1_EDA

April 15, 2020

1 Kaggle Problem

Don't Overfit! II is a challenging problem where we must avoid models to be overfitted (or crooked way to learn) given very small amount of training samples.

As per Kaggle say," It was a competition that challenged mere mortals to model a 20,000x200 matrix of continuous variables using only 250 training samples… without overfitting. "

Dataset can be download here: https://www.kaggle.com/c/dont-overfit-ii/overview

Dimension of train.csv – 250 samples and 300 features and 1 class label and 1 Id: (250,302)

Dimension of test.csv – 19750 samples and 300 features and 1 Id: (19750,301)

So, with the small amount of train data given, we must do to task carefully to avoid overfitting easily.

What do we need to predict? We are predicting the binary target value (binary classification) associated with each row which contains 300 continuous feature values. Also without overfitting with the minimal set of training samples given.

Evaluation Score

As per Kaggle problem statement, the score will be evaluated based on AUCROC between predicted target and actual target.

2 Import Necessary Files

[1]:
```
# To read file
import pandas as pd
# For plotting purpose
import matplotlib.pyplot as plt
import seaborn as sns
# For 3D plot
import plotly.express as px
# For computational stuff
import numpy as np
# Dimension redction technnique
from sklearn.manifold import TSNE
```

3 Read train data

[2]:
```python
# Locate parent directory
data_dir = "./"

# Read csv file and display top 5 rows
df_train = pd.read_csv(data_dir+'/train.csv')
df_train.head(5)
```

[2]:
```
   id  target      0      1      2      3      4      5      6      7  ...  \
0   0     1.0 -0.098  2.165  0.681 -0.614  1.309 -0.455 -0.236  0.276  ...
1   1     0.0  1.081 -0.973 -0.383  0.326 -0.428  0.317  1.172  0.352  ...
2   2     1.0 -0.523 -0.089 -0.348  0.148 -0.022  0.404 -0.023 -0.172  ...
3   3     1.0  0.067 -0.021  0.392 -1.637 -0.446 -0.725 -1.035  0.834  ...
4   4     1.0  2.347 -0.831  0.511 -0.021  1.225  1.594  0.585  1.509  ...

     290    291    292    293    294    295    296    297    298    299
0  0.867  1.347  0.504 -0.649  0.672 -2.097  1.051 -0.414  1.038 -1.065
1 -0.165 -1.695 -1.257  1.359 -0.808 -1.624 -0.458 -1.099 -0.936  0.973
2  0.013  0.263 -1.222  0.726  1.444 -1.165 -1.544  0.004  0.800 -1.211
3 -0.404  0.640 -0.595 -0.966  0.900  0.467 -0.562 -0.254 -0.533  0.238
4  0.898  0.134  2.415 -0.996 -1.006  1.378  1.246  1.478  0.428  0.253

[5 rows x 302 columns]
```

As expected! There are 302 columns in the train.csv as we mentioned in problem overview.

4 Exploratory Data Analysis (EDA)

Describe train data

[3]:
```python
df_train.describe()
```

[3]:
```
               id      target           0           1           2           3  \
count  250.000000  250.000000  250.000000  250.000000  250.000000  250.000000
mean   124.500000    0.640000    0.023292   -0.026872    0.167404    0.001904
std     72.312977    0.480963    0.998354    1.009314    1.021709    1.011751
min      0.000000    0.000000   -2.319000   -2.931000   -2.477000   -2.359000
25%     62.250000    0.000000   -0.644750   -0.739750   -0.425250   -0.686500
50%    124.500000    1.000000   -0.015500    0.057000    0.184000   -0.016500
75%    186.750000    1.000000    0.677000    0.620750    0.805000    0.720000
max    249.000000    1.000000    2.567000    2.419000    3.392000    2.771000

                4           5           6           7  ...         290  \
count  250.000000  250.000000  250.000000  250.000000  ...  250.000000
mean     0.001588   -0.007304    0.032052    0.078412  ...    0.044652
std      1.035411    0.955700    1.006657    0.939731  ...    1.011416
```

```
min     -2.566000   -2.845000   -2.976000   -3.444000  …   -2.804000
25%     -0.659000   -0.643750   -0.675000   -0.550750  …   -0.617000
50%     -0.023000    0.037500    0.060500    0.183500  …    0.067500
75%      0.735000    0.660500    0.783250    0.766250  …    0.797250
max      2.901000    2.793000    2.546000    2.846000  …    2.865000

              291         292         293         294         295         296  \
count  250.000000  250.000000  250.000000  250.000000  250.000000  250.000000
mean     0.126344    0.018436   -0.012092   -0.065720   -0.106112    0.046472
std      0.972567    0.954229    0.960630    1.057414    1.038389    0.967661
min     -2.443000   -2.757000   -2.466000   -3.287000   -3.072000   -2.634000
25%     -0.510500   -0.535750   -0.657000   -0.818500   -0.821000   -0.605500
50%      0.091000    0.057500   -0.021000   -0.009000   -0.079500    0.009500
75%      0.804250    0.631500    0.650250    0.739500    0.493000    0.683000
max      2.801000    2.736000    2.596000    2.226000    3.131000    3.236000

              297         298         299
count  250.000000  250.000000  250.000000
mean     0.006452    0.009372   -0.128952
std      0.998984    1.008099    0.971219
min     -2.776000   -3.211000   -3.500000
25%     -0.751250   -0.550000   -0.754250
50%      0.005500   -0.009000   -0.132500
75%      0.794250    0.654250    0.503250
max      2.626000    3.530000    2.771000

[8 rows x 302 columns]
```

