Don't Overfit

Problem Statement

- Donot Overfit 2 is a unique problem statement where we are provided with only 250 training samples and 19750 test samples.
- The Objective of the problem is not to overfit with this train data and generalize well with our test data samples.
- The data set consists of 300 continuous random variables each standardized with mean centered to zero and variance 1.

Performance Metrics Used

- The Problem uses ROC AUC SCORE as the metric to measure the model performance

In [1]:

```
import numpy as np
import pandas as pd
from scipy import stats
import sklearn
import warnings
warnings.filterwarnings('ignore')
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import RepeatedStratifiedKFold
import seaborn as sns
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.calibration import CalibratedClassifierCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler
pd.options.mode.chained_assignment = None   # default='warn'
from mlxtend.classifier import StackingClassifier
from sklearn.linear_model import Lasso
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from scipy.stats import randint as sp_randint
from scipy.stats import uniform
from scipy import stats
import xgboost as xgb
```

Getting the Data into Data Frame

In [2]:

```
train_data=pd.read_csv("train.csv")
test_data=pd.read_csv("test.csv")
```

Exploratory Data Analysis

In [6]:

```
train_data.describe()
```

Out[6]:

	id	target	0	1	2	3	4
count	250.000000	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	0.001588
std	72.312977	0.480963	0.998354	1.009314	1.021709	1.011751	1.035411
min	0.000000	0.000000	-2.319000	-2.931000	-2.477000	-2.359000	-2.566000
25%	62.250000	0.000000	-0.644750	-0.739750	-0.425250	-0.686500	-0.659000
50%	124.500000	1.000000	-0.015500	0.057000	0.184000	-0.016500	-0.023000
75%	186.750000	1.000000	0.677000	0.620750	0.805000	0.720000	0.735000
max	249.000000	1.000000	2.567000	2.419000	3.392000	2.771000	2.901000

8 rows × 302 columns

In [3]:

```
train_data.head(3)
```

Out[3]:

	id	target	0	1	2	3	4	5	6	7	...	290	291
0	0	1.0	-0.098	2.165	0.681	-0.614	1.309	-0.455	-0.236	0.276	...	0.867	1.347
1	1	0.0	1.081	-0.973	-0.383	0.326	-0.428	0.317	1.172	0.352	...	-0.165	-1.695
2	2	1.0	-0.523	-0.089	-0.348	0.148	-0.022	0.404	-0.023	-0.172	...	0.013	0.263

3 rows × 302 columns

In [4]:

```
train_data.columns
```

Out[4]:

```
Index(['id', 'target', '0', '1', '2', '3', '4', '5', '6', '7',
       ...
       '290', '291', '292', '293', '294', '295', '296', '297', '298', '29
9'],
      dtype='object', length=302)
```

In [5]:

```
test_data.head(2)
```

Out[5]:

	id	0	1	2	3	4	5	6	7	8	...	290	291
0	250	0.500	-1.033	-1.595	0.309	-0.714	0.502	0.535	-0.129	-0.687	...	-0.088	-2.628
1	251	0.776	0.914	-0.494	1.347	-0.867	0.480	0.578	-0.313	0.203	...	-0.683	-0.066

2 rows × 301 columns

Columns Description

- ID- Unique No for Each Datapoint
- Target-Independent Variable
- 0-299-Features having a mean close to 0 and standard deviation 1.

Check Null Values

Q. Is there any null values in this dataset?If yes then how many by count and percentage?

In [7]:

```
print((train_data.isna().sum()/train_data.shape[0])*100)
```

```
id        0.0
target    0.0
0         0.0
1         0.0
2         0.0
         ...
295       0.0
296       0.0
297       0.0
298       0.0
299       0.0
Length: 302, dtype: float64
```

In [8]:

```
print((test_data.isna().sum()/test_data.shape[0])*100)
```

```
id     0.0
0      0.0
1      0.0
2      0.0
3      0.0
      ...
295    0.0
296    0.0
297    0.0
298    0.0
299    0.0
Length: 301, dtype: float64
```

