is no null value
```

There are no null values found and all are real values

Exploratory Data Analysis(EDA)

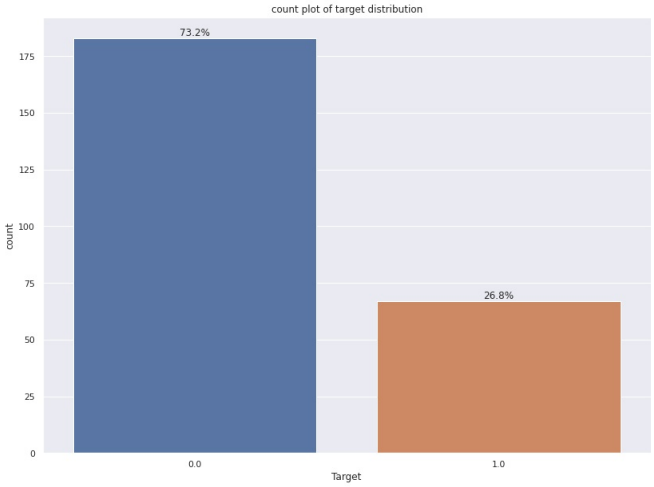
In [ ]:

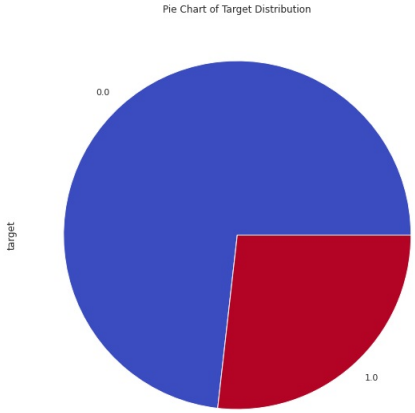
```
sns.set(rc={'figure.figsize':(30,10)})
plt.subplot(1,2,1)
ax = sns.countplot(train_data['target'])
for p in ax.patches:
  ax.annotate('{:.1f}%'.format(100*p.get_height()/len(train_data)), (p.get_x()+0.35, p.get_height()+1))
ax.set_title('count plot of target distribution')
ax.set_xlabel('Target')
ax.set_ylabel('count')

plt.subplot(1,2,2)
ax = train_data['target'].value_counts().plot(kind='pie',colormap='coolwarm')
ax.set_title("Pie Chart of Target Distribution")
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following
variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result in an error or misinterpretation
.
  FutureWarning
```

Out[ ]:

```
Text(0.5, 1.0, 'Pie Chart of Target Distribution')
```





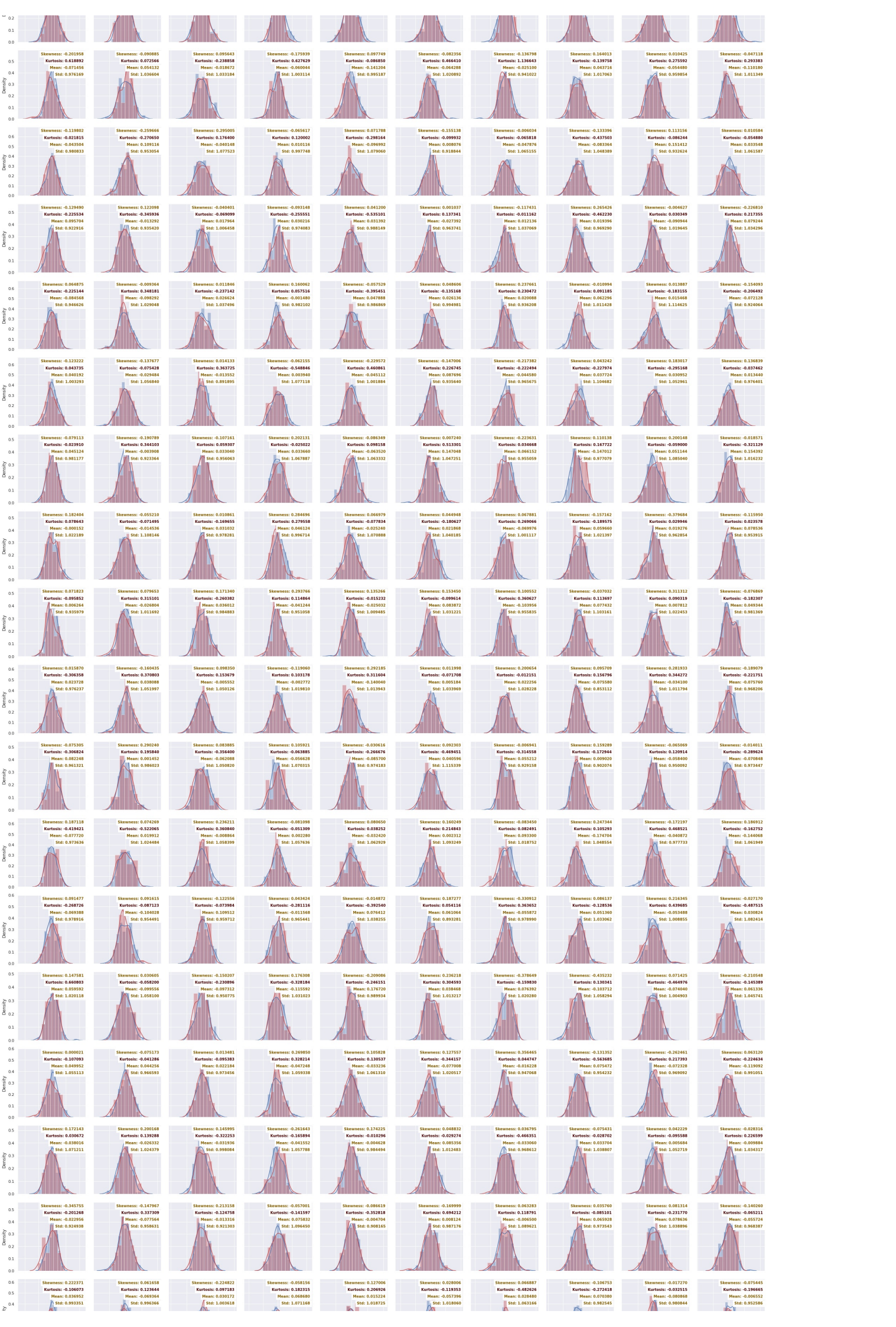
Observation

It is clearly said that data is imbalanced, where data points belonging to target 0 is 73.2% and 1 is 26.8%. Thus it is imbalanced data.

New Thing : I have tried to get mean and standandard deviation

In [ ]:

```
#https://www.analyticsvidhya.com/blog/2021/05/shape-of-data-skewness-and-kurtosis/

# https://stackoverflow.com/questions/50940283/show-metrics-like-kurtosis-skewness-on-distribution-plot-using-seaborn-in-pytho
import warnings
warnings.filterwarnings("ignore")
sns.set(rc={'figure.figsize':(30,90)})
fig, ax = plt.subplots(30, 10,sharex='col', sharey='row')
ax = ax.reshape(-1)
for i, col in enumerate(train_data.columns[2:]):
    g0 = train_data[train_data['target']==0.0][col]
    g1 = train_data[train_data['target']==1.0][col]
    sns.distplot(g0, label = 'target 0', ax=ax[i], color='b')
    sns.distplot(g1, label = 'target 1', ax=ax[i], color='r')

min_skew, max_skew = 100, -100
min_kurt, max_kurt = 100, -100

for i, x in enumerate(ax):
    skew = train_data.iloc[:,i+2].skew()
    min_skew = min(min_skew, skew)
    max_skew = max(max_skew, skew)
    x.text(x=0.97, y=0.97, transform=x.transAxes, s="Skewness: %f" % skew,\
        fontweight='demibold', fontsize=10, verticalalignment='top', horizontalalignment='right',\
        backgroundcolor='white', color='xkcd:poo brown')
    kurt = train_data.iloc[:,i+2].kurt()
    min_kurt = min(min_kurt, kurt)
    max_kurt = max(max_kurt, kurt)
    x.text(x=0.97, y=0.87, transform=x.transAxes, s="Kurtosis: %f" % kurt,\
        fontweight='demibold', fontsize=10, verticalalignment='top', horizontalalignment='right',\
        backgroundcolor='white', color='xkcd:dried blood')
for i, x in enumerate(ax):
    mean = train_data.iloc[:,i+2].mean()

    x.text(x=0.97, y=0.77, transform=x.transAxes, s="Mean: %f" % mean,\
        fontweight='demibold', fontsize=10, verticalalignment='top', horizontalalignment='right',\
        backgroundcolor='white', color='xkcd:poo brown')
for i, x in enumerate(ax):
    std = train_data.iloc[:,i+2].std()

    x.text(x=0.97, y=0.67, transform=x.transAxes, s="Std: %f" % std,\
        fontweight='demibold', fontsize=10, verticalalignment='top', horizontalalignment='right',\
        backgroundcolor='white', color='xkcd:poo brown')
plt.tight_layout()
plt.show()