Info train data

[4]:
```python
df_train.info()
```

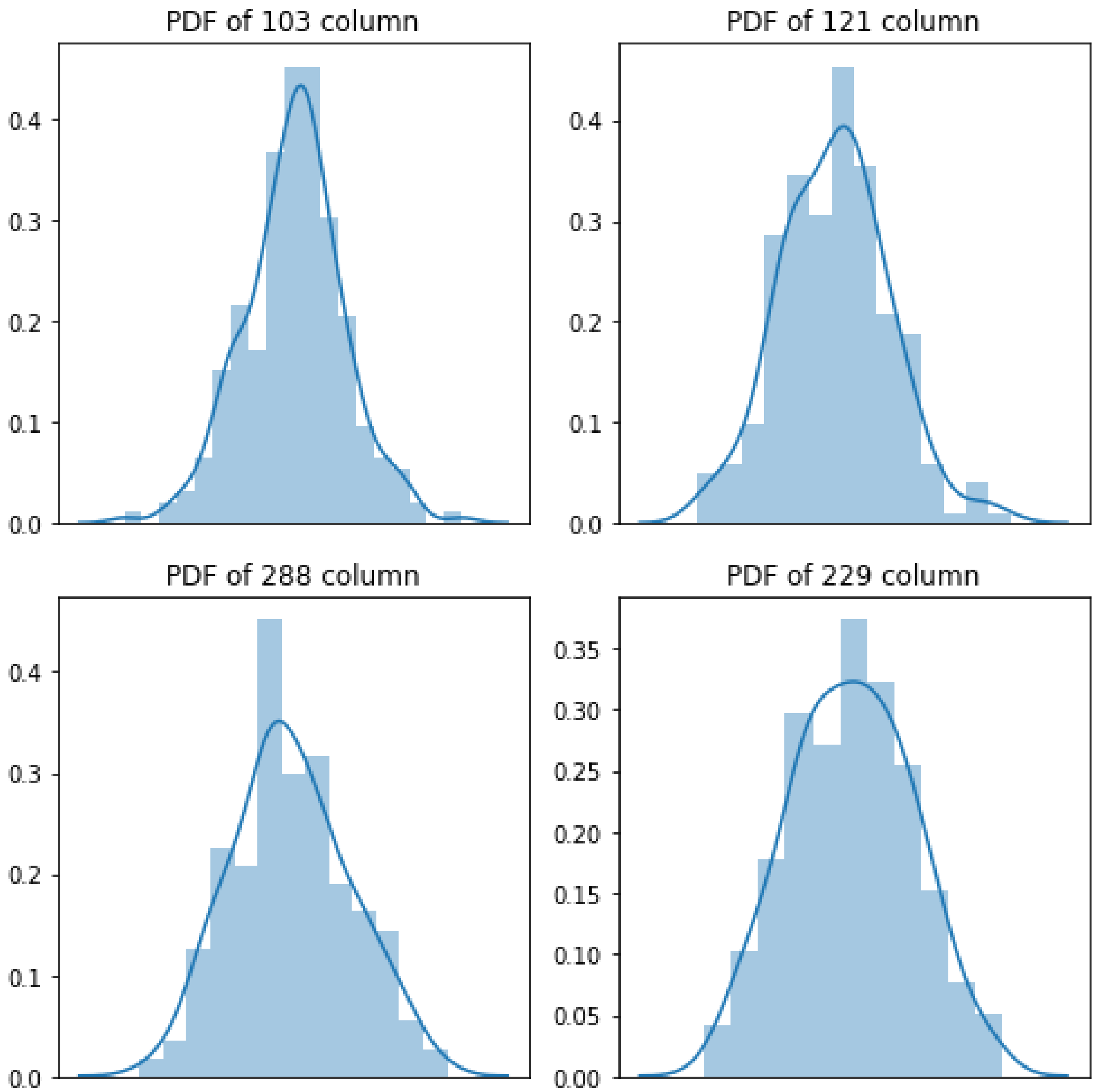
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Columns: 302 entries, id to 299
dtypes: float64(301), int64(1)
memory usage: 590.0 KB
```

4.1 Probability Density Function (PDF)

Run several time to generate any four random value that lie between 0 - 299 feature name to see the **PDF** plot

[5]:
```python
# Subplotting reference using sns (https://seaborn.pydata.org/examples/
 ↪distplot_options.html)
# Generate using numpy random (See Docs: https://docs.scipy.org/doc/numpy-1.14.
 ↪0/reference/generated/numpy.random.randint.html#numpy.random.randint )
```

```python
# Seaborn distplot (See Docs: https://seaborn.pydata.org/generated/seaborn.
↪distplot.html)
# Hiding label on x axis using axes (Ref: https://stackoverflow.com/questions/
↪2176424/hiding-axis-text-in-matplotlib-plots)

fig, axes = plt.subplots(2,2, figsize=(7,7))
r = np.random.randint(0,299,4,dtype=int)

sns.distplot(df_train[str(r[0])], ax=axes[0,0])
axes[0,0].set_title('PDF of {} column'.format(r[0]))
axes[0,0].get_xaxis().set_visible(False)

sns.distplot(df_train[str(r[1])], ax=axes[0,1])
axes[0,1].set_title('PDF of {} column'.format(r[1]))
axes[0,1].get_xaxis().set_visible(False)

sns.distplot(df_train[str(r[2])], ax=axes[1,0])
axes[1,0].set_title('PDF of {} column'.format(r[2]))
axes[1,0].get_xaxis().set_visible(False)

sns.distplot(df_train[str(r[3])], ax=axes[1,1])
axes[1,1].set_title('PDF of {} column'.format(r[3]))
axes[1,1].get_xaxis().set_visible(False)

fig.tight_layout()
plt.show()
```



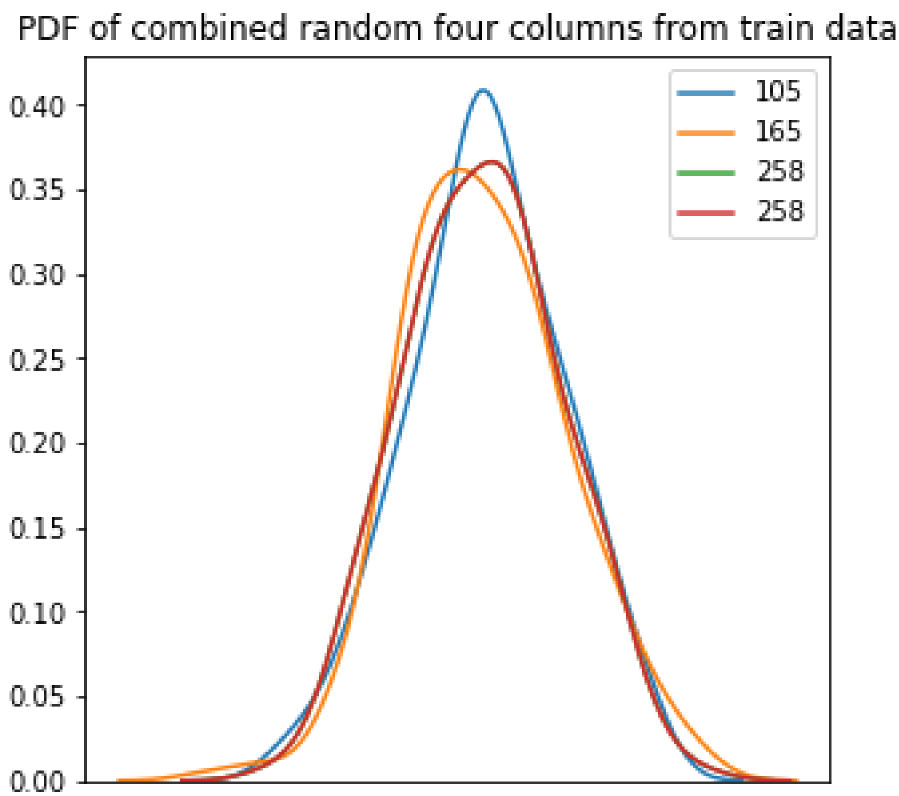
Observation: Ater running several times, it seems that most of them are **gaussian distribution**

> Note: If you want to know which feature names that its plotted, **See the legend** of each plot. Number corresponds to the feature name given in the train dataset.

Let try to overlapped **PDF** each other to see the difference of the plot.

[6]:
```
# Hiddem x-axis label using plt (Ref: https://stackoverflow.com/questions/
↪37039685/hide-axis-values-in-matplotlib/37045694)

plt.figure(figsize=(5,5))
r = np.random.randint(0,299,4,dtype=int)

sns.distplot(df_train[str(r[0])], hist=False, label=str(r[0]))
```

```
sns.distplot(df_train[str(r[1])], hist=False, label=str(r[1]))
sns.distplot(df_train[str(r[2])], hist=False, label=str(r[2]))
sns.distplot(df_train[str(r[3])], hist=False, label=str(r[3]))