- Seems like we dont have null values in our data

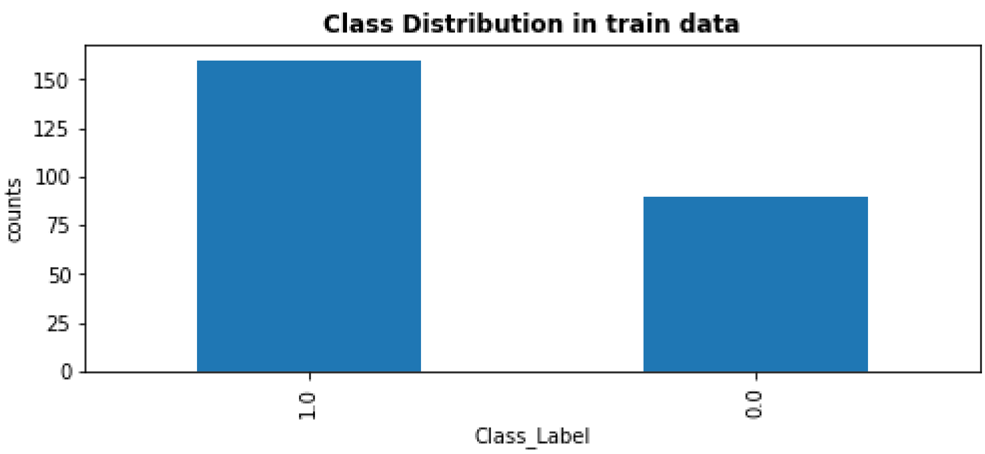
Distribution of Target variable of Train Data Points

In [9]:

```
plt.figure(figsize=(8,3))
ax =train_data.target.value_counts().plot(kind='bar')
plt.title('Class Distribution in train data', weight='bold')
plt.xlabel('Class_Label')
plt.ylabel('counts')
```

Out[9]:

```
Text(0, 0.5, 'counts')
```



- **We have 160 data points belonging to Class 1**
- **We have 90 data points belonging to Class 2**
- **The Dataset is imbalanced**

In [10]:

```
train_data['target'].value_counts()
```

Out[10]:

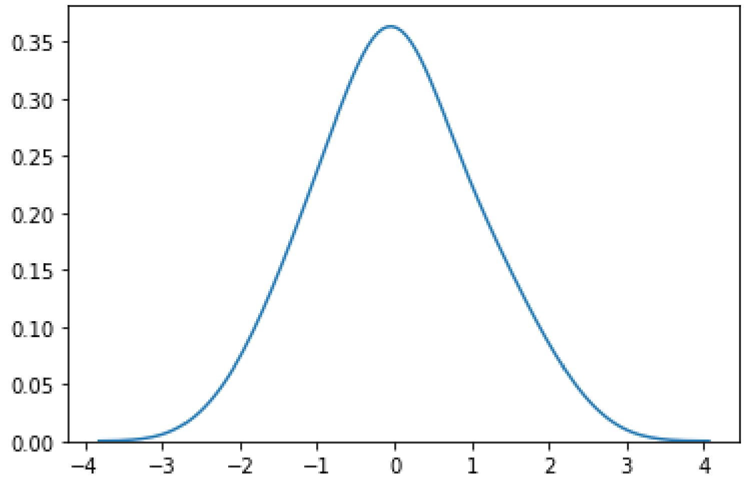
```
1.0    160
0.0     90
Name: target, dtype: int64
```