print('Skewness Range:[{},{}] '.format(min_skew,max_skew))
print('Kurtosis Range:[{},{}] '.format(min_kurt,max_kurt))
```


Skewness: -0.201958
Kurtosis: 0.618892
Mean: -0.071456
Std: 0.976169
Skewness: -0.090885
Kurtosis: 0.072566
Mean: 0.054132
Std: 1.036604
Skewness: 0.095643
Kurtosis: -0.238858
Mean: -0.018672
Std: 1.033184
Skewness: -0.175939
Kurtosis: 0.627629
Mean: -0.060044
Std: 1.003114
Skewness: 0.097749
Kurtosis: -0.086850
Mean: -0.141204
Std: 0.995187
Skewness: -0.082356
Kurtosis: 0.466410
Mean: -0.064288
Std: 1.020892
Skewness: -0.136798
Kurtosis: 1.136643
Mean: -0.025100
Std: 0.941022
Skewness: 0.164013
Kurtosis: -0.139758
Mean: 0.043716
Std: 1.017063
Skewness: 0.010425
Kurtosis: 0.275592
Mean: -0.054480
Std: 0.959854
Skewness: -0.047118
Kurtosis: 0.293383
Mean: -0.110180
Std: 1.011349
Skewness: -0.119802
Kurtosis: -0.021815
Mean: -0.043504
Std: 0.980833
Skewness: -0.259666
Kurtosis: -0.270650
Mean: 0.109116
Std: 0.953054
Skewness: 0.295005
Kurtosis: 0.176400
Mean: -0.040148
Std: 1.077523
Skewness: -0.065617
Kurtosis: 0.120002
Mean: 0.010116
Std: 0.997748
Skewness: 0.071788
Kurtosis: -0.298164
Mean: -0.096992
Std: 1.079060
Skewness: -0.155138
Kurtosis: -0.099932
Mean: 0.008076
Std: 0.918844
Skewness: -0.006034
Kurtosis: -0.065818
Mean: -0.047876
Std: 1.065155
Skewness: -0.133396
Kurtosis: -0.437503
Mean: -0.083364
Std: 1.048389
Skewness: 0.113156
Kurtosis: -0.086244
Mean: 0.151412
Std: 0.932624
Skewness: 0.010584
Kurtosis: -0.054880
Mean: 0.033548
Std: 1.061587
Skewness: -0.129490
Kurtosis: -0.225534
Mean: 0.095704
Std: 0.922916
Skewness: 0.122098
Kurtosis: -0.345936
Mean: -0.013292
Std: 0.935420
Skewness: -0.040401
Kurtosis: -0.069099
Mean: 0.017964
Std: 1.006458
Skewness: -0.093148
Kurtosis: -0.255551
Mean: 0.030216
Std: 0.974083
Skewness: 0.041200
Kurtosis: -0.535101
Mean: 0.031392
Std: 0.988149
Skewness: 0.001037
Kurtosis: 0.137341
Mean: -0.027392
Std: 0.963741
Skewness: -0.117431
Kurtosis: -0.011162
Mean: 0.012136
Std: 1.037069
Skewness: 0.265426
Kurtosis: -0.462230
Mean: 0.019396
Std: 0.969290
Skewness: -0.004627
Kurtosis: 0.030349
Mean: -0.090944
Std: 1.019645
Skewness: -0.226810
Kurtosis: 0.217355
Mean: 0.079244
Std: 1.034296
Skewness: 0.064875
Kurtosis: -0.225144
Mean: -0.084568
Std: 0.946626
Skewness: -0.009364
Kurtosis: 0.348181
Mean: -0.098292
Std: 1.029048
Skewness: 0.011846
Kurtosis: -0.237142
Mean: 0.026624
Std: 1.037496
Skewness: 0.160062
Kurtosis: 0.057516
Mean: -0.001480
Std: 0.982102
Skewness: -0.057529
Kurtosis: -0.395451
Mean: 0.047888
Std: 0.986869
Skewness: 0.048606
Kurtosis: -0.135168
Mean: 0.026136
Std: 0.994981
Skewness: 0.237661
Kurtosis: 0.230472
Mean: 0.020088
Std: 0.936208
Skewness: -0.010994
Kurtosis: 0.091185
Mean: 0.062296
Std: 1.011428
Skewness: 0.013887
Kurtosis: -0.183155
Mean: 0.015468
Std: 1.114625
Skewness: -0.154093
Kurtosis: -0.206492
Mean: -0.072128
Std: 0.924064
Skewness: -0.123222
Kurtosis: 0.043735
Mean: 0.040192
Std: 1.003293
Skewness: -0.137677
Kurtosis: -0.075428
Mean: -0.029484
Std: 1.056840
Skewness: 0.014133
Kurtosis: 0.363725
Mean: -0.013552
Std: 0.891895
Skewness: -0.062155
Kurtosis: -0.548846
Mean: 0.003940
Std: 1.077118
Skewness: -0.229572
Kurtosis: 0.460861
Mean: -0.045112
Std: 1.001884
Skewness: -0.147006
Kurtosis: 0.226745
Mean: 0.087696
Std: 0.935640
Skewness: -0.217382
Kurtosis: 0.222494
Mean: -0.044580
Std: 0.965675
Skewness: 0.043242
Kurtosis: -0.227974
Mean: 0.037724
Std: 1.104682
Skewness: 0.183017
Kurtosis: -0.295168
Mean: 0.030952
Std: 1.052961
Skewness: 0.136839
Kurtosis: -0.037462
Mean: 0.013440
Std: 0.976401
Skewness: -0.079113
Kurtosis: -0.023910
Mean: 0.045124
Std: 0.981177
Skewness: -0.190789
Kurtosis: 0.344103
Mean: -0.003908
Std: 0.923364
Skewness: -0.107161
Kurtosis: 0.059307
Mean: 0.033040
Std: 0.956063
Skewness: 0.202131
Kurtosis: -0.025022
Mean: 0.033660
Std: 1.067887
Skewness: -0.086349
Kurtosis: 0.098158
Mean: -0.063520
Std: 1.063332
Skewness: 0.007240
Kurtosis: 0.513301
Mean: 0.147048
Std: 1.047251
Skewness: -0.223631
Kurtosis: 0.034668
Mean: 0.066152
Std: 0.955059
Skewness: 0.110138
Kurtosis: 0.167722
Mean: -0.147012
Std: 0.977079
Skewness: 0.200148
Kurtosis: -0.059000
Mean: 0.051144
Std: 1.085040
Skewness: -0.018571
Kurtosis: -0.321129
Mean: 0.154392
Std: 1.016232
Skewness: 0.182404
Kurtosis: 0.078643
Mean: -0.000152
Std: 1.022189
Skewness: -0.055210
Kurtosis: -0.071495
Mean: -0.014536
Std: 1.108146
Skewness: 0.010861
Kurtosis: -0.169655
Mean: 0.031032
Std: 0.978281
Skewness: 0.284696
Kurtosis: 0.279558
Mean: 0.046124
Std: 0.996714
Skewness: 0.066979
Kurtosis: -0.077834
Mean: -0.025240
Std: 1.070888
Skewness: 0.044948
Kurtosis: -0.180627
Mean: 0.021868
Std: 1.040185
Skewness: 0.067881
Kurtosis: 0.269066
Mean: -0.069976
Std: 1.001117
Skewness: -0.157162
Kurtosis: -0.189575
Mean: 0.059660
Std: 1.021397
Skewness: -0.379684
Kurtosis: 0.029946
Mean: 0.019276
Std: 0.962854
Skewness: -0.115950
Kurtosis: 0.023578
Mean: 0.078536
Std: 0.953915
Skewness: 0.071823
Kurtosis: -0.095852
Mean: 0.006264
Std: 0.935579
Skewness: 0.079653
Kurtosis: 0.315101
Mean: -0.026804
Std: 1.011692
Skewness: 0.171340
Kurtosis: -0.260382
Mean: 0.036012
Std: 0.984883
Skewness: 0.293766
Kurtosis: 0.114864
Mean: -0.041244
Std: 0.951058
Skewness: 0.135266
Kurtosis: -0.015232
Mean: -0.025032
Std: 1.009485
Skewness: 0.153450
Kurtosis: -0.099614
Mean: 0.083872
Std: 1.031221
Skewness: 0.100552
Kurtosis: 0.360627
Mean: -0.103956
Std: 0.955835
Skewness: -0.037032
Kurtosis: 0.113697
Mean: 0.077432
Std: 1.103161
Skewness: 0.311312
Kurtosis: 0.090319
Mean: 0.007812
Std: 1.022453
Skewness: -0.076869
Kurtosis: -0.182307
Mean: 0.049344
Std: 0.981369
Skewness: 0.015870
Kurtosis: -0.306358
Mean: 0.023728
Std: 0.976237
Skewness: -0.160435
Kurtosis: 0.370803
Mean: 0.038088
Std: 1.051997
Skewness: 0.098350
Kurtosis: 0.153679
Mean: -0.005552
Std: 1.050126
Skewness: -0.119060
Kurtosis: 0.103178
Mean: -0.002772
Std: 1.019810
Skewness: 0.292185
Kurtosis: 0.311604
Mean: -0.140040
Std: 1.013943
Skewness: 0.011998
Kurtosis: -0.071708
Mean: 0.005184
Std: 1.033969
Skewness: 0.200654
Kurtosis: -0.012151
Mean: 0.022256
Std: 1.028228
Skewness: 0.095709
Kurtosis: 0.156796
Mean: -0.075580
Std: 0.853112
Skewness: 0.281933
Kurtosis: 0.344272
Mean: -0.034100
Std: 1.011794
Skewness: -0.189079
Kurtosis: -0.221751
Mean: -0.075760
Std: 0.968206
Skewness: -0.075305
Kurtosis: -0.306824
Mean: 0.082248
Std: 0.961321
Skewness: 0.290240
Kurtosis: 0.195840
Mean: 0.001452
Std: 0.986023
Skewness: 0.083885
Kurtosis: -0.354000
Mean: -0.062088
Std: 1.050820
Skewness: 0.105921
Kurtosis: -0.063885
Mean: -0.056628
Std: 1.070315
Skewness: -0.030616
Kurtosis: 0.311604
Mean: -0.085700
Std: 0.974183
Skewness: 0.092303
Kurtosis: -0.469451
Mean: 0.040596
Std: 1.115339
Skewness: -0.006941
Kurtosis: -0.314558
Mean: 0.055212
Std: 0.929158
Skewness: 0.159289
Kurtosis: -0.172944
Mean: 0.009020
Std: 0.902074
Skewness: -0.065069
Kurtosis: 0.120914
Mean: -0.058400
Std: 0.950092
Skewness: -0.014011
Kurtosis: -0.289624
Mean: -0.070848
Std: 0.973447
Skewness: 0.187118
Kurtosis: -0.419421
Mean: -0.077720
Std: 0.973636
Skewness: 0.074269
Kurtosis: -0.522065
Mean: 0.019912
Std: 1.024484
Skewness: 0.236211
Kurtosis: 0.360840
Mean: -0.008864
Std: 1.058399
Skewness: -0.081098
Kurtosis: -0.051309
Mean: 0.002280
Std: 1.057636
Skewness: 0.080650
Kurtosis: 0.038252
Mean: -0.032420
Std: 1.062929
Skewness: 0.160249
Kurtosis: 0.214843
Mean: 0.002312
Std: 1.093249
Skewness: -0.083450
Kurtosis: 0.082491
Mean: 0.093300
Std: 1.018752
Skewness: 0.247344
Kurtosis: 0.105293
Mean: -0.174704
Std: 1.048554
Skewness: -0.172197
Kurtosis: 0.468521
Mean: -0.040872
Std: 0.977733
Skewness: 0.186912
Kurtosis: -0.162752
Mean: -0.144068
Std: 1.061949
Skewness: 0.091477
Kurtosis: -0.268726
Mean: -0.069388
Std: 0.978916
Skewness: 0.091615
Kurtosis: -0.087123
Mean: -0.104028
Std: 0.954491
Skewness: -0.122556
Kurtosis: -0.073984
Mean: 0.109512
Std: 0.959712
Skewness: 0.043424
Kurtosis: -0.281116
Mean: -0.011568
Std: 0.965441
Skewness: -0.014872
Kurtosis: -0.392540
Mean: 0.076412
Std: 1.038255
Skewness: 0.187277
Kurtosis: 0.054116
Mean: 0.061064
Std: 0.893281
Skewness: -0.330912
Kurtosis: 0.363652
Mean: -0.055872
Std: 0.978990
Skewness: 0.086137
Kurtosis: -0.128536
Mean: 0.051360
Std: 1.033062
Skewness: 0.216345
Kurtosis: 0.439685
Mean: -0.053488
Std: 1.008855
Skewness: -0.027170
Kurtosis: -0.487515
Mean: 0.030824
Std: 1.082414
Skewness: 0.147581
Kurtosis: 0.660803
Mean: 0.059592
Std: 1.020118
Skewness: 0.030605
Kurtosis: -0.058200
Mean: -0.099556
Std: 1.058100
Skewness: -0.150207
Kurtosis: -0.230896
Mean: -0.097312
Std: 0.950775
Skewness: 0.176308
Kurtosis: -0.328184
Mean: -0.115592
Std: 1.031023
Skewness: -0.209086
Kurtosis: -0.246151
Mean: 0.176720
Std: 0.989934
Skewness: 0.236218
Kurtosis: 0.304593
Mean: 0.038468
Std: 1.013217
Skewness: -0.378649
Kurtosis: -0.159830
Mean: 0.076392
Std: 1.020280
Skewness: -0.435232
Kurtosis: 0.130341
Mean: -0.103712
Std: 1.058294
Skewness: 0.071425
Kurtosis: -0.464976
Mean: -0.074040
Std: 1.004903
Skewness: -0.210548
Kurtosis: -0.145389
Mean: 0.061336
Std: 1.045741
Skewness: 0.000021
Kurtosis: -0.107093
Mean: 0.048952
Std: 1.055113
Skewness: -0.075173
Kurtosis: -0.041286
Mean: 0.044256
Std: 0.966593
Skewness: 0.013481
Kurtosis: -0.095383
Mean: 0.022184
Std: 0.973456
Skewness: 0.269850
Kurtosis: 0.328214
Mean: -0.047248
Std: 1.059338
Skewness: 0.105828
Kurtosis: 0.130537
Mean: -0.033236
Std: 1.061310
Skewness: 0.127557
Kurtosis: -0.344157
Mean: -0.077008
Std: 1.020517
Skewness: 0.356465
Kurtosis: 0.044747
Mean: -0.016228
Std: 0.947068
Skewness: -0.131352
Kurtosis: -0.563685
Mean: 0.075472
Std: 0.954232
Skewness: -0.262461
Kurtosis: 0.217393
Mean: -0.072328
Std: 0.969092
Skewness: 0.063120
Kurtosis: -0.224634
Mean: -0.119092
Std: 0.991051
Skewness: 0.172143
Kurtosis: 0.030672
Mean: -0.038016
Std: 1.071211
Skewness: 0.200168
Kurtosis: 0.139288
Mean: -0.026332
Std: 1.024379
Skewness: 0.145995
Kurtosis: -0.322253
Mean: -0.031936
Std: 0.998084
Skewness: -0.261643
Kurtosis: -0.165894
Mean: -0.041552
Std: 1.057788
Skewness: 0.174225
Kurtosis: -0.010296
Mean: -0.004628
Std: 0.984494
Skewness: 0.048832
Kurtosis: -0.029274
Mean: 0.085356
Std: 1.012483
Skewness: 0.036795
Kurtosis: -0.466351
Mean: -0.033060
Std: 0.968612
Skewness: -0.075431
Kurtosis: -0.028702
Mean: 0.033704
Std: 1.038807
Skewness: 0.042229
Kurtosis: -0.095588
Mean: 0.005684
Std: 1.052719
Skewness: -0.028316
Kurtosis: 0.226599
Mean: -0.009884
Std: 1.034317
Skewness: -0.345755
Kurtosis: -0.201268
Mean: -0.022956
Std: 0.924938
Skewness: -0.147967
Kurtosis: 0.337309
Mean: -0.077564
Std: 0.958631
Skewness: 0.213158
Kurtosis: -0.124758
Mean: -0.013316
Std: 0.921303
Skewness: -0.057001
Kurtosis: -0.141597
Mean: 0.075832
Std: 1.096450
Skewness: -0.086619
Kurtosis: -0.352818
Mean: -0.004704
Std: 0.908165
Skewness: -0.169999
Kurtosis: 0.694212
Mean: 0.008124
Std: 0.987176
Skewness: 0.063283
Kurtosis: 0.118791
Mean: -0.006500
Std: 1.089621
Skewness: 0.035760
Kurtosis: -0.085101
Mean: 0.065928
Std: 0.973543
Skewness: 0.081314
Kurtosis: -0.231770
Mean: 0.078636
Std: 1.038896
Skewness: -0.140260
Kurtosis: -0.065211
Mean: -0.055724
Std: 0.968387
Skewness: 0.222371
Kurtosis: -0.106073
Mean: 0.036952
Std: 0.993351
Skewness: 0.061658
Kurtosis: 0.123644
Mean: -0.069364
Std: 0.996366
Skewness: -0.224822
Kurtosis: 0.097183
Mean: 0.030172
Std: 1.003618
Skewness: -0.058156
Kurtosis: 0.182315
Mean: 0.068680
Std: 1.071168
Skewness: 0.127006
Kurtosis: 0.206926
Mean: 0.015224
Std: 1.018725
Skewness: 0.028006
Kurtosis: -0.119353
Mean: -0.057396
Std: 1.018060
Skewness: 0.066887
Kurtosis: -0.482626
Mean: 0.028480
Std: 1.063166
Skewness: -0.106753
Kurtosis: -0.272418
Mean: 0.070380
Std: 0.982545
Skewness: -0.017270
Kurtosis: -0.032515
Mean: -0.080868
Std: 0.980844
Skewness: -0.075445
Kurtosis: -0.196665
Mean: -0.006552
Std: 0.952586
Density

```
Skewness Range:[-0.4889114906320757,0.4864285153563513]
Kurtosis Range:[-0.5636848359050028,1.5206991488929726]
```

In [ ]:

```
print('Skewness Range:[{},{}] '.format(min_skew,max_skew))
print('Kurtosis Range:[{},{}] '.format(min_kurt,max_kurt))
```