plt.title('PDF of combined random four columns from train data')
plt.gca().axes.get_xaxis().set_visible(False)
plt.show()
```

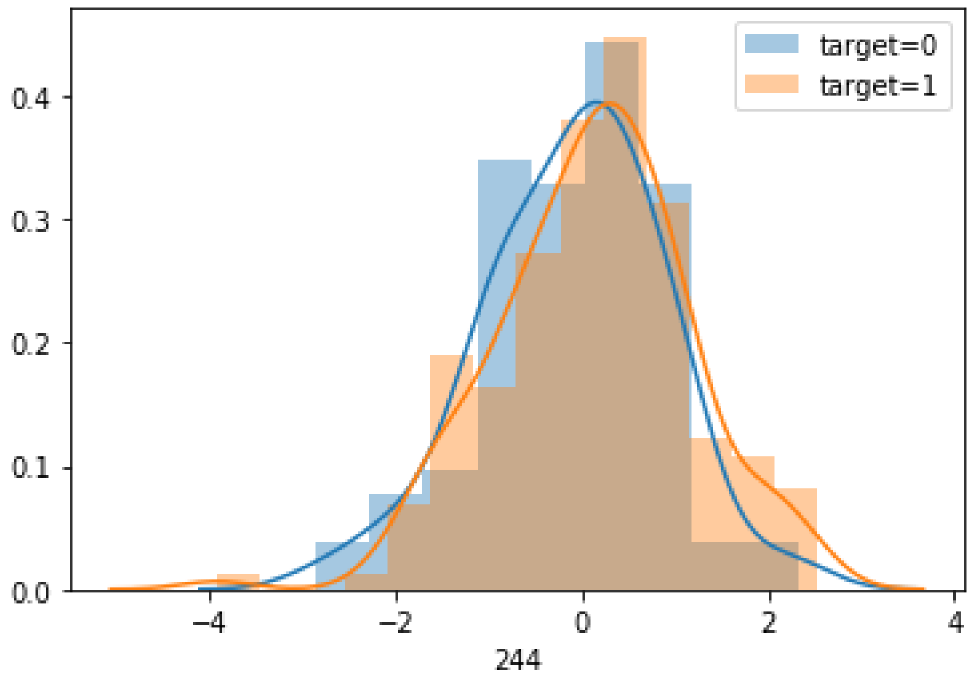


Observation: After running many times to generate any four random feature name, most of the PDF plot are **overlapping each other** (even with the tailness of the graph are mostly falling in the **same devation**).

[7]:
```
# PDF based on target

r = np.random.randint(0,299,1,dtype=int)

sns.distplot(df_train[df_train['target']==0][str(r[0])], label='target=0')
sns.distplot(df_train[df_train['target']==1][str(r[0])], label='target=1')
print('PDF of {} column on target based value'.format(r[0]))

plt.legend()
```

```
plt.show()
```

```
PDF of 244 column on target based value
```

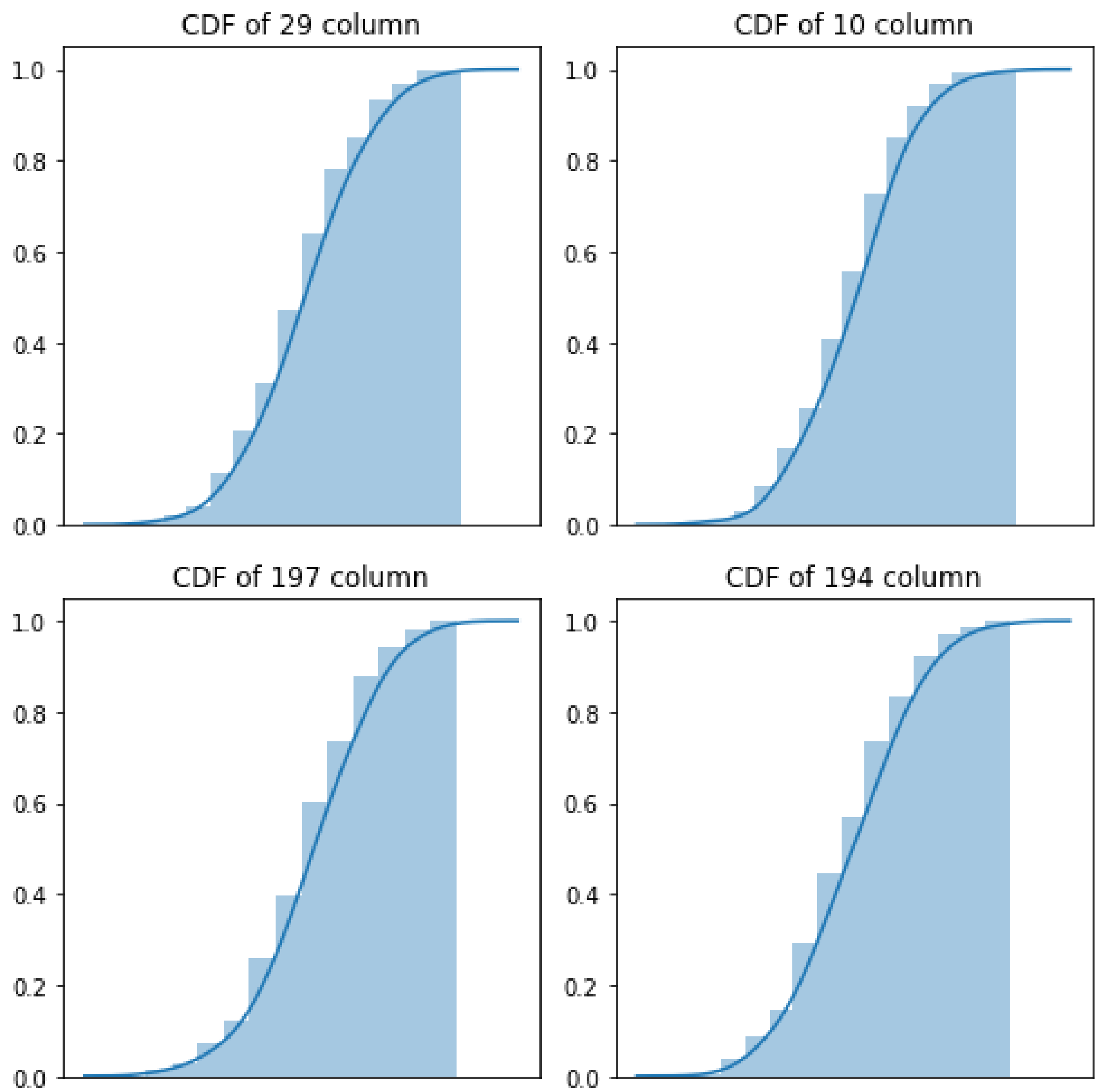


Its look like more overlapping there. So, it will harder to get better AUROC result. Let try for the best as much as possible

4.2 Cumulative Density Fuction (CDF)

[8]:
```
# Seaborn distplot for cumulative (Ref: https://stackoverflow.com/questions/
↪39297523/plot-cdf-cumulative-histogram-using-seaborn-python)

fig, axes = plt.subplots(2,2, figsize=(7,7))
r = np.random.randint(0,299,4,dtype=int)

sns.distplot(df_train[str(r[0])], ax=axes[0,0], hist_kws={'cumulative': True},␣
↪kde_kws={'cumulative': True})
axes[0,0].set_title('CDF of {} column'.format(r[0]))
axes[0,0].get_xaxis().set_visible(False)