KS-TEST TRAIN DATA OF SAMPLE 250 PTS

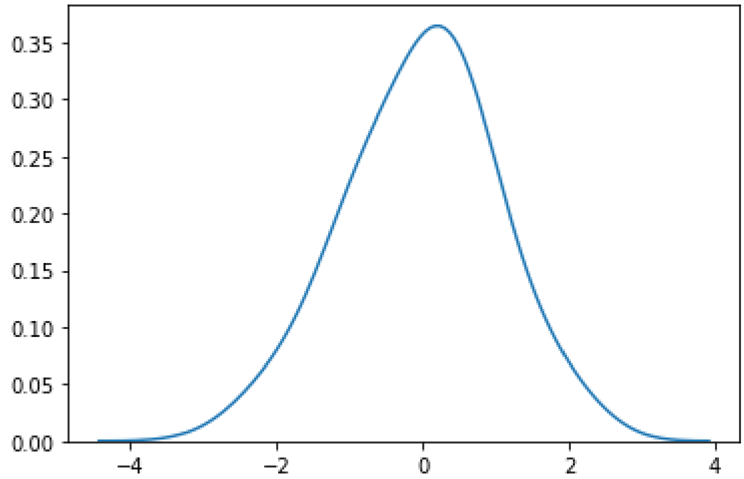
In [11]:

```
for i in range(0,25):
    sns.kdeplot(np.array(train_data[str(i)]), bw=0.5)
    print(stats.kstest(train_data[str(i)],"norm"))
    plt.show()
```

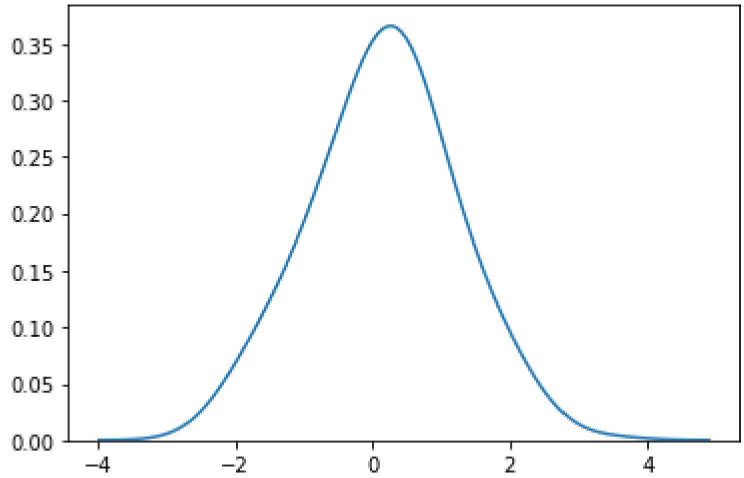
```
KstestResult(statistic=0.03171261921735258, pvalue=0.9630294012606941)
```



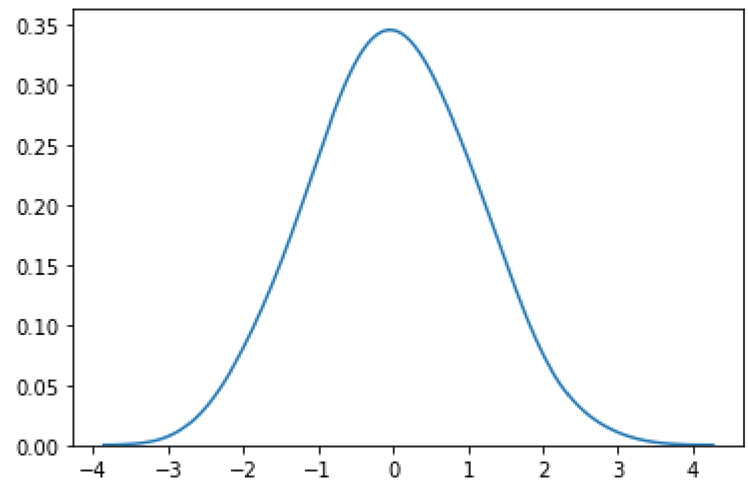
```
KstestResult(statistic=0.04136319722522663, pvalue=0.7857765399000378)
```