```
Skewness Range:[-0.4889114906320757,0.4864285153563513]
Kurtosis Range:[-0.5636848359050028,1.5206991488929726]
```

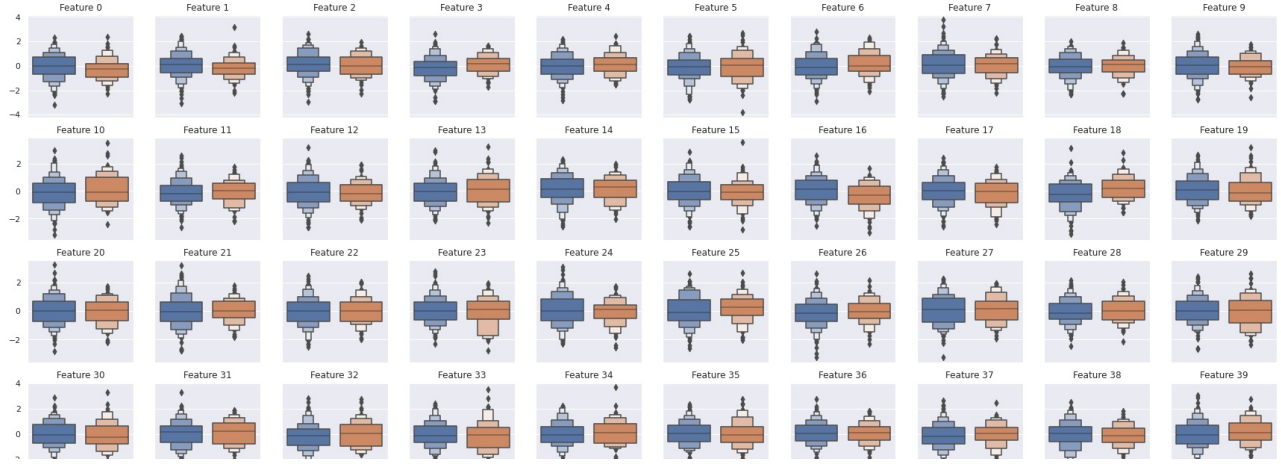
Based on above distriution,We have found skewness range between -0.5 to 0.5 and kurtosis range is less than 3. The distribution for all 300 features are showing alomost Symetric distribution.

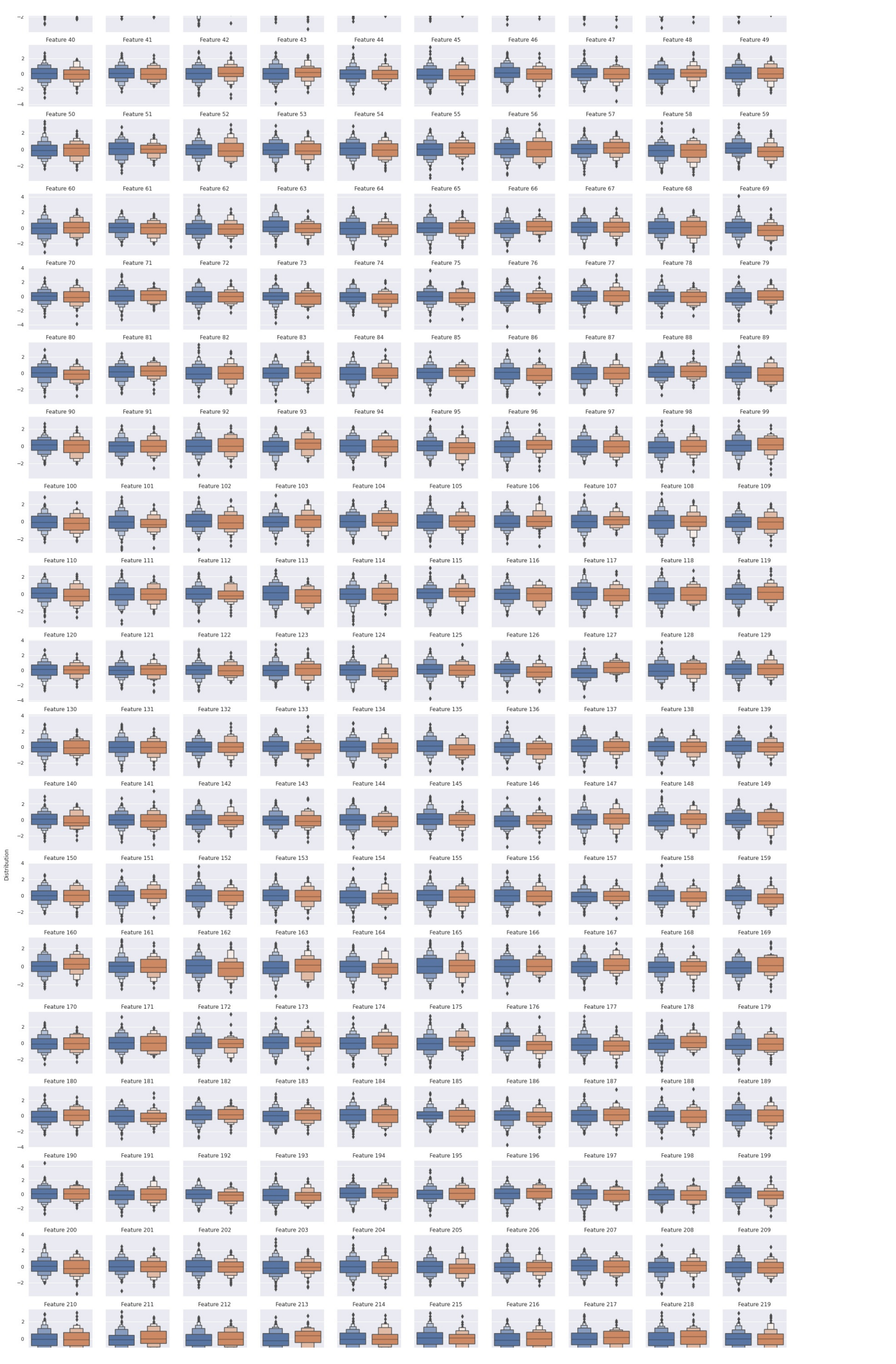
The mean for all 300 features are almost close to 0. And The mean for all 300 features are almost close to 1. The Distributions are very close to Standard Gaussian Distribution

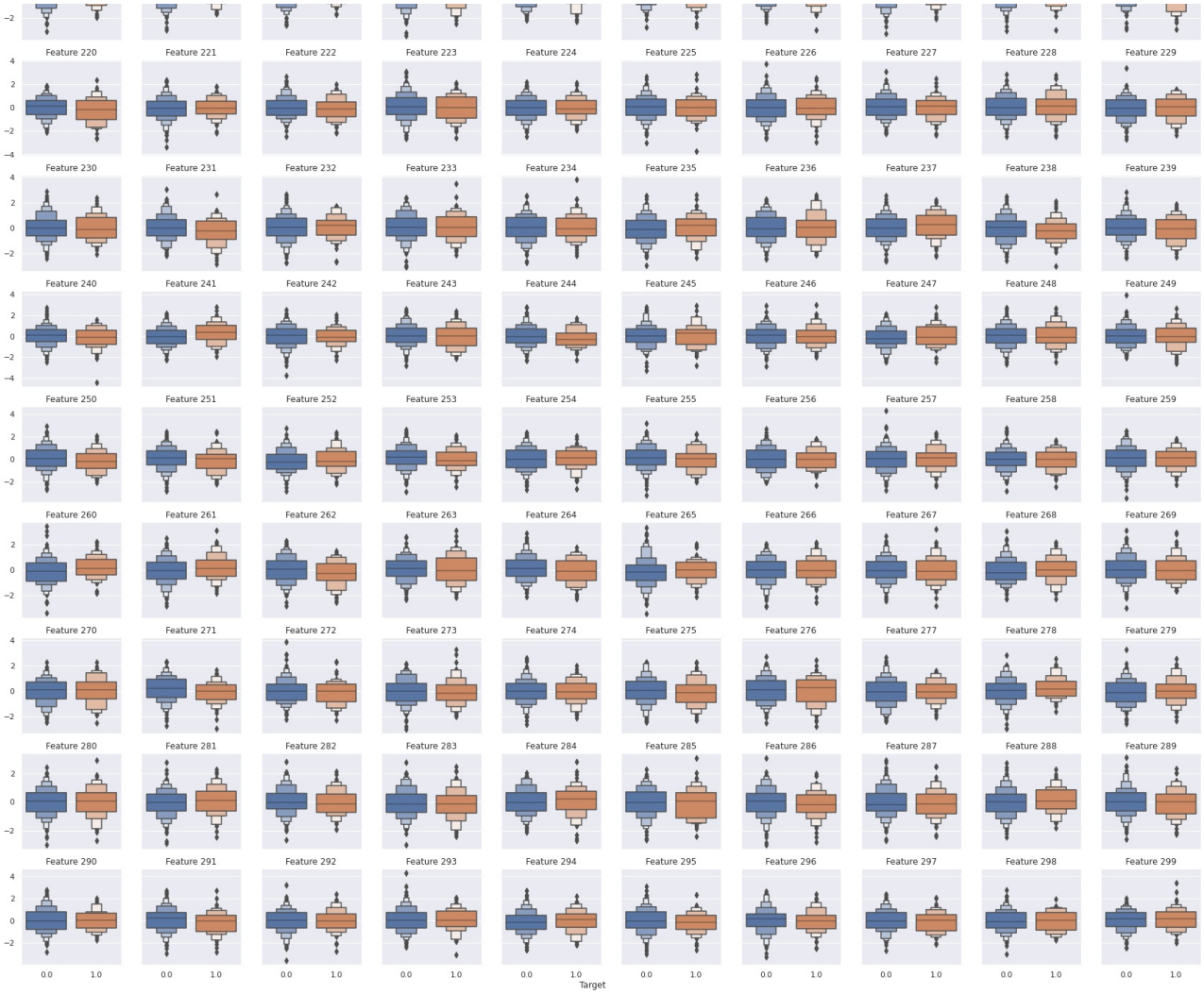
Box Plot of all 300 features in single frame

In [ ]:

```
sns.set(rc={'figure.figsize':(30,90)})
fig, ax = plt.subplots(30, 10,sharex='col', sharey='row')
fig.text(0.5, 0.12, s='Target', ha='center',)
fig.text(0.1, 0.5, s='Distribution', va='center', rotation='vertical')
ax = ax.reshape(-1)
for i, col in enumerate(train_data.columns[2:]):
    axe= sns.boxenplot(np.array(train_data['target']),np.array(train_data[col]), ax=ax[i])
    ax[i].set_title('Feature '+col)
plt.show()
```




Feature 40
Feature 41
Feature 42
Feature 43
Feature 44
Feature 45
Feature 46
Feature 47
Feature 48
Feature 49
Feature 50
Feature 51
Feature 52
Feature 53
Feature 54
Feature 55
Feature 56
Feature 57
Feature 58
Feature 59
Feature 60
Feature 61
Feature 62
Feature 63
Feature 64
Feature 65
Feature 66
Feature 67
Feature 68
Feature 69
Feature 70
Feature 71
Feature 72
Feature 73
Feature 74
Feature 75
Feature 76
Feature 77
Feature 78
Feature 79
Feature 80
Feature 81
Feature 82
Feature 83
Feature 84
Feature 85
Feature 86
Feature 87
Feature 88
Feature 89
Feature 90
Feature 91
Feature 92
Feature 93
Feature 94
Feature 95
Feature 96
Feature 97
Feature 98
Feature 99
Feature 100
Feature 101
Feature 102
Feature 103
Feature 104
Feature 105
Feature 106
Feature 107
Feature 108
Feature 109
Feature 110
Feature 111
Feature 112
Feature 113
Feature 114
Feature 115
Feature 116
Feature 117
Feature 118
Feature 119
Feature 120
Feature 121
Feature 122
Feature 123
Feature 124
Feature 125
Feature 126
Feature 127
Feature 128
Feature 129
Feature 130
Feature 131
Feature 132
Feature 133
Feature 134
Feature 135
Feature 136
Feature 137
Feature 138
Feature 139
Feature 140
Feature 141
Feature 142
Feature 143
Feature 144
Feature 145
Feature 146
Feature 147
Feature 148
Feature 149
Feature 150
Feature 151
Feature 152
Feature 153
Feature 154
Feature 155
Feature 156
Feature 157
Feature 158
Feature 159
Feature 160
Feature 161
Feature 162
Feature 163
Feature 164
Feature 165
Feature 166
Feature 167
Feature 168
Feature 169
Feature 170
Feature 171
Feature 172
Feature 173
Feature 174
Feature 175
Feature 176
Feature 177
Feature 178
Feature 179
Feature 180
Feature 181
Feature 182
Feature 183
Feature 184
Feature 185
Feature 186
Feature 187
Feature 188
Feature 189
Feature 190
Feature 191
Feature 192
Feature 193
Feature 194
Feature 195
Feature 196
Feature 197
Feature 198
Feature 199
Feature 200
Feature 201
Feature 202
Feature 203
Feature 204
Feature 205
Feature 206
Feature 207
Feature 208
Feature 209
Feature 210
Feature 211
Feature 212
Feature 213
Feature 214
Feature 215
Feature 216
Feature 217
Feature 218
Feature 219
Distribution



Observation

From the above two plot it is clearly mentioned that distribution of all 300 features are almost identical. All the features have almost same means and standard deviations

Seem like , Median of both target 0 and 1 have somewhat same. Boxplot detecting few points as outlier

In [ ]:

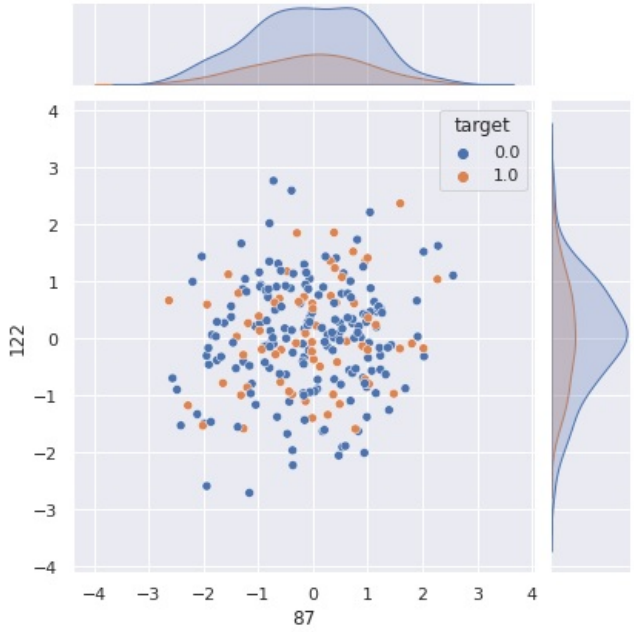
```
###Scatter plot
r = np.random.randint(0,299,2,dtype=int)

print('*********  scatter plot between {} and {} *********'.format(r[0],r[1]))

fig = plt.figure(figsize=(7,7))

sns.jointplot(data=train_data,x = str(r[0]),y = str(r[1]),hue=train_data['target'])
```

```
*********  scatter plot between 87 and 122 *********
```

Out[ ]:

```
<seaborn.axisgrid.JointGrid at 0x7f8f65615e50>
```

```
<Figure size 504x504 with 0 Axes>
```



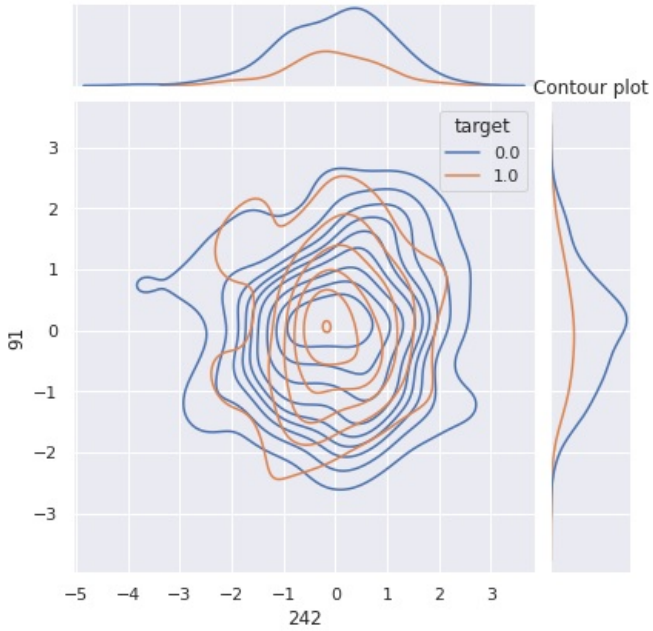
We are unable to interpret using scatter plot

Let's observe the density of region rather than points. One of the seaborn toolkit called Contour

In [ ]:

```
print('*********  Contour plot between {} and {} *********'.format(r[0],r[1]))

fig = plt.figure(figsize=(7,7))
r = np.random.randint(0,299,2,dtype=int)
sns.jointplot(x = str(r[0]),y = str(r[1]),data = train_data, hue=train_data['target'],kind = "kde")
plt.title("Contour plot")
plt.show()
```

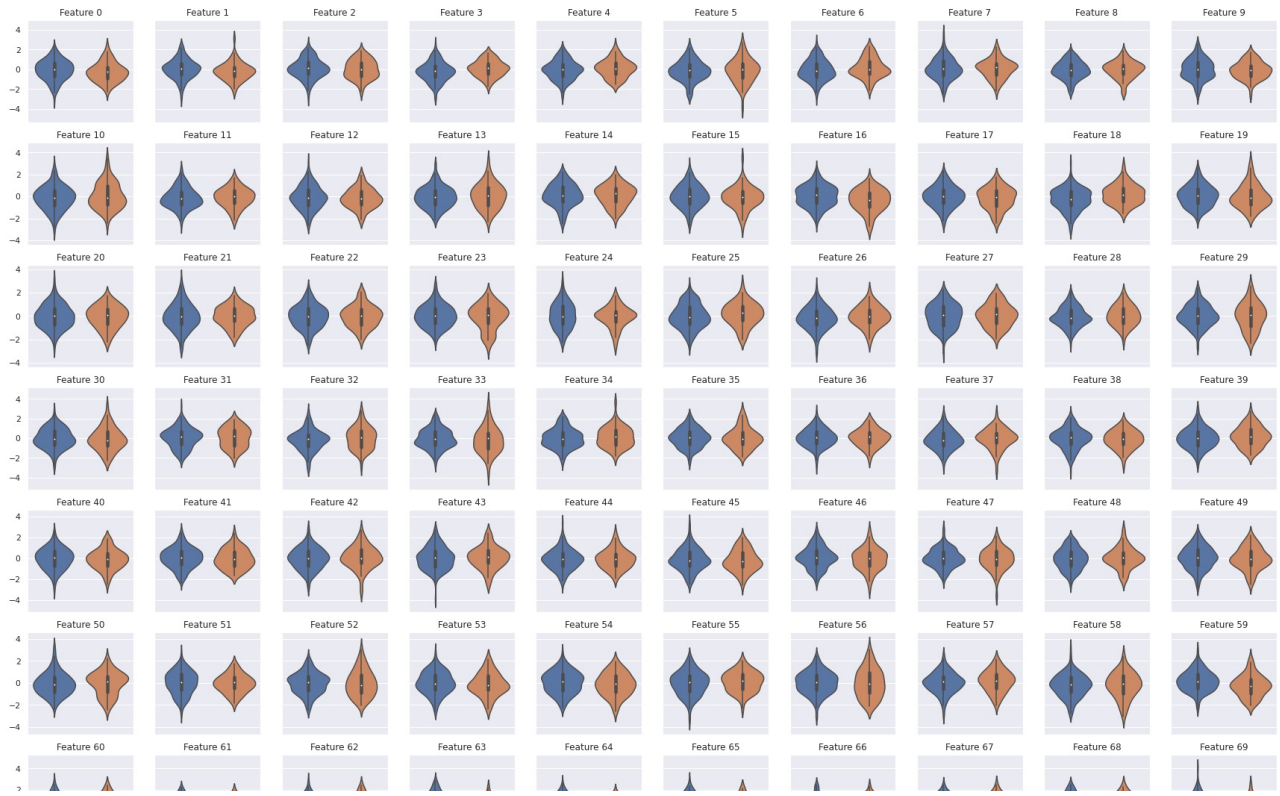
```
*********  Contour plot between 87 and 122 *********