sns.distplot(df_train[str(r[1])], ax=axes[0,1], hist_kws={'cumulative': True},␣
↪kde_kws={'cumulative': True})
axes[0,1].set_title('CDF of {} column'.format(r[1]))
axes[0,1].get_xaxis().set_visible(False)
```

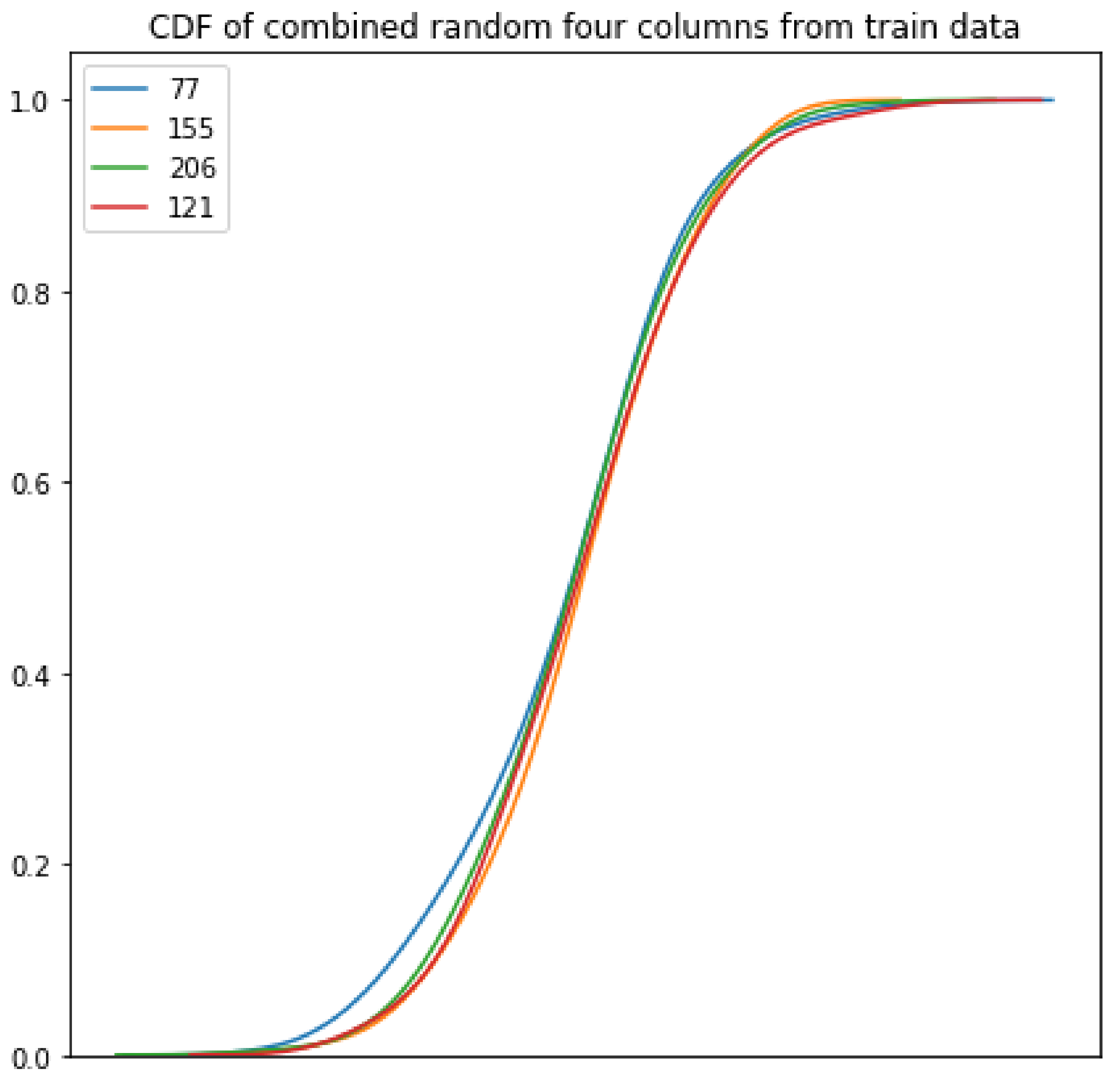
```
sns.distplot(df_train[str(r[2])], ax=axes[1,0], hist_kws={'cumulative': True},␣
 ↪kde_kws={'cumulative': True})
axes[1,0].set_title('CDF of {} column'.format(r[2]))
axes[1,0].get_xaxis().set_visible(False)

sns.distplot(df_train[str(r[3])], ax=axes[1,1], hist_kws={'cumulative': True},␣
 ↪kde_kws={'cumulative': True})
axes[1,1].set_title('CDF of {} column'.format(r[3]))
axes[1,1].get_xaxis().set_visible(False)

fig.tight_layout()
plt.show()
```



Observation: Well nothing seems to be different. Mostly feature have similar slope in CDF graph

Let try to overlapped **CDF** each other to see the difference of the plot.

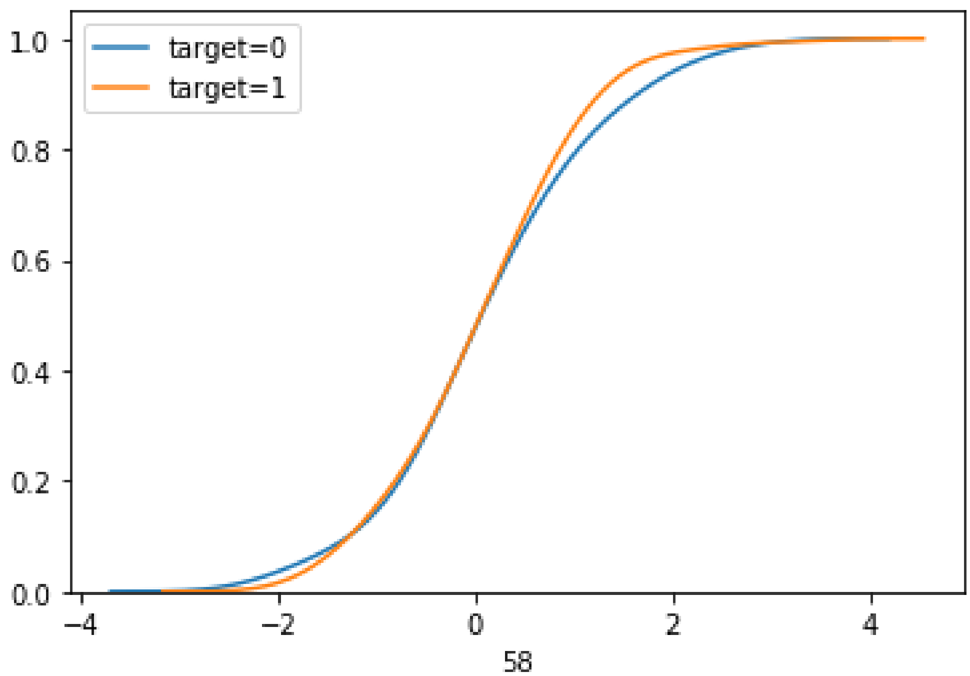
[9]:
```
plt.subplots(figsize=(7,7))
r = np.random.randint(0,299,4,dtype=int)