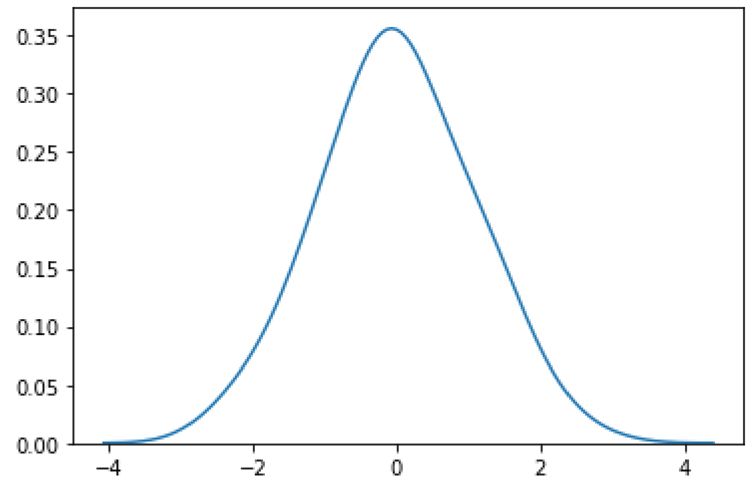
```
KstestResult(statistic=0.10678169211374755, pvalue=0.006165025199299171)
```



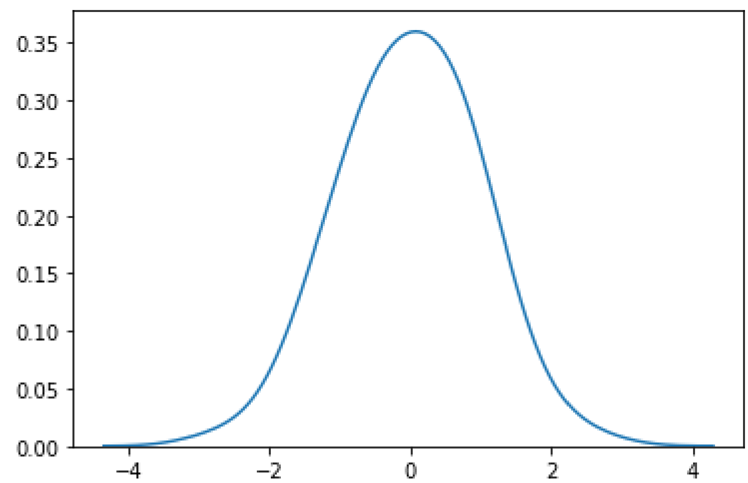
```
KstestResult(statistic=0.030792214968717868, pvalue=0.9717286095262895)
```



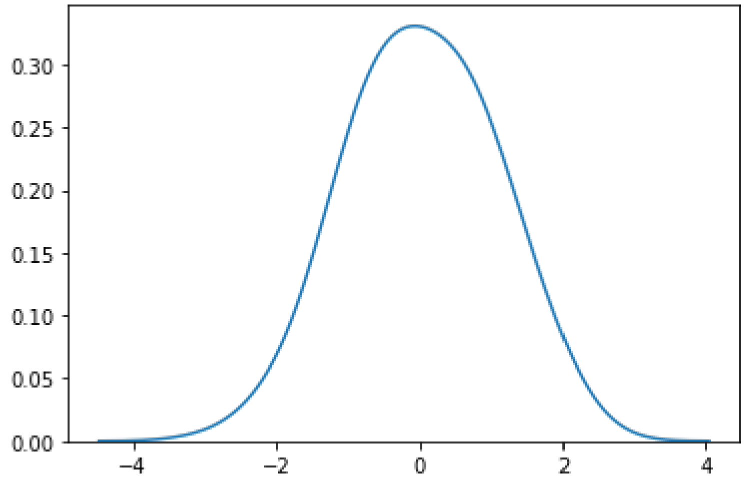
```
KstestResult(statistic=0.038500486757252816, pvalue=0.8524941994125361)
```



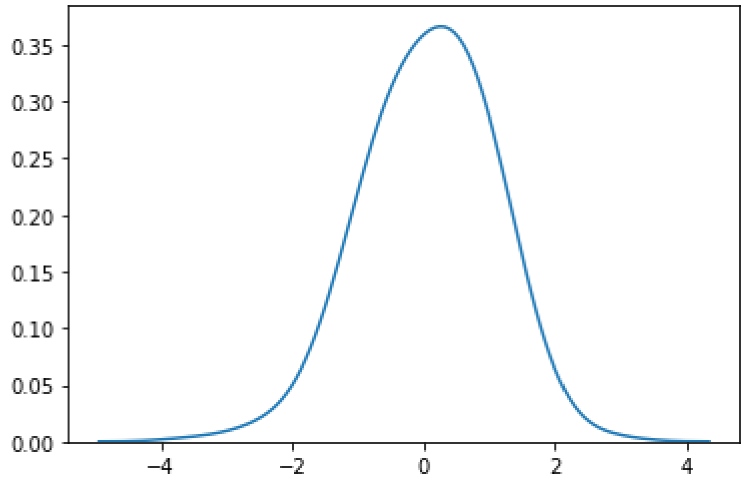
```
KstestResult(statistic=0.040963628200492375, pvalue=0.7955785542603029)
```