<Figure size 504x504 with 0 Axes>
```

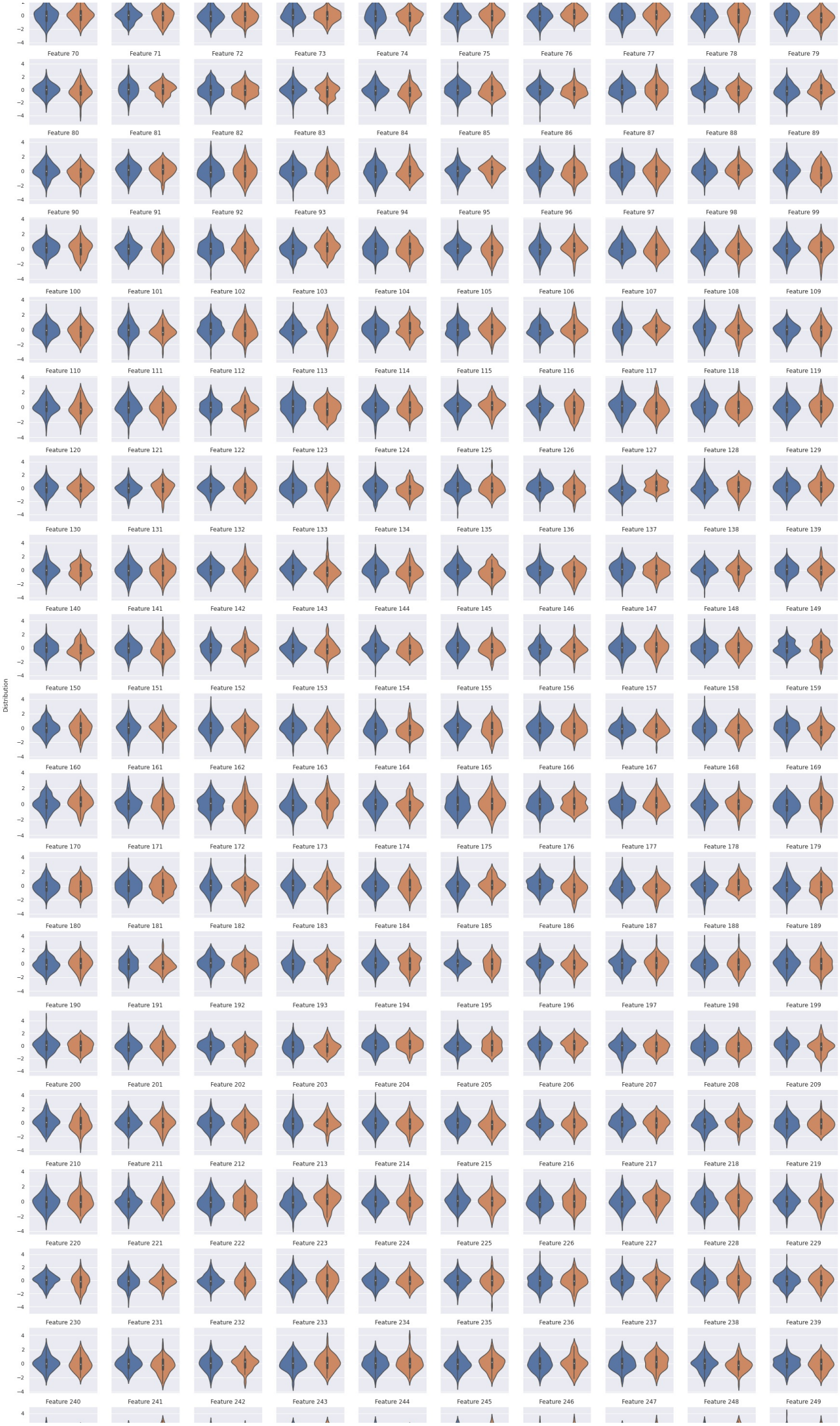


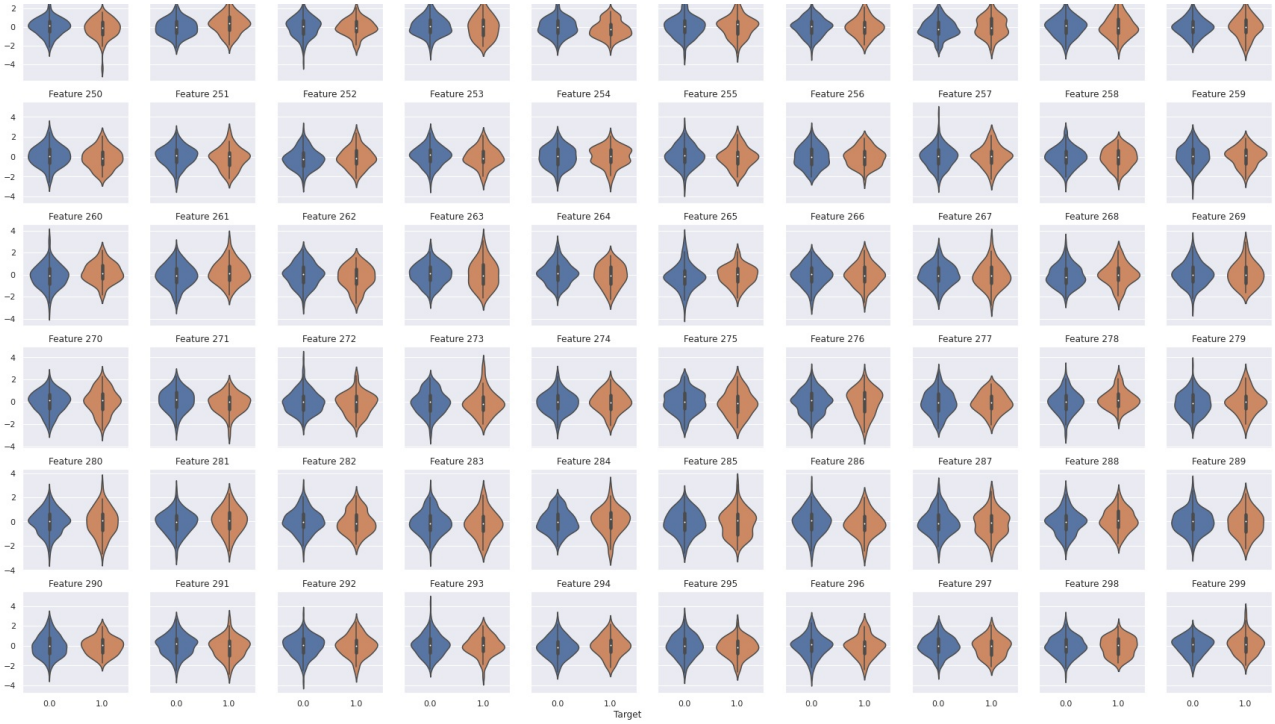
More density region is in range between -1 to 1.

In [ ]:

```
sns.set(rc={'figure.figsize':(30,90)})
fig, ax = plt.subplots(30, 10,sharex='col', sharey='row')
fig.text(0.5, 0.12, s='Target', ha='center',)
fig.text(0.1, 0.5, s='Distribution', va='center', rotation='vertical')
ax = ax.reshape(-1)
for i, col in enumerate(train_data.columns[2:]):
    axe= sns.violinplot(np.array(train_data['target']),np.array(train_data[col]), ax=ax[i])
    ax[i].set_title('Feature '+col)
plt.show()
```




Distribution
Feature 70
Feature 71
Feature 72
Feature 73
Feature 74
Feature 75
Feature 76
Feature 77
Feature 78
Feature 79
Feature 80
Feature 81
Feature 82
Feature 83
Feature 84
Feature 85
Feature 86
Feature 87
Feature 88
Feature 89
Feature 90
Feature 91
Feature 92
Feature 93
Feature 94
Feature 95
Feature 96
Feature 97
Feature 98
Feature 99
Feature 100
Feature 101
Feature 102
Feature 103
Feature 104
Feature 105
Feature 106
Feature 107
Feature 108
Feature 109
Feature 110
Feature 111
Feature 112
Feature 113
Feature 114
Feature 115
Feature 116
Feature 117
Feature 118
Feature 119
Feature 120
Feature 121
Feature 122
Feature 123
Feature 124
Feature 125
Feature 126
Feature 127
Feature 128
Feature 129
Feature 130
Feature 131
Feature 132
Feature 133
Feature 134
Feature 135
Feature 136
Feature 137
Feature 138
Feature 139
Feature 140
Feature 141
Feature 142
Feature 143
Feature 144
Feature 145
Feature 146
Feature 147
Feature 148
Feature 149
Feature 150
Feature 151
Feature 152
Feature 153
Feature 154
Feature 155
Feature 156
Feature 157
Feature 158
Feature 159
Feature 160
Feature 161
Feature 162
Feature 163
Feature 164
Feature 165
Feature 166
Feature 167
Feature 168
Feature 169
Feature 170
Feature 171
Feature 172
Feature 173
Feature 174
Feature 175
Feature 176
Feature 177
Feature 178
Feature 179
Feature 180
Feature 181
Feature 182
Feature 183
Feature 184
Feature 185
Feature 186
Feature 187
Feature 188
Feature 189
Feature 190
Feature 191
Feature 192
Feature 193
Feature 194
Feature 195
Feature 196
Feature 197
Feature 198
Feature 199
Feature 200
Feature 201
Feature 202
Feature 203
Feature 204
Feature 205
Feature 206
Feature 207
Feature 208
Feature 209
Feature 210
Feature 211
Feature 212
Feature 213
Feature 214
Feature 215
Feature 216
Feature 217
Feature 218
Feature 219
Feature 220
Feature 221
Feature 222
Feature 223
Feature 224
Feature 225
Feature 226
Feature 227
Feature 228
Feature 229
Feature 230
Feature 231
Feature 232
Feature 233
Feature 234
Feature 235
Feature 236
Feature 237
Feature 238
Feature 239
Feature 240
Feature 241
Feature 242
Feature 243
Feature 244
Feature 245
Feature 246
Feature 247
Feature 248
Feature 249



It seems like Median for both target value 0 and 1 have somewhat same.

New thing : I have tried Correlation Among Features till Feature Engineering

In [ ]:

```
train_data.drop(['id'],axis=1).corr()
```

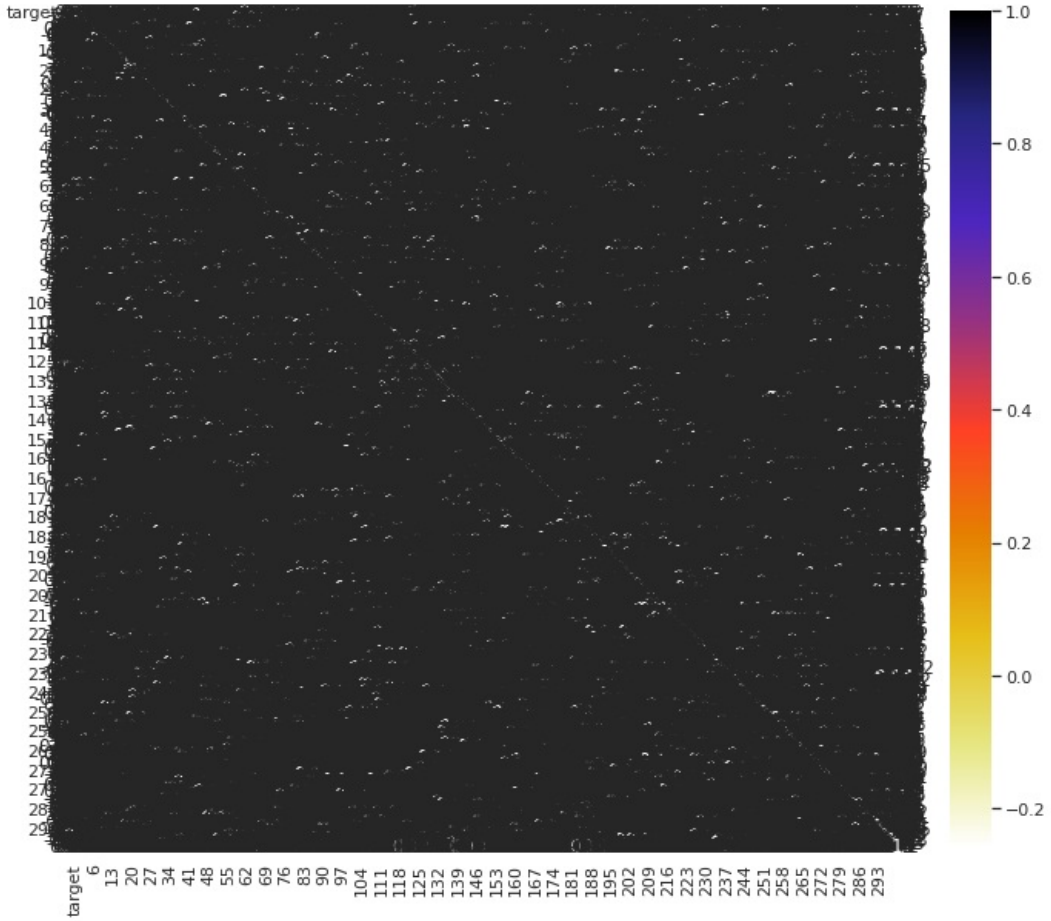
Out[ ]:

	target	0	1	2	3	4	5	6	7	8	9
target	1.000000	-0.085268	-0.111172	-0.036050	0.153317	0.077830	0.006206	0.090943	-0.003443	0.020330	-0.037787
0	-0.085268	1.000000	0.039939	0.069846	-0.139829	0.079360	-0.063259	-0.071990	0.016923	0.043081	0.051027
1	-0.111172	0.039939	1.000000	0.023237	-0.096873	-0.052313	0.048742	-0.086880	-0.013418	-0.005268	0.007490
2	-0.036050	0.069846	0.023237	1.000000	-0.107534	0.077051	-0.044687	-0.086411	0.000560	0.028217	-0.115959
3	0.153317	-0.139829	-0.096873	-0.107534	1.000000	-0.068026	-0.009757	0.049916	-0.063416	0.069039	0.047272
...	...	...	...	...	...	...	...	...	...	...	...
295	-0.071524	0.041362	-0.060408	0.151670	0.019118	-0.014687	0.070082	-0.069957	0.048203	0.113801	0.024786
296	-0.020375	0.009961	-0.192962	-0.030104	-0.049607	-0.002116	0.016213	-0.004527	-0.087313	-0.042409	-0.041123
297	-0.065313	-0.002739	0.025713	-0.025195	-0.008971	-0.031669	0.074662	0.007120	-0.047059	0.013366	-0.080101
298	-0.012973	0.092893	-0.049503	-0.086761	-0.052998	-0.044680	0.032147	-0.020821	-0.061363	0.019664	-0.045190
299	0.050805	0.065807	0.010546	-0.169051	0.076426	0.054352	0.064655	0.037515	-0.034875	0.060992	0.082712

301 rows × 301 columns

In [ ]:

```
#using pearson correlation
plt.figure(figsize=(12,10))
cor = train_data.drop(['id'],axis=1).corr()
sns.heatmap(cor,annot=True,cmap = plt.cm.CMRmap_r)
plt.show()
```



From above heatmap, it is really difficult to get exact correlation value between the features

Top 20 correlated features

In [ ]:

```
correlations = train_data.drop(['id'],axis=1).corr().drop_duplicates()
print("Top 20 positive correlated features with target")
print(correlations['target'].sort_values(ascending = False)[:20])

correlations = train_data.drop(['id'],axis=1).corr().drop_duplicates()
print("Top 20 Negative correlated features with target")
print(correlations['target'].sort_values(ascending = True)[:20])
```

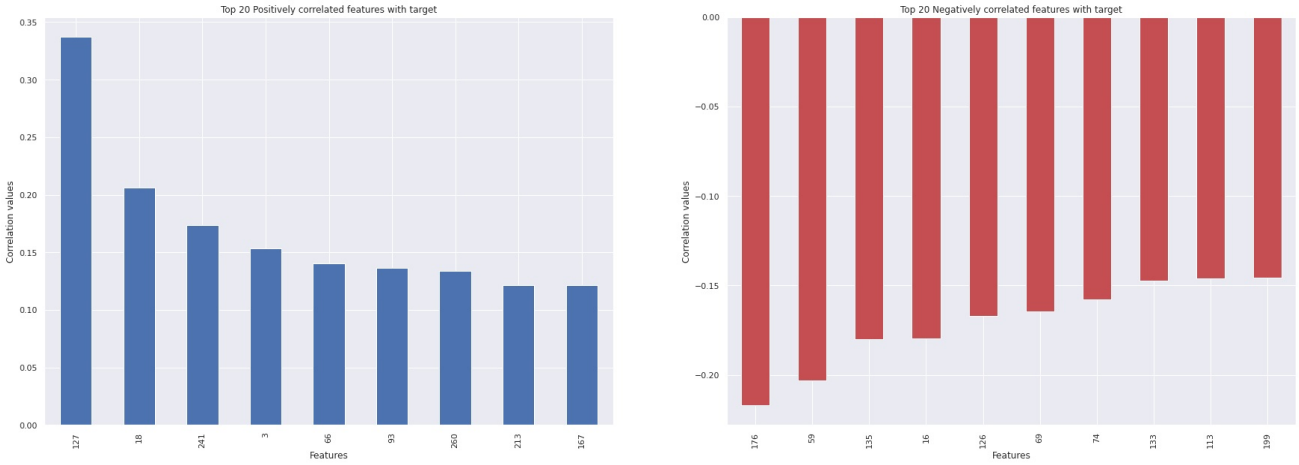
```
Top 20 positive correlated features with target
target    1.000000
127       0.337540
18        0.206452
241       0.173879
3         0.153317
66        0.140056
93        0.136455
260       0.134119
213       0.121495
167       0.121217
175       0.120121
261       0.118305
278       0.118187
211       0.116670
151       0.110165
178       0.108901
169       0.106084
208       0.103583
196       0.101347
183       0.098502
Name: target, dtype: float64
Top 20 Negative correlated features with target
176   -0.217100
59    -0.203166
135   -0.179960
16    -0.179796
126   -0.167064
69    -0.164571
74    -0.157756
133   -0.147518
113   -0.146098
199   -0.145666
136   -0.140761
231   -0.137473
271   -0.132168
159   -0.128942
63    -0.128926
177   -0.123470
262   -0.119571
220   -0.117128
95    -0.116410
192   -0.114118
Name: target, dtype: float64
```

In [ ]:

```
correlations = train_data.drop(['id'], axis=1).corr().unstack().drop_duplicates()
sns.set(rc={'figure.figsize':(30,10)})
plt.subplot(1,2,1)
x = correlations["target"].sort_values(ascending=False)[1:10].plot(kind='bar', title='Top 20 Positively correlate
d features with target')
x.set_xlabel('Features')
x.set_ylabel('Correlation values')

plt.subplot(1,2,2)
x = correlations["target"].sort_values(ascending=True)[:10].plot(kind='bar', title='Top 20 Negatively correlated
features with target', color='r')
x.set_xlabel('Features')
x.set_ylabel('Correlation values')
```

Out[ ]:

```
Text(0, 0.5, 'Correlation values')
```

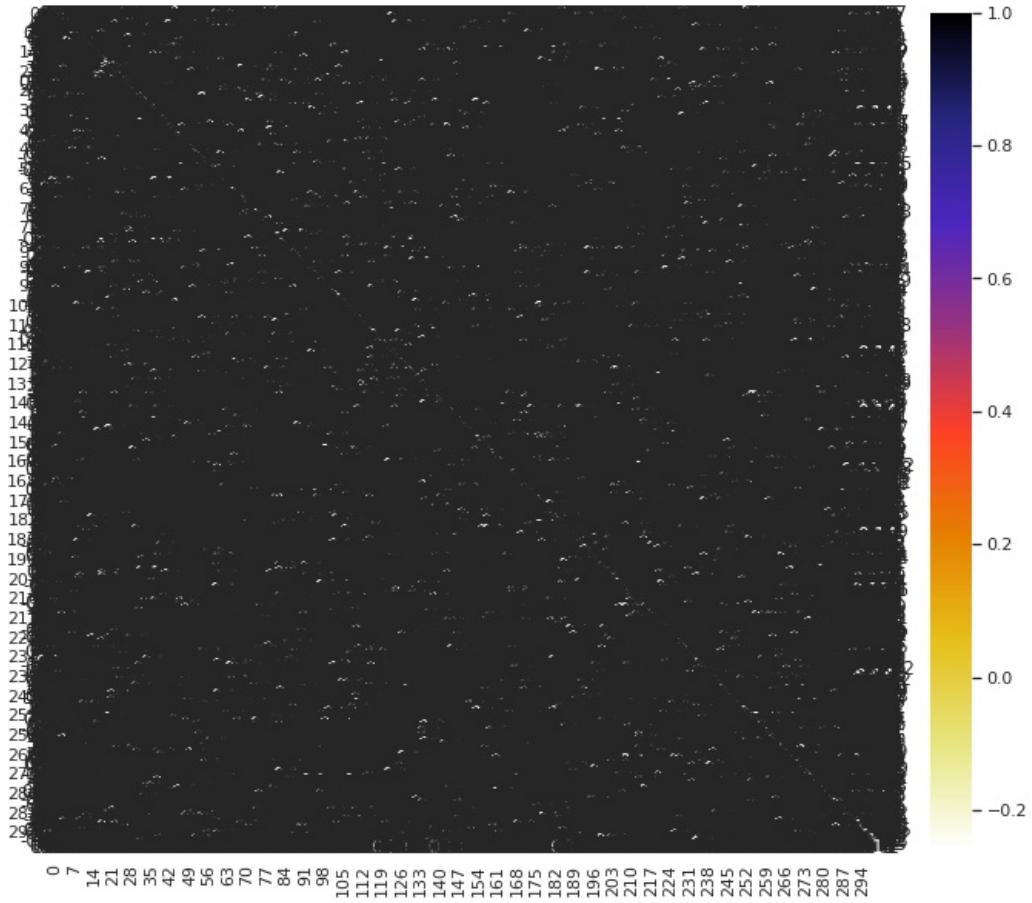


The Blue color is showing positive feature and red color negative features. Based on correlation values , we are see that features are not highly correlated with target. Feature 127 and 176 most correlated with target.

Correlation among 300 features

In [ ]:

```
plt.figure(figsize=(12,10))
cor = train_data.drop(['id','target'],axis=1).corr()
sns.heatmap(cor,annot=True,cmap = plt.cm.CMRmap_r)
plt.show()
```



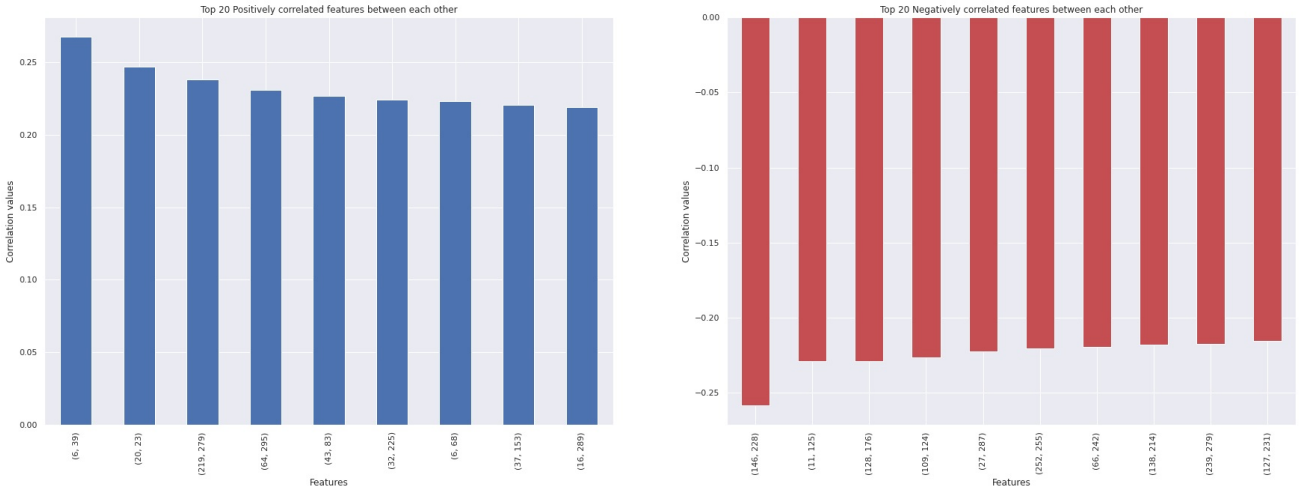
From above heatmap, it is really difficult to get exact correlation value between the features

In [ ]:

```
correlations = train_data.drop(['id','target'], axis=1).corr().unstack().drop_duplicates()
sns.set(rc={'figure.figsize':(30,10)})
plt.subplot(1,2,1)
x = correlations.sort_values(ascending=False)[1:10].plot(kind='bar', title='Top 20 Positively correlated features
between each other')
x.set_xlabel('Features')
x.set_ylabel('Correlation values')

plt.subplot(1,2,2)
x = correlations.sort_values(ascending=True)[:10].plot(kind='bar', title='Top 20 Negatively correlated features b
etween each other', color='r')
x.set_xlabel('Features')
x.set_ylabel('Correlation values')
```

Out[ ]:

```
Text(0, 0.5, 'Correlation values')
```



In [ ]:

```
def correlation(dataset,threshold):
  col_corr = set()#Set of all name of correlated columns
  corr_matrix = dataset.corr()
  for i in range(len(corr_matrix.columns)):
    for j in range(i):
      if abs(corr_matrix.iloc[i,j])>threshold:
         # we are interested in absolute coeff value
         colname = corr_matrix.columns[i] #getting the name of column
         col_corr.add(colname)
  return col_corr
```

In [ ]:

```
corr_features = correlation(train_data,0.5)
len(set(corr_features))
```

Out[ ]:

```
0
```

We have set threshold value for correlation value between feature is 0.5 but features are not higly correlated to each other. So we are not deleting any features here.

Feature Engineering

In [ ]:

```
train_data = pd.read_csv('train.csv')
train_data.head()
```

Out[ ]:

	id	target	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	1
0	0	1.0	-1.067	-1.114	-0.616	0.376	1.090	0.467	-0.422	0.460	-0.443	-0.338	0.416	-2.177	-0.326	0.340	1.174	-0.24
1	1	0.0	-0.831	0.271	1.716	1.096	1.731	-0.197	1.904	-0.265	0.557	1.202	0.542	0.424	-1.572	-0.968	-1.483	0.564
2	2	0.0	0.099	1.390	-0.732	-1.065	0.005	-0.081	-1.450	0.317	-0.624	-0.017	-0.665	1.905	0.376	-1.373	1.587	1.464
3	3	1.0	-0.989	-0.916	-1.343	0.145	0.543	0.636	1.127	0.189	-0.118	-0.638	0.760	-0.360	-2.048	-0.996	-0.361	0.962
4	4	0.0	0.811	-1.509	0.522	-0.360	-0.220	-0.959	0.334	-0.566	-0.656	-0.499	-0.653	-0.058	-0.046	0.654	-0.697	-1.17

5 rows × 302 columns

In [ ]:

```
#create duplicate
data = train_data.drop(['id','target'],axis=1)
data.head()
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	-1.067	-1.114	-0.616	0.376	1.090	0.467	-0.422	0.460	-0.443	-0.338	0.416	-2.177	-0.326	0.340	1.174	-0.245	-1.070
1	-0.831	0.271	1.716	1.096	1.731	-0.197	1.904	-0.265	0.557	1.202	0.542	0.424	-1.572	-0.968	-1.483	0.564	0.047
2	0.099	1.390	-0.732	-1.065	0.005	-0.081	-1.450	0.317	-0.624	-0.017	-0.665	1.905	0.376	-1.373	1.587	1.464	-1.550
3	-0.989	-0.916	-1.343	0.145	0.543	0.636	1.127	0.189	-0.118	-0.638	0.760	-0.360	-2.048	-0.996	-0.361	0.962	0.021
4	0.811	-1.509	0.522	-0.360	-0.220	-0.959	0.334	-0.566	-0.656	-0.499	-0.653	-0.058	-0.046	0.654	-0.697	-1.175	0.720

5 rows × 300 columns

Mean and Standard deviation value of each row

In [ ]:

```
train_data['mean'] = np.mean(data,axis=1)
train_data['std'] = np.std(data,axis=1)
train_data.head()
```

Out[ ]:

	id	target	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	0	1.0	-1.067	-1.114	-0.616	0.376	1.090	0.467	-0.422	0.460	-0.443	-0.338	0.416	-2.177	-0.326	0.340	1.174	-0.
1	1	0.0	-0.831	0.271	1.716	1.096	1.731	-0.197	1.904	-0.265	0.557	1.202	0.542	0.424	-1.572	-0.968	-1.483	0.5
2	2	0.0	0.099	1.390	-0.732	-1.065	0.005	-0.081	-1.450	0.317	-0.624	-0.017	-0.665	1.905	0.376	-1.373	1.587	1.4
3	3	1.0	-0.989	-0.916	-1.343	0.145	0.543	0.636	1.127	0.189	-0.118	-0.638	0.760	-0.360	-2.048	-0.996	-0.361	0.9
4	4	0.0	0.811	-1.509	0.522	-0.360	-0.220	-0.959	0.334	-0.566	-0.656	-0.499	-0.653	-0.058	-0.046	0.654	-0.697	-1.