sns.distplot(df_train[str(r[0])], kde_kws={'cumulative': True}, hist=False,␣
 ↪label=str(r[0]))
sns.distplot(df_train[str(r[1])], kde_kws={'cumulative': True}, hist=False,␣
 ↪label=str(r[1]))
sns.distplot(df_train[str(r[2])], kde_kws={'cumulative': True}, hist=False,␣
 ↪label=str(r[2]))
sns.distplot(df_train[str(r[3])], kde_kws={'cumulative': True}, hist=False,␣
 ↪label=str(r[3]))

plt.title('CDF of combined random four columns from train data')
plt.gca().axes.get_xaxis().set_visible(False)
plt.show()
```



As expected! From the previous observation, slope of the CDF plot are similar.

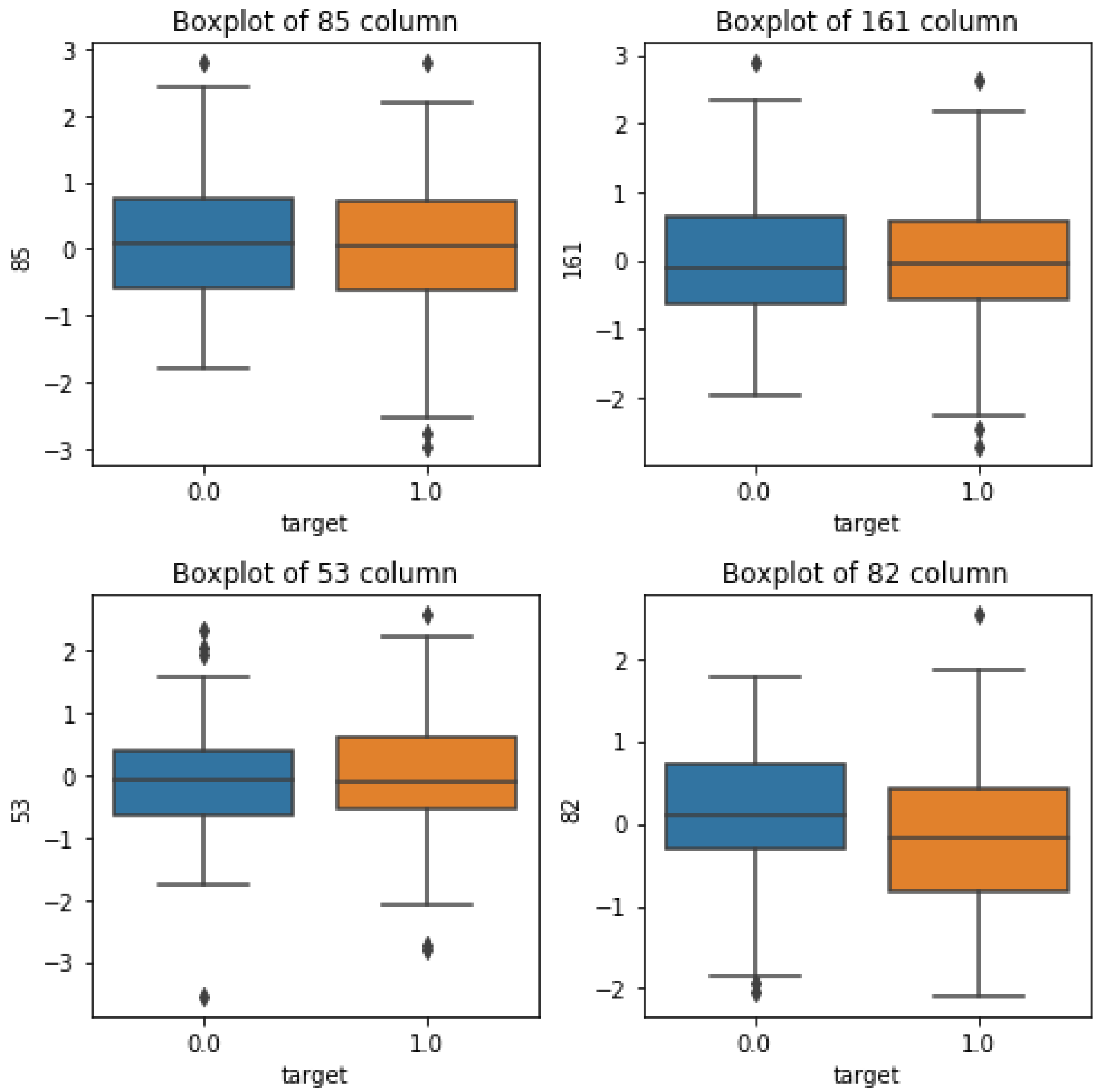
[10]:
```
# CDF based on target

r = np.random.randint(0,299,1,dtype=int)

sns.distplot(df_train[df_train['target']==0][str(r[0])], kde_kws={'cumulative':␣
 ↪True}, hist=False, label='target=0')
sns.distplot(df_train[df_train['target']==1][str(r[0])], kde_kws={'cumulative':␣
 ↪True}, hist=False, label='target=1')
print('PDF of {} column on target based value'.format(r[0]))

plt.legend()
plt.show()
```

PDF of 58 column on target based value



4.3 BoxPlot

[11]:
```
# Seaborn boxplot (See Docs: https://seaborn.pydata.org/generated/seaborn.
↪boxplot.html?highlight=boxplot#seaborn.boxplot)

fig, axes = plt.subplots(2,2, figsize=(7,7))
r = np.random.randint(0,299,4,dtype=int)

sns.boxplot(data=df_train, x='target', y=str(r[0]), ax=axes[0,0])
axes[0,0].set_title('Boxplot of {} column'.format(r[0]))

sns.boxplot(data=df_train, x='target', y=str(r[1]), ax=axes[0,1])
axes[0,1].set_title('Boxplot of {} column'.format(r[1]))

sns.boxplot(data=df_train, x='target', y=str(r[2]), ax=axes[1,0])
axes[1,0].set_title('Boxplot of {} column'.format(r[2]))

sns.boxplot(data=df_train, x='target', y=str(r[3]), ax=axes[1,1])
axes[1,1].set_title('Boxplot of {} column'.format(r[3]))