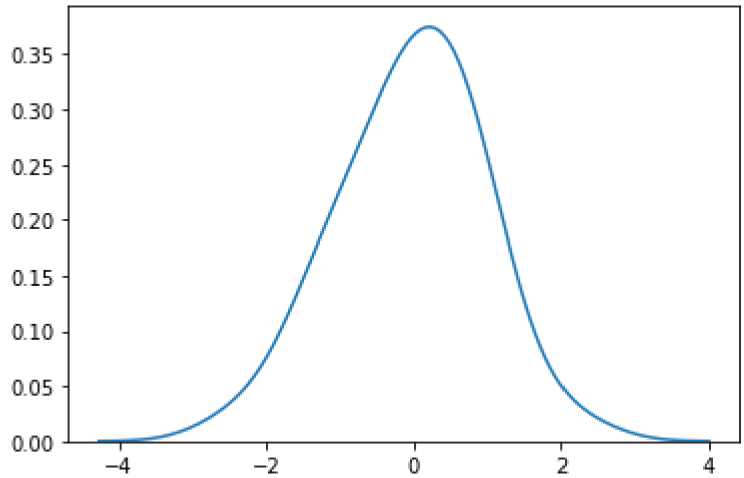
```
KstestResult(statistic=0.053944034605123536, pvalue=0.4504729471406711)
```



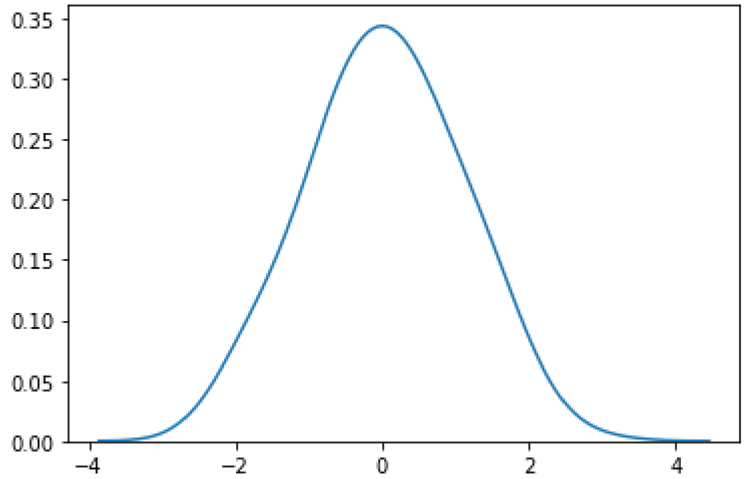
KstestResult(statistic=0.07831497553761624, pvalue=0.08828235812070054)



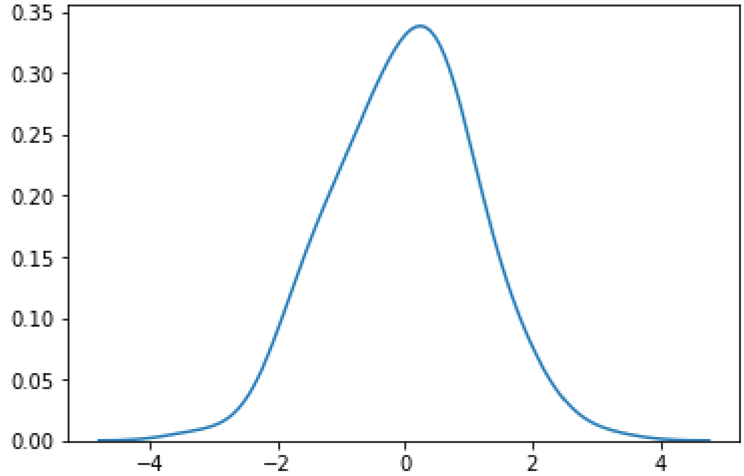
KstestResult(statistic=0.05144623171344842, pvalue=0.5147258593904032)