5 rows × 304 columns

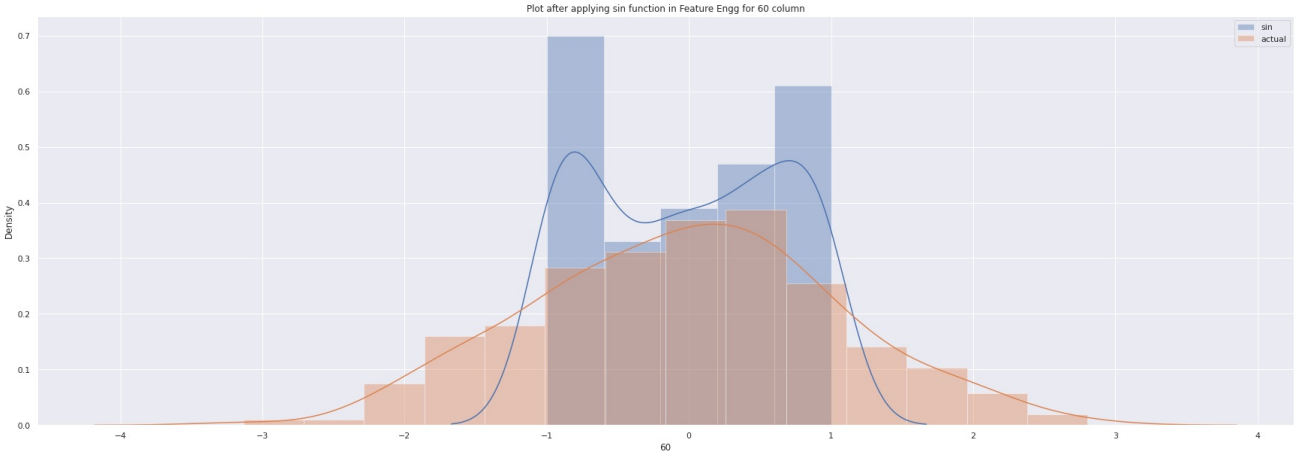
In [ ]:

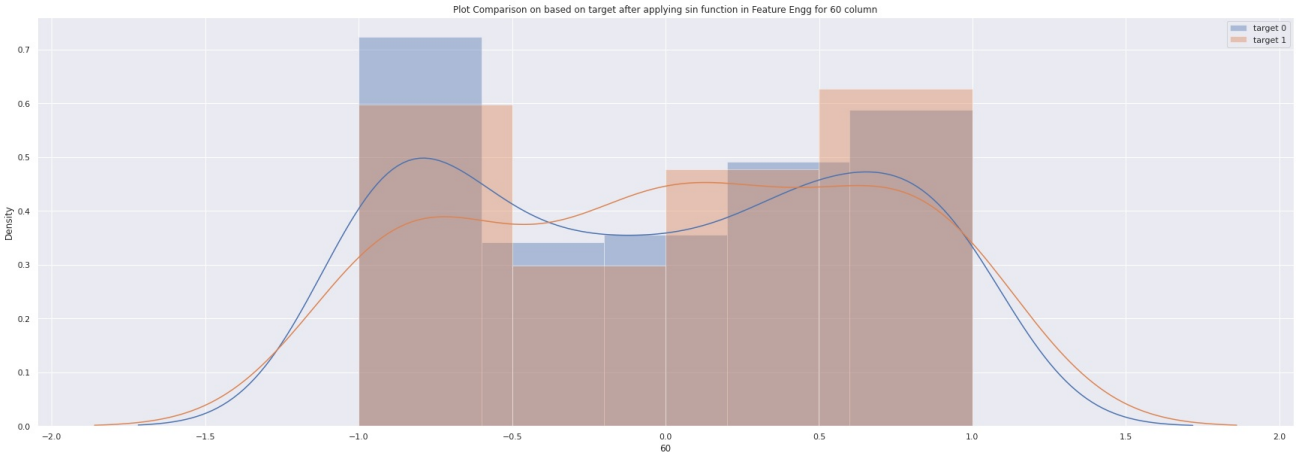
```
#ref : : https://docs.scipy.org/doc/numpy/reference/routines.math.html)
sin_data = np.sin(data)
cos_data = np.cos(data)
tan_data = np.tan(data)

print("sin data:",sin_data)
print("cos_data:",cos_data)
print("tan_data:", tan_data)
```

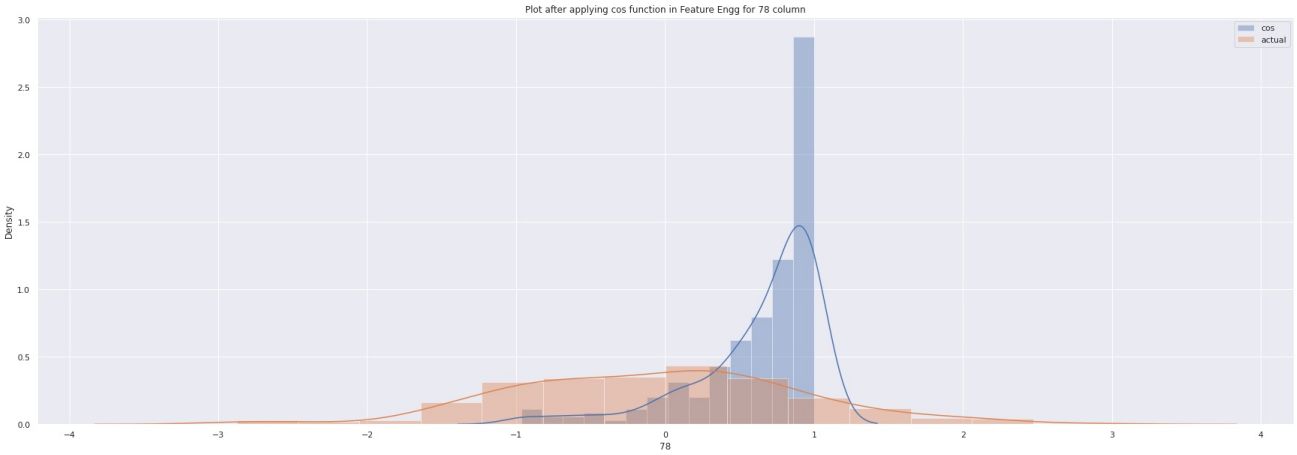
```
sin data:             0         1         2  ...       297       298       299
0   -0.875756 -0.897470 -0.577775  ...  0.422317  0.818043 -0.733869
1   -0.738606  0.267695  0.989476  ... -0.028996 -0.935472 -0.336314
2    0.098838  0.983701 -0.668359  ... -0.147460 -0.601997  0.663135
3   -0.835477 -0.793172 -0.974166  ... -0.882429  0.720831 -0.256508
4    0.724976 -0.998091  0.498615  ...  0.991717 -0.382961 -0.599599
..        ...       ...       ...  ...       ...       ...       ...
245 -0.067948 -0.182964 -0.913985  ...  0.928740  0.858327 -0.391260
246 -0.231870 -0.980502 -0.887362  ... -0.852631  0.679441  0.127651
247 -0.727446 -0.965561 -0.690370  ...  0.625678 -0.939099 -0.754571
248 -0.435866 -0.202588 -0.690370  ... -0.854503 -0.519273 -0.092866
249  0.663135  0.874304  0.971859  ... -0.992904 -0.196709  0.599599

[250 rows x 300 columns]
cos_data:             0         1         2  ...       297       298       299
0    0.482754  0.441075  0.816196  ...  0.906448  0.575157  0.679291
1    0.674137  0.963504 -0.144694  ...  0.999580 -0.353400  0.941750
2    0.995104  0.179813  0.743839  ...  0.989068  0.798498  0.748499
3    0.549526  0.608998  0.225831  ...  0.470446  0.693111 -0.966542
4    0.688774  0.061757  0.866824  ...  0.128441  0.923764  0.800301
..        ...       ...       ...  ...       ...       ...       ...
245  0.997689  0.983120  0.405747  ...  0.370731  0.513103  0.920280
246  0.972747  0.196509 -0.461073  ...  0.522514  0.733730  0.991819
247 -0.686165 -0.260175  0.723457  ...  0.780082  0.343646  0.656219
248  0.900012  0.979264  0.723457  ... -0.519447  0.854608  0.995679
249  0.748499  0.485379  0.235562  ... -0.118922  0.980462  0.800301

[250 rows x 300 columns]
tan_data:             0          1         2  ...       297       298       299
0   -1.814085  -2.034733 -0.707888  ...  0.465903  1.422294 -1.080345
1   -1.095631   0.277835 -6.838409  ... -0.029008  2.647059 -0.357116
2    0.099325   5.470689 -0.898526  ... -0.149090 -0.753912  0.885953
3   -1.520360  -1.302422 -4.313690  ... -1.875727  1.039993  0.265387
4    1.052561 -16.161588  0.575221  ...  7.721217 -0.414566 -0.749217
..        ...        ...       ...  ...       ...       ...       ...
245 -0.068105  -0.186105 -2.252597  ...  2.505157  1.672815 -0.425153
246 -0.238367  -4.989601  1.924561  ... -1.631787  0.926008  0.128704
247  1.060162   3.711197 -0.954265  ...  0.802067 -2.732754 -1.149878
248 -0.484289  -0.206878 -0.954265  ...  1.645024 -0.607615 -0.093269
249  0.885953   1.801282  4.125714  ...  8.349231 -0.200629  0.749217

[250 rows x 300 columns]
```

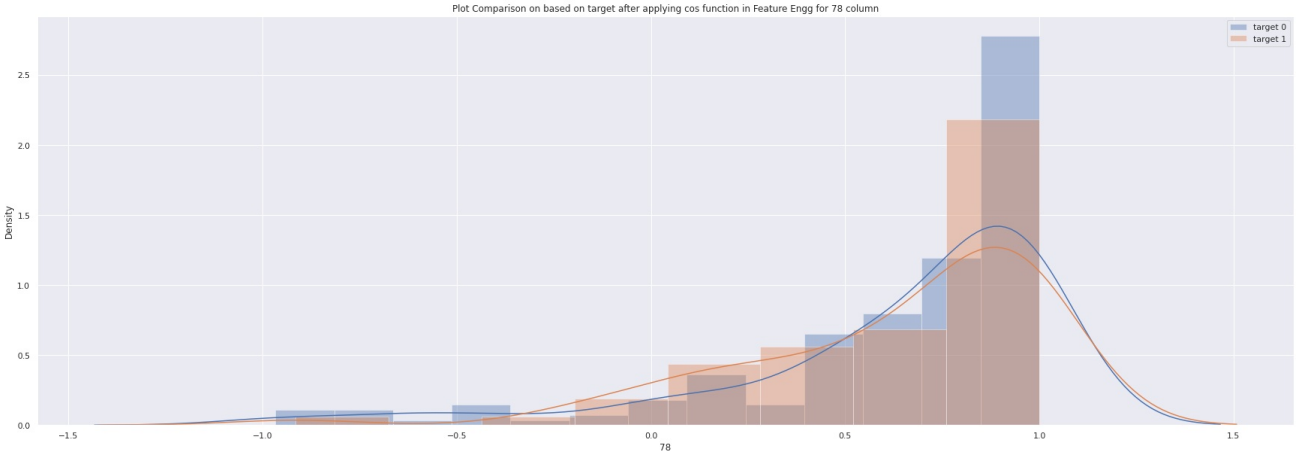
In [ ]:

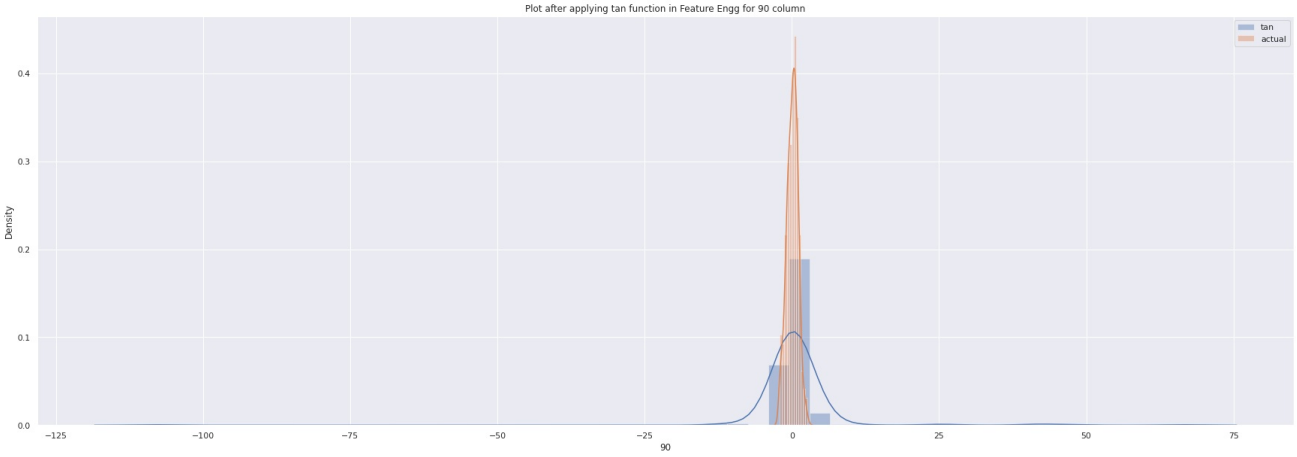
```
def feature_function(fe_name, fe_var):
  '''
  Parameter:
  fe_name: name of transformation Feature engg (string)
  fe_var: data after applying feature engineering
  Return:
  Plot 2 graphs.
  First plot for transformation of actual plot and after applying feature␣
  ,→engg plot for any 'r' column
  Second plot for showing the plot of applying feature engg on the based on␣
  ,→target value for any 'r' column
  '''
  r = str(np.random.randint(0,300))
  plt.figure(1)
  sns.distplot(fe_var[r], label=fe_name)
  sns.distplot(train_data[r], label='actual')
  plt.title('Plot after applying {} function in Feature Engg for {} column'.format(fe_name,r))
  plt.legend()
  plt.figure(2)
  sns.distplot(fe_var[train_data['target']==0][r], label='target 0')
  sns.distplot(fe_var[train_data['target']==1][r], label='target 1')
  plt.title('Plot Comparison on based on target after applying {} function in Feature Engg for {} column'.format(
fe_name,r))

  plt.legend()

  plt.show()

feature_function('sin',sin_data)
feature_function('cos',cos_data)
feature_function('tan',tan_data)
```