fig.tight_layout()
plt.show()
```

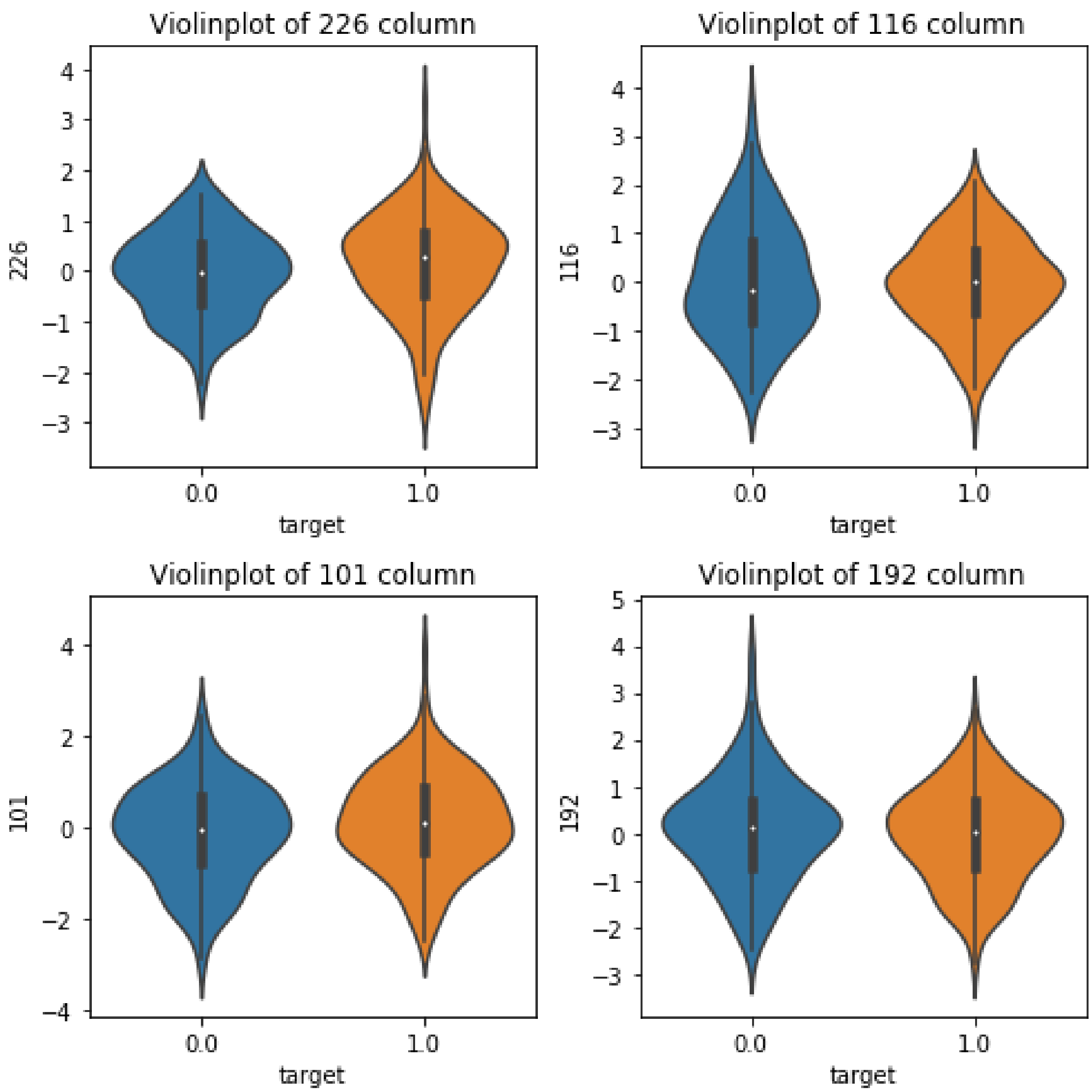


Observation: Seems like Median of both target class 0 and 1 have somewhat similar. Few of the points are outliers according to boxplot

4.4 Violin Plot

[12]:
```
# Seaborn violinplot (See Docs: https://seaborn.pydata.org/generated/seaborn.
↪violinplot.html?highlight=violinplot#seaborn.violinplot)

fig, axes = plt.subplots(2,2, figsize=(7,7))
r = np.random.randint(0,299,4,dtype=int)

sns.violinplot(data=df_train, x='target', y=str(r[0]), ax=axes[0,0])
axes[0,0].set_title('Violinplot of {} column'.format(r[0]))
```

```
sns.violinplot(data=df_train, x='target', y=str(r[1]), ax=axes[0,1])
axes[0,1].set_title('Violinplot of {} column'.format(r[1]))

sns.violinplot(data=df_train, x='target', y=str(r[2]), ax=axes[1,0])
axes[1,0].set_title('Violinplot of {} column'.format(r[2]))

sns.violinplot(data=df_train, x='target', y=str(r[3]), ax=axes[1,1])
axes[1,1].set_title('Violinplot of {} column'.format(r[3]))

fig.tight_layout()
plt.show()
```



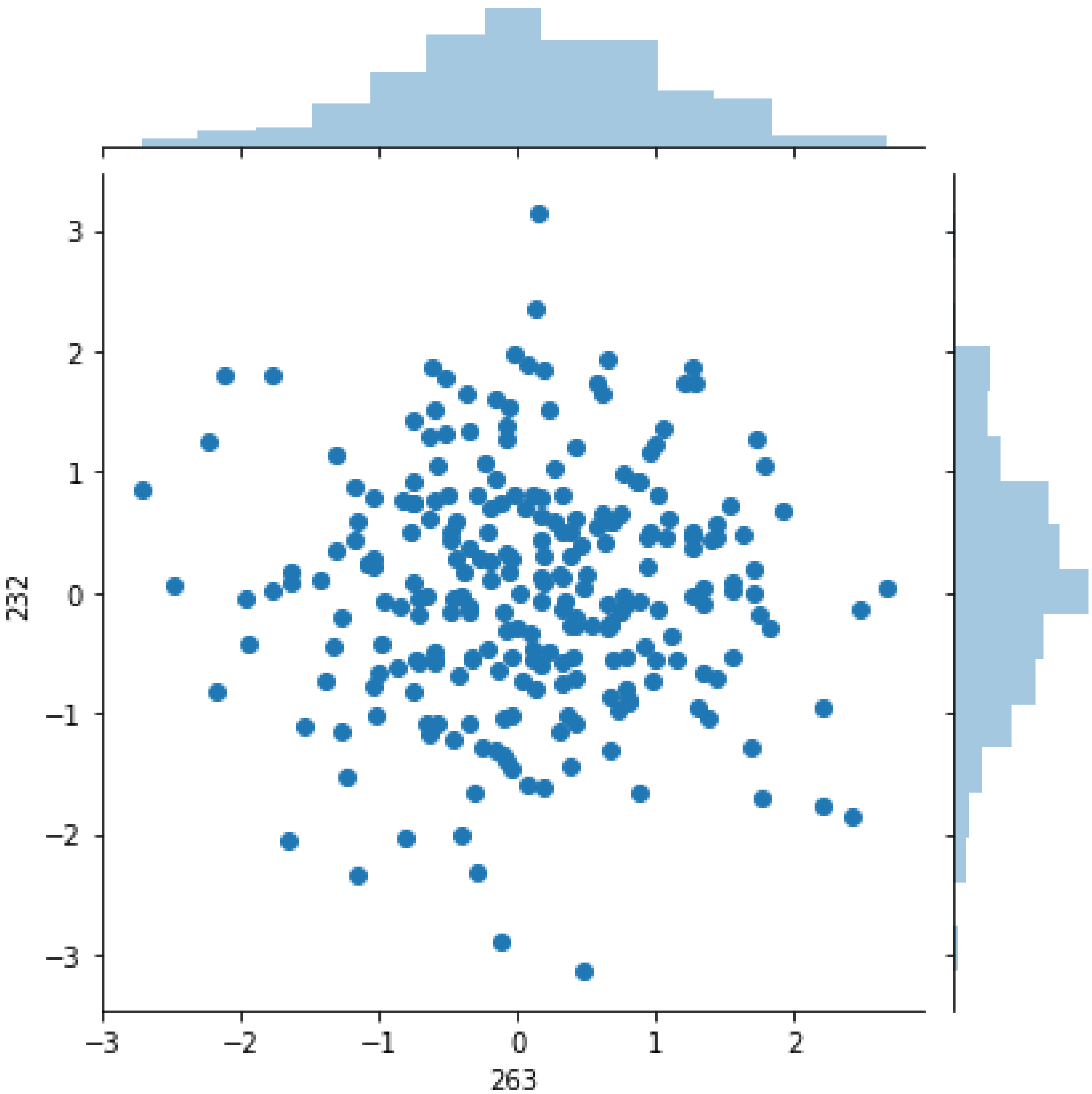
Observation: After running several times in this cell, we found that some features (or columns) are **dissimilar in distribution** on the basis of target class and **median** value are **mostly likely** to lie in the **same position** in violin plot.

4.5 ScatterPlot

[13]:
```
# Seaborn jointplot (See Docs: https://seaborn.pydata.org/generated/seaborn.
↪jointplot.html)