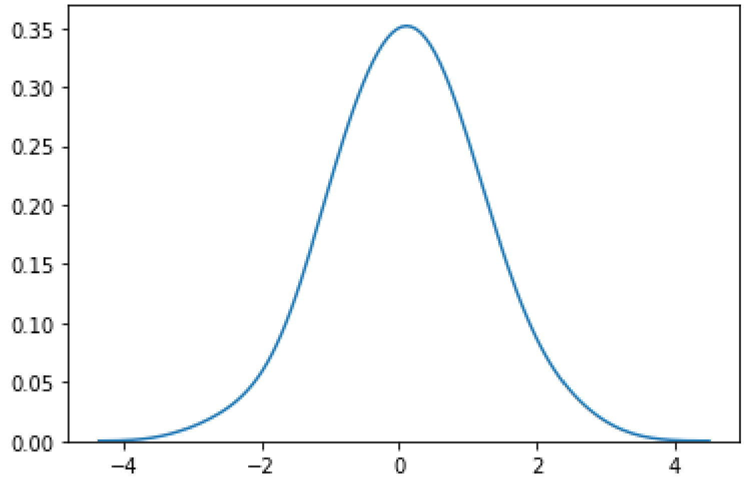
KstestResult(statistic=0.0393009539263684, pvalue=0.8347394170653292)



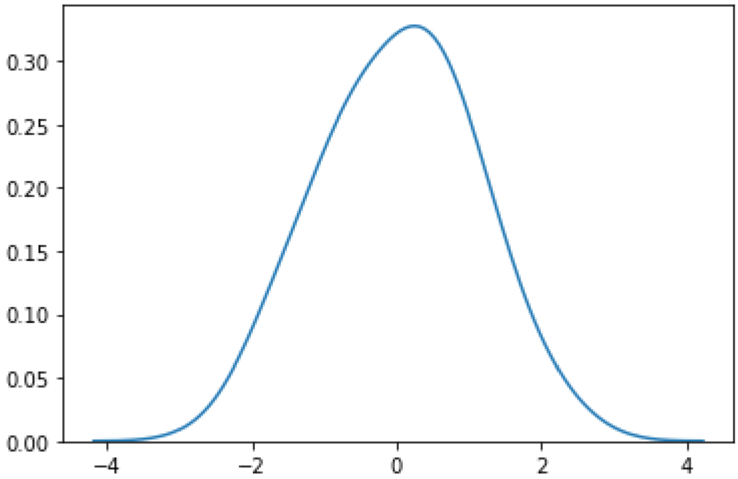
```
KstestResult(statistic=0.05149200856259806, pvalue=0.5134984920818352)
```



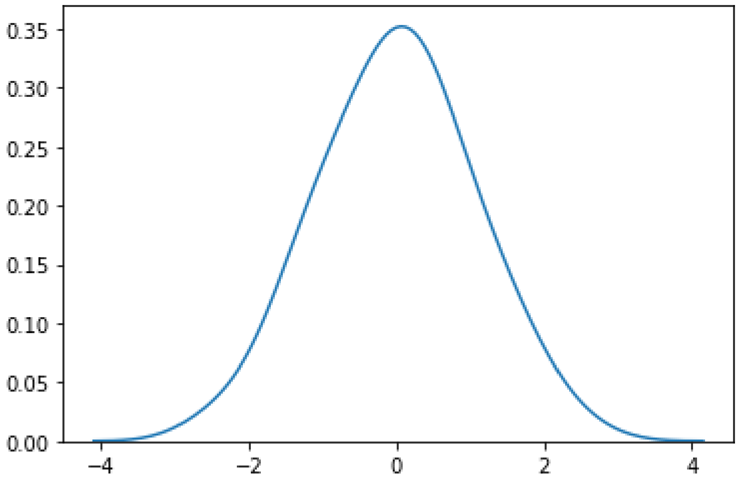
```
KstestResult(statistic=0.061096941243909686, pvalue=0.29703084400292784)
```