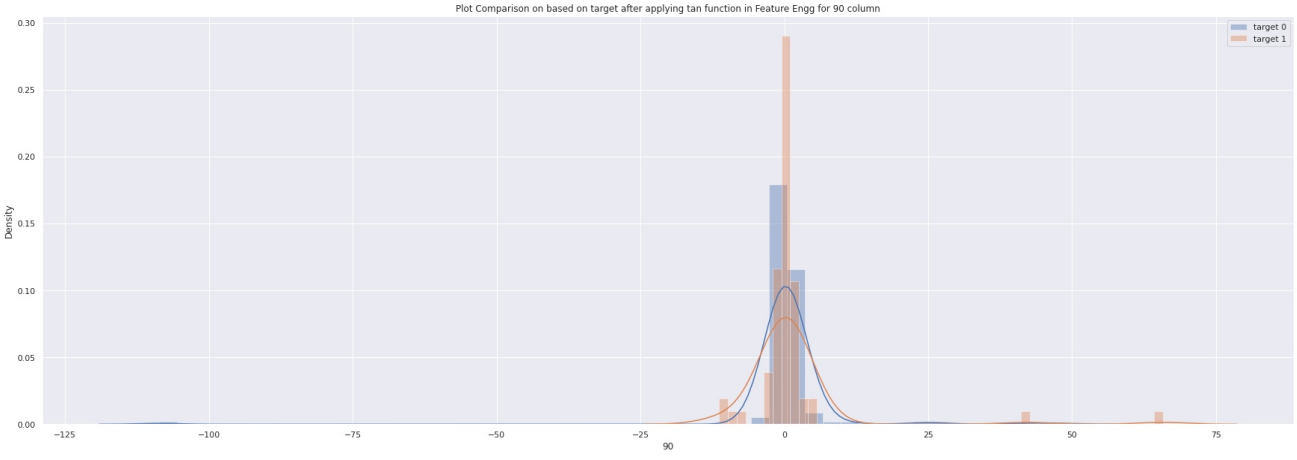












In [ ]:

```
train_data['mean_sin'] = np.mean(sin_data,axis=1)
train_data['cos_sin'] = np.mean(cos_data,axis=1)
train_data['mean_tan'] = np.mean(tan_data,axis=1)
```

In [ ]:

```
train_data.head()
```

Out[ ]:

	id	target	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	1
0	0	1.0	-1.067	-1.114	-0.616	0.376	1.090	0.467	-0.422	0.460	-0.443	-0.338	0.416	-2.177	-0.326	0.340	1.174	-0.24
1	1	0.0	-0.831	0.271	1.716	1.096	1.731	-0.197	1.904	-0.265	0.557	1.202	0.542	0.424	-1.572	-0.968	-1.483	0.564
2	2	0.0	0.099	1.390	-0.732	-1.065	0.005	-0.081	-1.450	0.317	-0.624	-0.017	-0.665	1.905	0.376	-1.373	1.587	1.464
3	3	1.0	-0.989	-0.916	-1.343	0.145	0.543	0.636	1.127	0.189	-0.118	-0.638	0.760	-0.360	-2.048	-0.996	-0.361	0.962
4	4	0.0	0.811	-1.509	0.522	-0.360	-0.220	-0.959	0.334	-0.566	-0.656	-0.499	-0.653	-0.058	-0.046	0.654	-0.697	-1.17

5 rows × 307 columns

We apply hyperbolic function

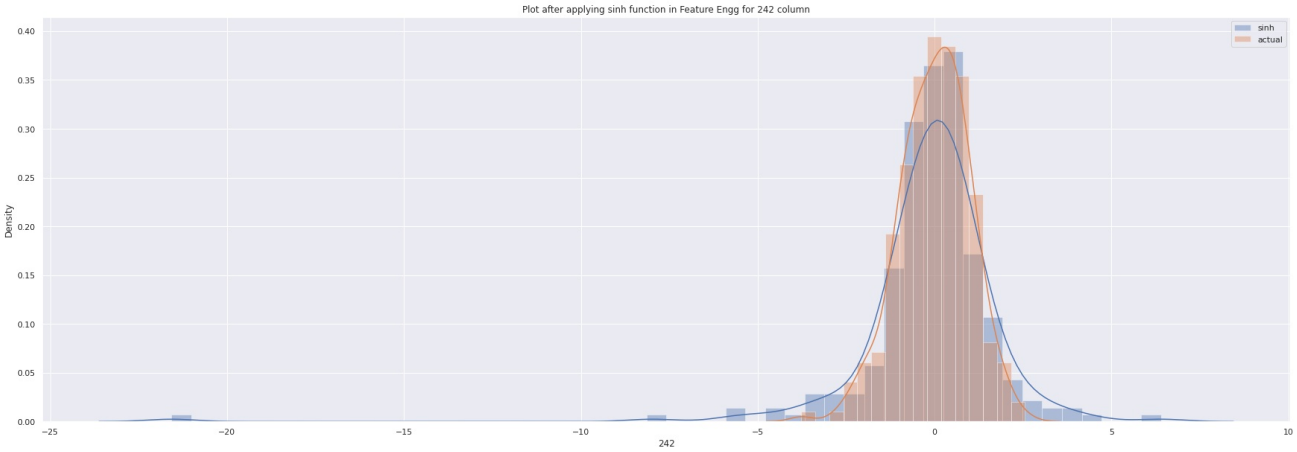
In [ ]:

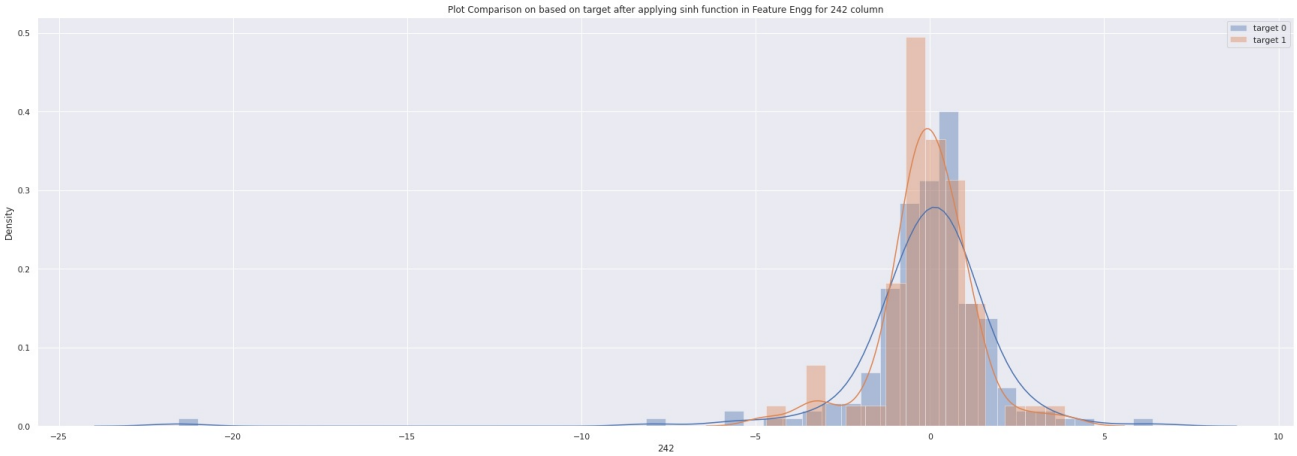
```
sinh_data = np.sinh(data)
cosh_data = np.cosh(data)
tanh_data = np.tanh(data)
print(sinh_data)
print(cosh_data)
print(tanh_data)
```

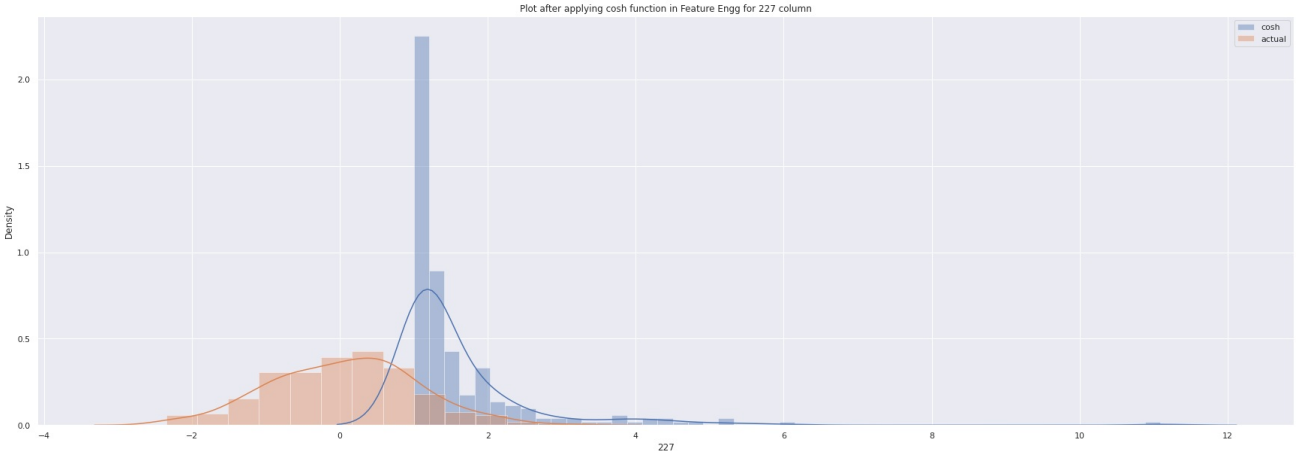
```
            0         1         2  ...       297       298        299
0   -1.281304 -1.359138 -0.655703  ...  0.449946  1.111409  -0.920463
1   -0.930000  0.274329  2.691226  ... -0.029004 -3.379222  -0.349765
2    0.099162  1.882887 -0.799144  ... -0.148541 -0.691878   0.790203
3   -1.158298 -1.049578 -1.784728  ... -1.304185  0.894804  14.980370
4    0.902872 -2.150538  0.546031  ...  1.996346 -0.403195  -0.688233
..        ...       ...       ...  ...       ...       ...        ...
245 -0.068052 -0.185040 -1.425997  ...  1.493226  1.225190  -0.412915
246 -0.236141 -1.846914 -3.819583  ... -1.207867  0.818436   0.128350
247 -5.074783 -3.049549 -0.837913  ...  0.728675 -1.545979  -0.963046
248 -0.466445 -0.205418 -0.837913  ... -4.092895 -0.573536  -0.093134
249  0.790203  1.276433  1.764359  ... -2.617481 -0.199296   0.688233

[250 rows x 300 columns]
            0         1         2  ...       297       298        299
0    1.625343  1.687382  1.195804  ...  1.096563  1.495069   1.359137
1    1.365613  1.036946  2.871009  ...  1.000421  3.524081   1.059403
2    1.004905  2.131963  1.280091  ...  1.010972  1.216016   1.274528
3    1.530247  1.449695  2.045790  ...  1.643441  1.341892  15.013710
4    1.347285  2.371669  1.139364  ...  2.232800  1.078224   1.213946
..        ...       ...       ...  ...       ...       ...        ...
245  1.002313  1.016976  1.741685  ...  1.797144  1.581484   1.081896
246  1.027503  2.100260  3.948318  ...  1.568102  1.292222   1.008203
247  5.172371  3.209323  1.304645  ...  1.237323  1.841209   1.388329
248  1.103436  1.020880  1.304645  ...  4.213287  1.152798   1.004328
249  1.274528  1.621506  2.028044  ...  2.802000  1.019666   1.213946

[250 rows x 300 columns]
            0         1         2  ...       297       298       299
0   -0.788328 -0.805472 -0.548337  ...  0.410323  0.743383 -0.677241
1   -0.681013  0.264555  0.937380  ... -0.028992 -0.958895 -0.330153
2    0.098678  0.883171 -0.624287  ... -0.146929 -0.568971  0.619997
3   -0.756936 -0.724000 -0.872391  ... -0.793570  0.666823  0.997779
4    0.670142 -0.906761  0.479242  ...  0.894100 -0.373944 -0.566939
..        ...       ...       ...  ...       ...       ...       ...
245 -0.067895 -0.181951 -0.818745  ...  0.830889  0.774709 -0.381659
246 -0.229821 -0.879374 -0.967395  ... -0.770274  0.633356  0.127306
247 -0.981133 -0.950216 -0.642253  ...  0.588913 -0.839654 -0.693673
248 -0.422721 -0.201216 -0.642253  ... -0.971426 -0.497516 -0.092733
249  0.619997  0.787190  0.869981  ... -0.934147 -0.195452  0.566939

[250 rows x 300 columns]
```

In [ ]:

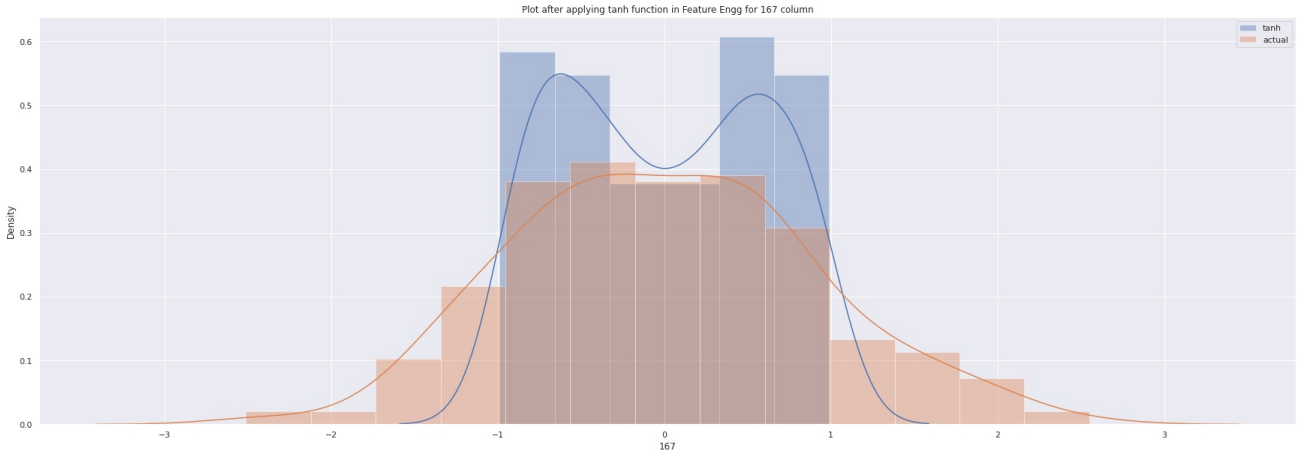
```
feature_function('sinh',sinh_data)
feature_function('cosh',cosh_data)
feature_function('tanh',tanh_data)
```


Plot after applying sinh function in Feature Engg for 242 column
sinh
actual
Density
242


Plot Comparison on based on target after applying sinh function in Feature Engg for 242 column
target 0
target 1
Density
242


Plot after applying cosh function in Feature Engg for 227 column
cosh
actual
Density
227


Plot Comparison on based on target after applying cosh function in Feature Engg for 227 column
target 0
target 1
Density
227



In [ ]:

```
train_data['mean_sinh'] = np.mean(sinh_data,axis=1)
train_data['mean_cosh'] = np.mean(cosh_data,axis=1)
train_data['mean_tanh'] = np.mean(tanh_data,axis=1)
```

In [ ]:

```
train_data.head()
```

Out[ ]:

	id	target	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	0	1.0	-1.067	-1.114	-0.616	0.376	1.090	0.467	-0.422	0.460	-0.443	-0.338	0.416	-2.177	-0.326	0.340	1.174	-0.
1	1	0.0	-0.831	0.271	1.716	1.096	1.731	-0.197	1.904	-0.265	0.557	1.202	0.542	0.424	-1.572	-0.968	-1.483	0.5
2	2	0.0	0.099	1.390	-0.732	-1.065	0.005	-0.081	-1.450	0.317	-0.624	-0.017	-0.665	1.905	0.376	-1.373	1.587	1.4
3	3	1.0	-0.989	-0.916	-1.343	0.145	0.543	0.636	1.127	0.189	-0.118	-0.638	0.760	-0.360	-2.048	-0.996	-0.361	0.9
4	4	0.0	0.811	-1.509	0.522	-0.360	-0.220	-0.959	0.334	-0.566	-0.656	-0.499	-0.653	-0.058	-0.046	0.654	-0.697	-1.