print('**********   Scatterplot between {} and {} columns **********'.
↪format(r[0],r[1]))

fig = plt.figure(figsize=(7,7))
r = np.random.randint(0,299,2,dtype=int)

sns.jointplot(data=df_train, x=str(r[0]), y=str(r[1]))

plt.show()
```

```
**********   Scatterplot between 226 and 116 columns **********
```

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



Observation: Can't interpret using scatter points

Let observe with the density of the region rather than points. One of the seaborn toolkit called **Contour**

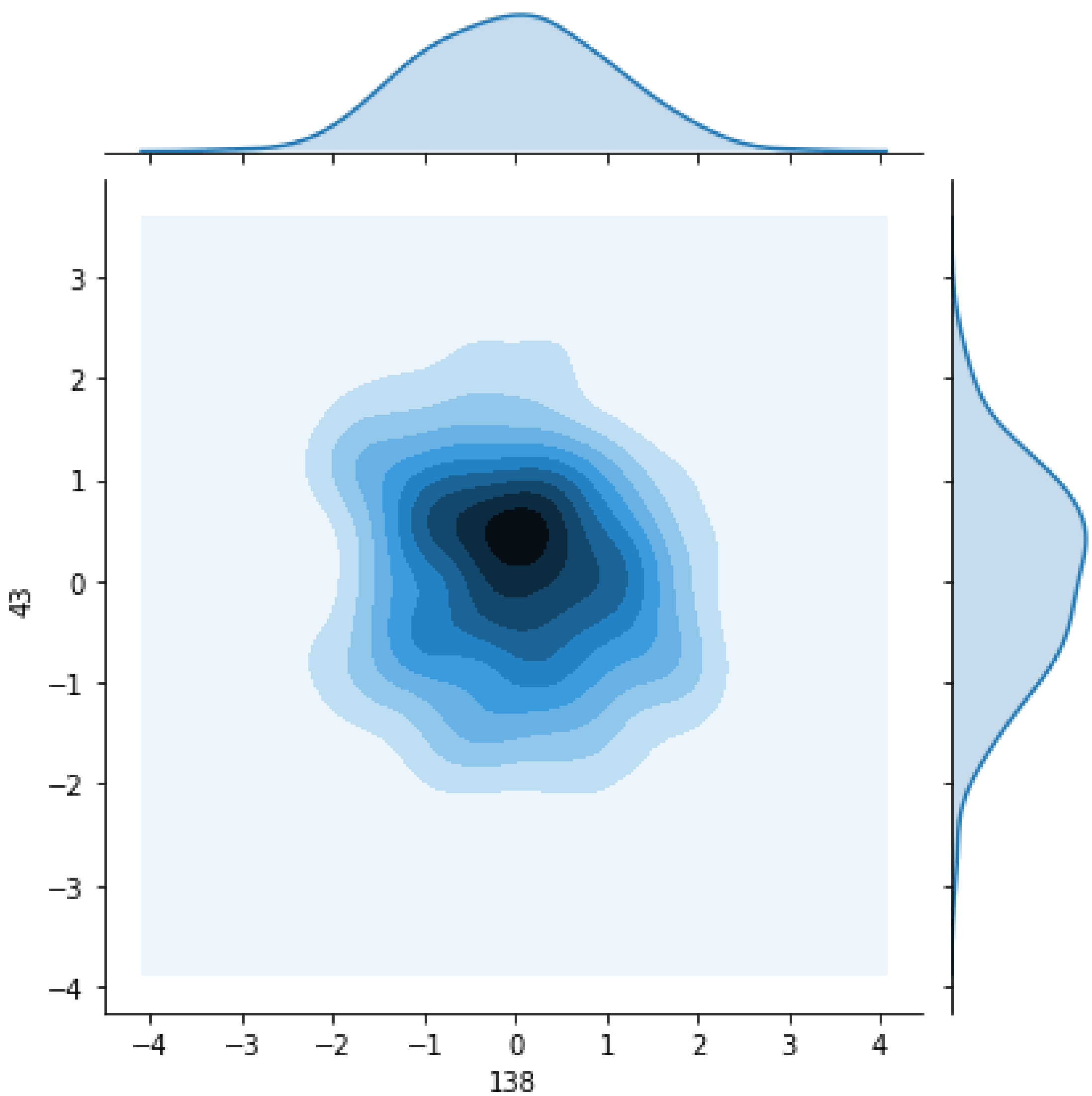
4.6 Contour

[14]:
```
# Nothing you havr to do. All you have to do is put parameter 'kind=kde'.␣
↪Everything is same

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

```
fig = plt.figure(figsize=(7,7))
r = np.random.randint(0,299,2,dtype=int)

sns.jointplot(data=df_train, x=str(r[0]), y=str(r[1]), kind='kde')

plt.show()
```

```
**********   Contour between 263 and 232 columns **********
```

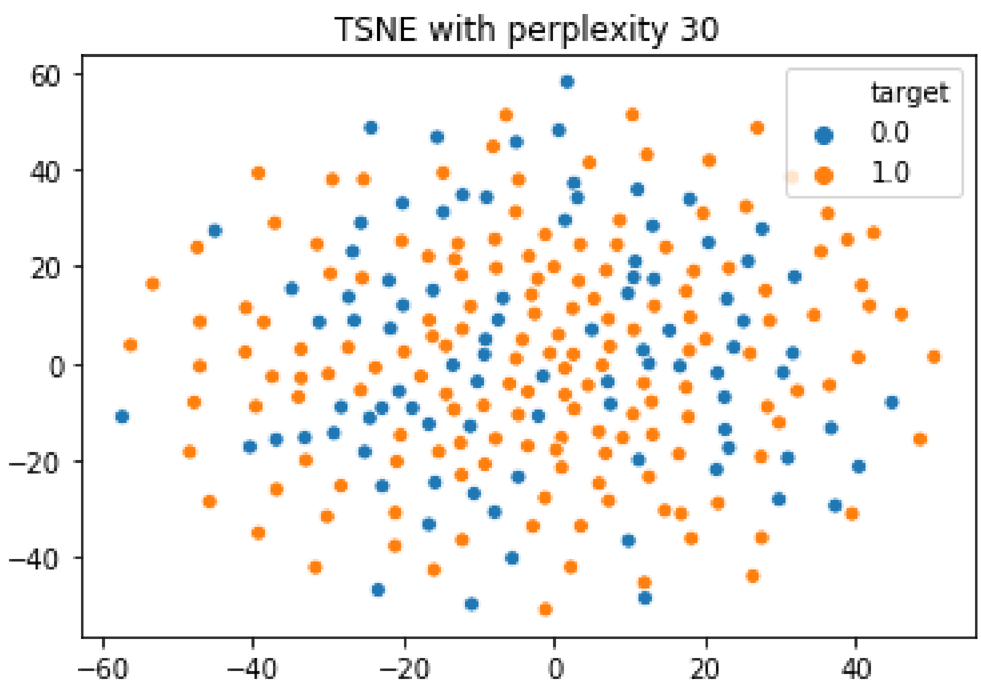
```
<Figure size 504x504 with 0 Axes>
```