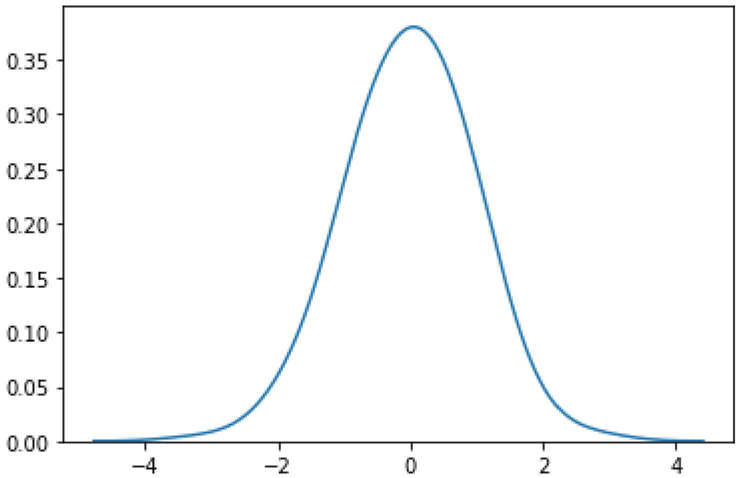
```
KstestResult(statistic=0.04918316271726686, pvalue=0.5776907366915892)
```



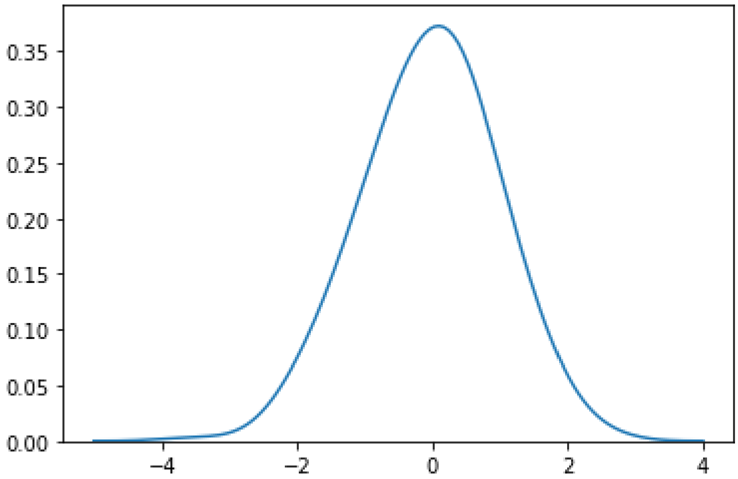

```
KstestResult(statistic=0.035300953926368395, pvalue=0.914390751844368)
```



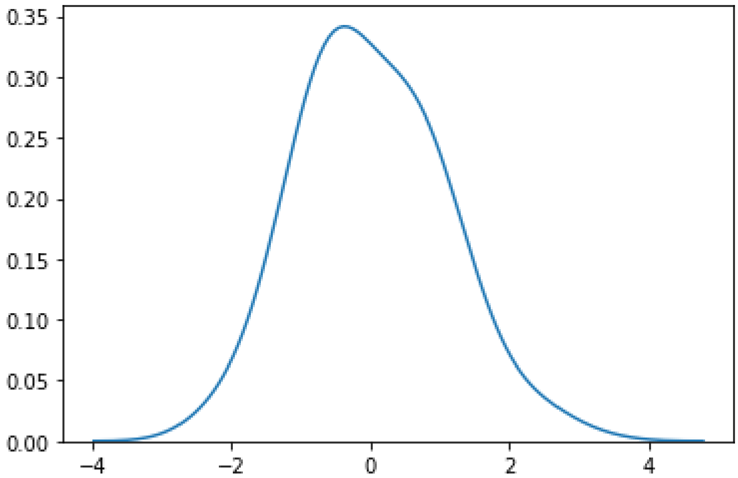

```
KstestResult(statistic=0.05159775312640458, pvalue=0.5106703473467841)
```