5 rows × 310 columns

In [ ]:

```
# We have taken ref https://numpy.org/doc/stable/reference/routines.math.html

exp_data = np.exp(data)
expm1_data = np.expm1(data)
exp2_data = np. exp2(data)
```

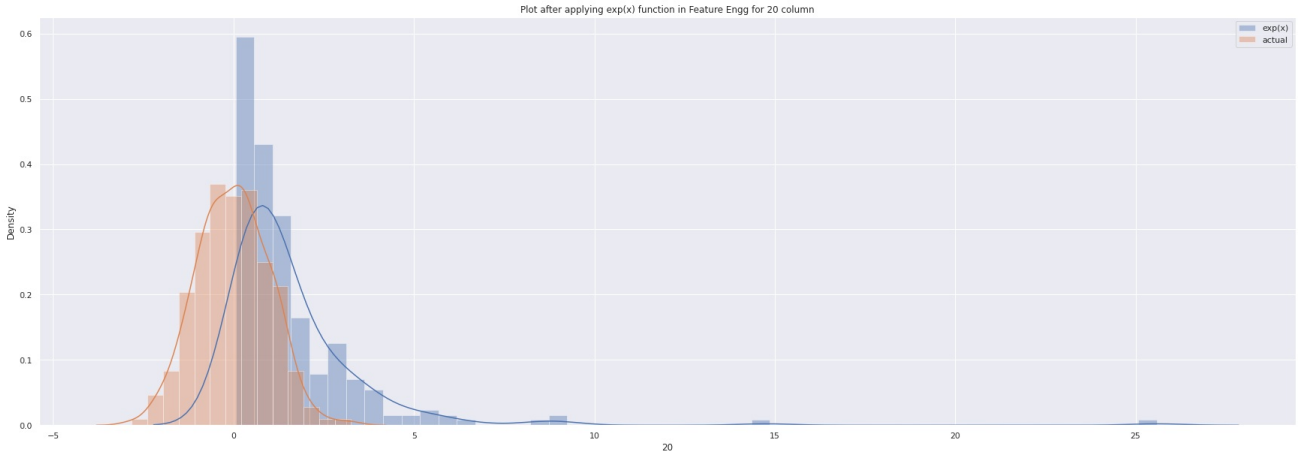
In [ ]:

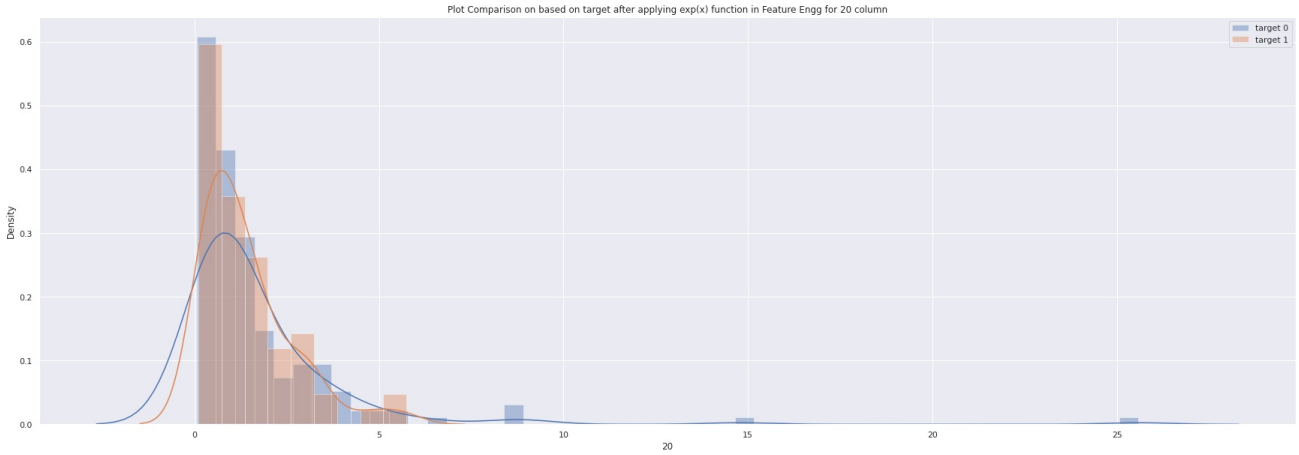
```
print(exp_data)
print(expm1_data)
print(exp2_data)
```

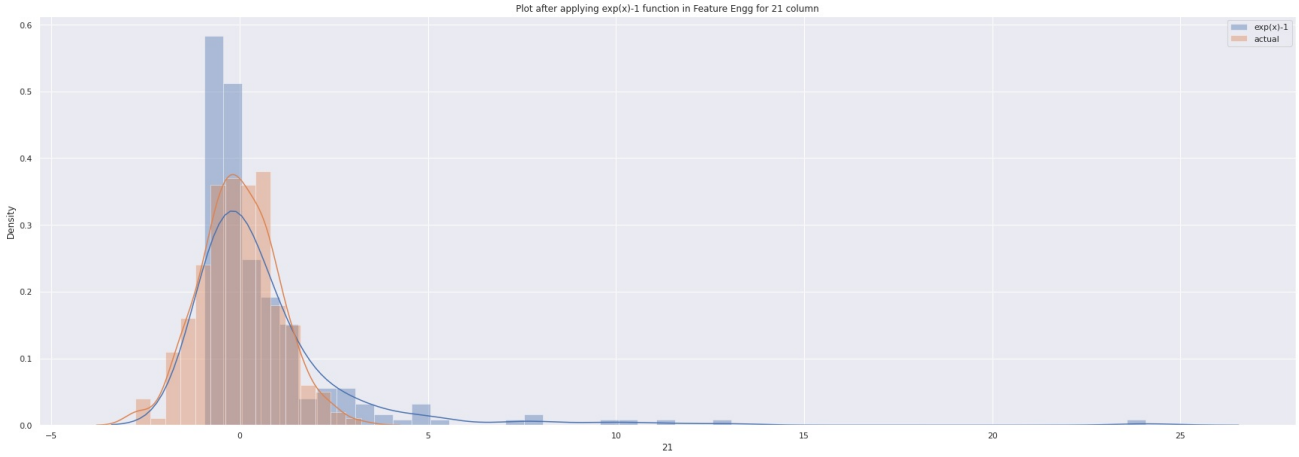
```
            0         1         2  ...       297       298        299
0    0.344039  0.328243  0.540101  ...  1.546509  2.606478   0.438673
1    0.435613  1.311275  5.562235  ...  0.971416  0.144858   0.709638
2    1.104066  4.014850  0.480946  ...  0.862431  0.524138   2.064731
3    0.371948  0.400116  0.261061  ...  0.339256  2.236696  29.994079
4    2.250157  0.221131  1.685395  ...  4.229146  0.675029   0.525713
..        ...       ...       ...  ...       ...       ...        ...
245  0.934260  0.831936  0.315688  ...  3.290370  2.806674   0.668981
246  0.791362  0.253346  0.128735  ...  0.360235  2.110659   1.136553
247  0.097588  0.159773  0.466732  ...  1.965998  0.295230   0.425283
248  0.636991  0.815462  0.466732  ...  0.120392  0.579262   0.911194
249  2.064731  2.897940  3.792404  ...  0.184520  0.820370   1.902179

[250 rows x 300 columns]
            0         1         2  ...       297       298        299
0   -0.655961 -0.671757 -0.459899  ...  0.546509  1.606478  -0.561327
1   -0.564387  0.311275  4.562235  ... -0.028584 -0.855142  -0.290362
2    0.104066  3.014850 -0.519054  ... -0.137569 -0.475862   1.064731
3   -0.628052 -0.599884 -0.738939  ... -0.660744  1.236696  28.994079
4    1.250157 -0.778869  0.685395  ...  3.229146 -0.324971  -0.474287
..        ...       ...       ...  ...       ...       ...        ...
245 -0.065740 -0.168064 -0.684312  ...  2.290370  1.806674  -0.331019
246 -0.208638 -0.746654 -0.871265  ... -0.639765  1.110659   0.136553
247 -0.902412 -0.840227 -0.533268  ...  0.965998 -0.704770  -0.574717
248 -0.363009 -0.184538 -0.533268  ... -0.879608 -0.420738  -0.088806
249  1.064731  1.897940  2.792404  ... -0.815480 -0.179630   0.902179

[250 rows x 300 columns]
            0         1         2  ...       297       298        299
0    0.477311  0.462011  0.652477  ...  1.352848  1.942615   0.564874
1    0.562139  1.206644  3.285243  ...  0.980099  0.262066   0.788400
2    1.071031  2.620787  0.602069  ...  0.902501  0.639050   1.652901
3    0.503827  0.529976  0.394200  ...  0.472701  1.747146  10.563383
4    1.754427  0.351355  1.435945  ...  2.716973  0.761544   0.640380
..        ...       ...       ...  ...       ...       ...        ...
245  0.953960  0.880259  0.449689  ...  2.283109  2.044857   0.756808
246  0.850274  0.386088  0.241484  ...  0.492775  1.678299   1.092778
247  0.199298  0.280486  0.589678  ...  1.597704  0.429283   0.552865
248  0.731536  0.868140  0.589678  ...  0.230526  0.684916   0.937571
249  1.652901  2.090720  2.519260  ...  0.309927  0.871758   1.561573

[250 rows x 300 columns]
```

In [ ]:

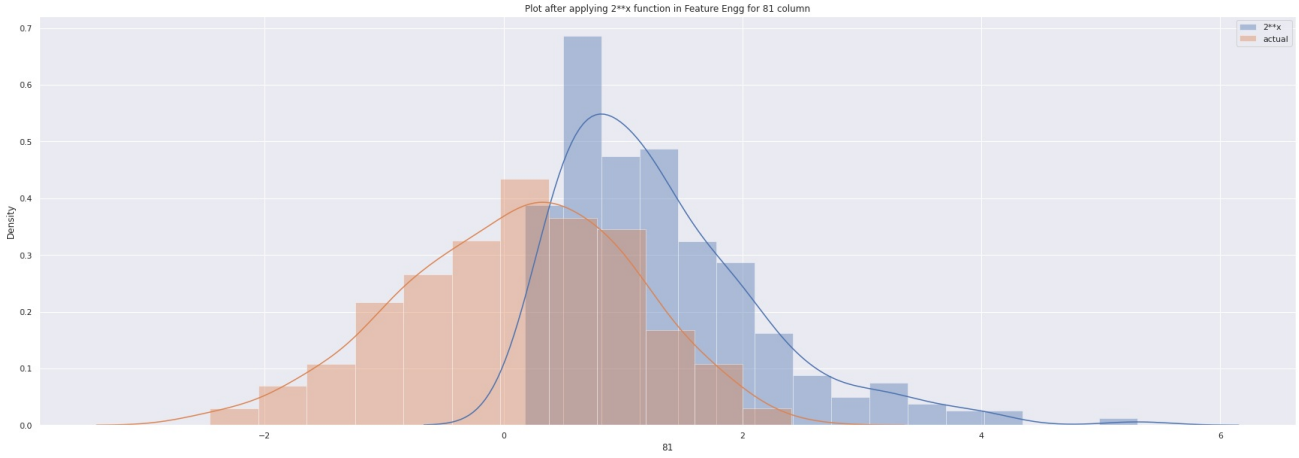
```
feature_function('exp(x)',exp_data)
feature_function('exp(x)-1',expm1_data)
feature_function('2**x',exp2_data)
```

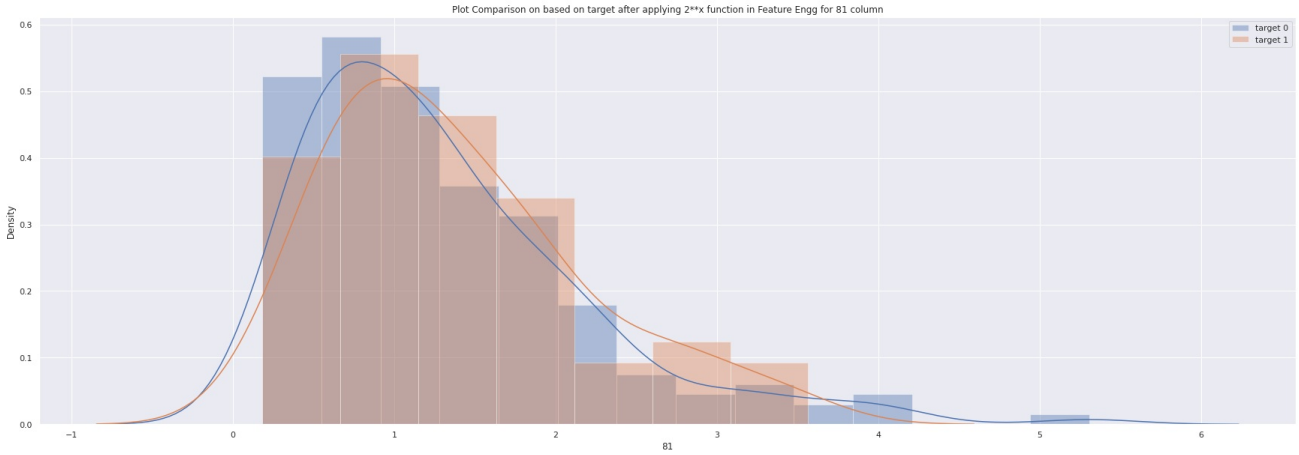



Plot Comparison on based on target after applying exp(x) function in Feature Engg for 20 column
target 0
target 1
Density
20


Plot after applying exp(x)-1 function in Feature Engg for 21 column
exp(x)-1
actual
Density
21


Plot Comparison on based on target after applying exp(x)-1 function in Feature Engg for 21 column
target 0
target 1
Density
21


Plot after applying 2**x function in Feature Engg for 81 column
2**x
actual
Density
81



In [ ]:

```
train_data['mean_exp'] = np.mean(exp_data,axis=1)
train_data['mean_expm1'] = np.mean(expm1_data,axis=1)
train_data['mean_exp2'] = np. mean(exp2_data,axis=1)
```

In [ ]:

```
train_data.head()
```

Out[ ]:

	id	target	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	0	1.0	-1.067	-1.114	-0.616	0.376	1.090	0.467	-0.422	0.460	-0.443	-0.338	0.416	-2.177	-0.326	0.340	1.174	-0.
1	1	0.0	-0.831	0.271	1.716	1.096	1.731	-0.197	1.904	-0.265	0.557	1.202	0.542	0.424	-1.572	-0.968	-1.483	0.5
2	2	0.0	0.099	1.390	-0.732	-1.065	0.005	-0.081	-1.450	0.317	-0.624	-0.017	-0.665	1.905	0.376	-1.373	1.587	1.4
3	3	1.0	-0.989	-0.916	-1.343	0.145	0.543	0.636	1.127	0.189	-0.118	-0.638	0.760	-0.360	-2.048	-0.996	-0.361	0.9
4	4	0.0	0.811	-1.509	0.522	-0.360	-0.220	-0.959	0.334	-0.566	-0.656	-0.499	-0.653	-0.058	-0.046	0.654	-0.697	-1.

5 rows x 313 columns