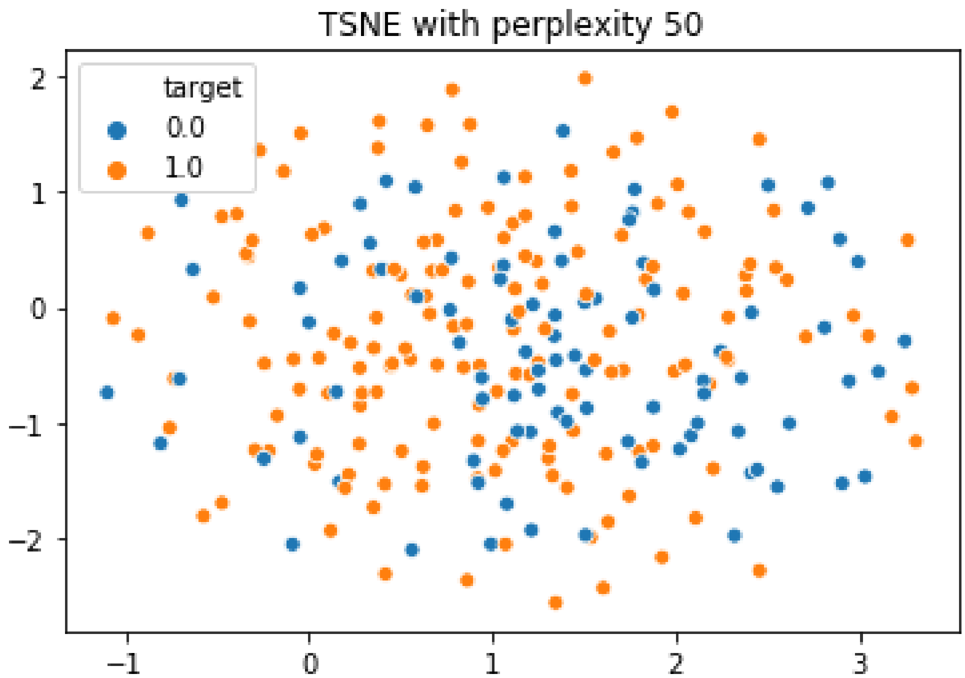
Observation: Most of the two features are having more density in range(-1,1) on both x-axis and y-axis

4.7 Visualize in 2D (Using TSNE)

[15]:
```
# TSNE (See Docs: https://scikit-learn.org/stable/modules/generated/sklearn.
↪manifold.TSNE.html)
# https://stackoverflow.com/questions/26558816/
↪matplotlib-scatter-plot-with-legend

train_tsne = TSNE(n_components=2).fit_transform(df_train.drop(['id','target'],␣
↪axis=1))
sns.scatterplot(train_tsne[:,0], train_tsne[:,1], hue=df_train['target'])
plt.title('TSNE with perplexity 30')
plt.legend()
plt.show()
```



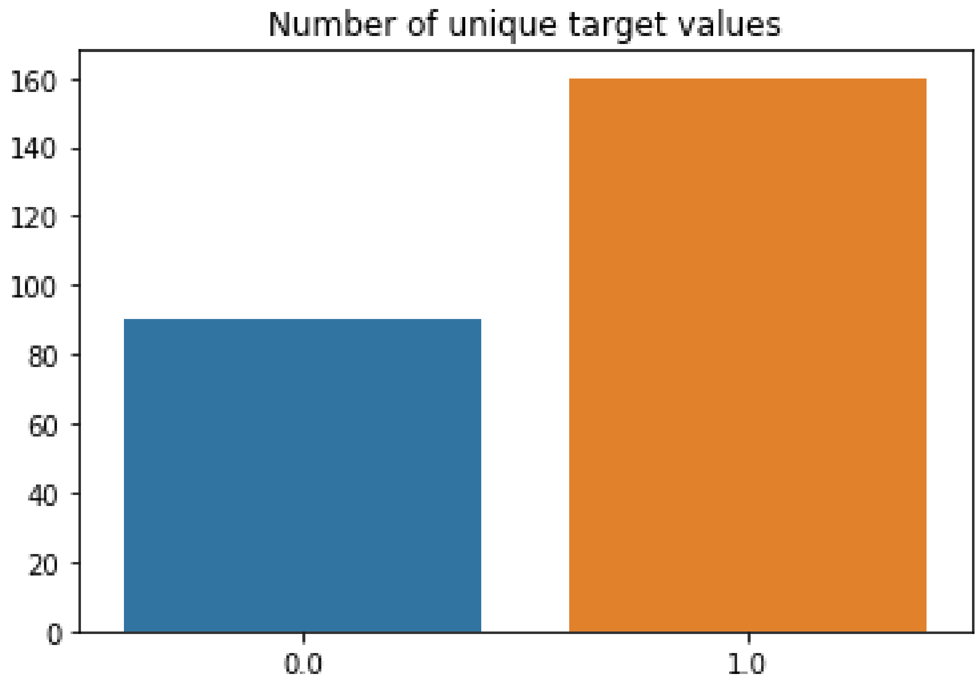
[16]:
```
train_tsne = TSNE(n_components=2, perplexity=50).fit_transform(df_train.
↪drop(['id','target'], axis=1))
sns.scatterplot(train_tsne[:,0], train_tsne[:,1], hue=df_train['target'])
plt.title('TSNE with perplexity 50')
plt.legend()
plt.show()
```



[17]:
```
# Scatter 3D (Ref: https://plotly.com/python/3d-scatter-plots/)

train_tsne = TSNE(n_components=3).fit_transform(df_train.drop(['id','target'],␣
↪axis=1))
df = pd.DataFrame(train_tsne, columns=['0','1','2'])
df['target'] = df_train['target']
fig = px.scatter_3d(df, x='0', y='1', z='2',color='target')
fig.show()
```

5 How balance dataset is? Imbalance or Balance?

[18]:
```
sns.barplot(x=df_train['target'].value_counts().index, y=df_train['target'].
↪value_counts().values)
plt.title('Number of unique target values')
plt.show()
```



[19]:
```
total_num = df_train.shape[0]
count_1, count_0 = df_train['target'].value_counts().values
print('Fraction of dataset contain total number of 0s:',(count_0/
 ↪total_num)*100,'%')
print('Fraction of dataset contain total number of 1s:',(count_1/
 ↪total_num)*100,'%')
```

```
Fraction of dataset contain total number of 0s: 36.0 %
Fraction of dataset contain total number of 1s: 64.0 %
```

6 Summary

1. From PDF observation, Most of the feature follows gaussian distribution and their comparison with the other features are quite similar having lying on same standard deviation.
2. From CDF observation, we observe that every features have even same (or most similar) slope.
3. From Boxplot observation, we didn't find any separable difference in between them. Few of them have outliers
4. From Violinplot observation, some of the feature on the basis of target value are dissimilar for follow gaussian different manner but median look like to lying in the same poosition
5. From Scatterplot or Contour, we observe that most of the features between them lie in the range (-1,1)

6. Most Aspect of this dataset: **Imbalance dataset** (Not highly but decent)

```
[ ]:
```