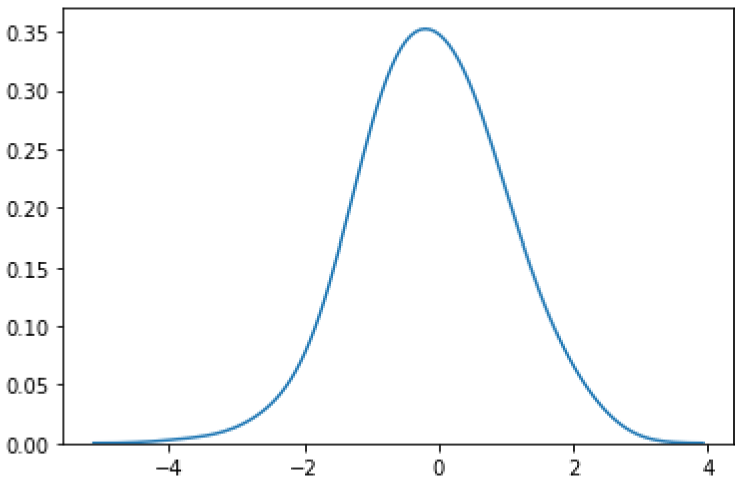

```
KstestResult(statistic=0.04670447225369956, pvalue=0.6516761026285204)
```



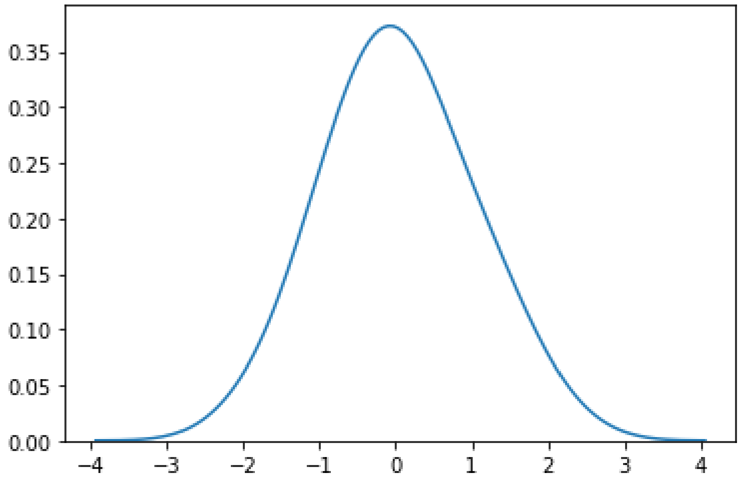

```
KstestResult(statistic=0.0671874568912148, pvalue=0.2000831519397673)
```



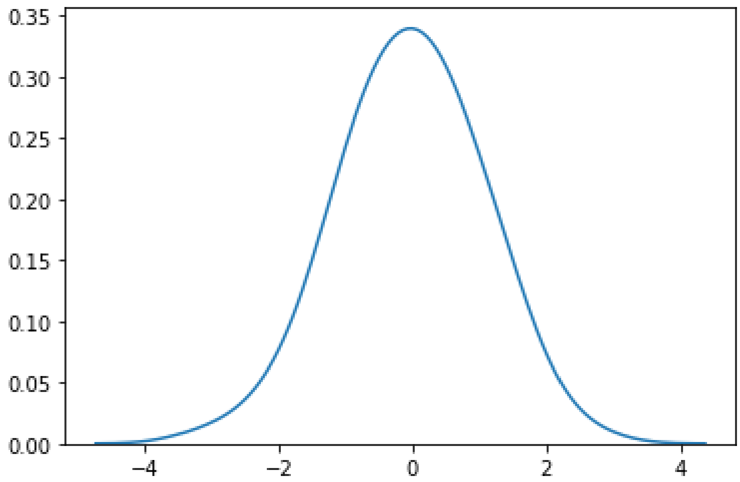

```
KstestResult(statistic=0.06755946289143283, pvalue=0.1950766170905474)
```



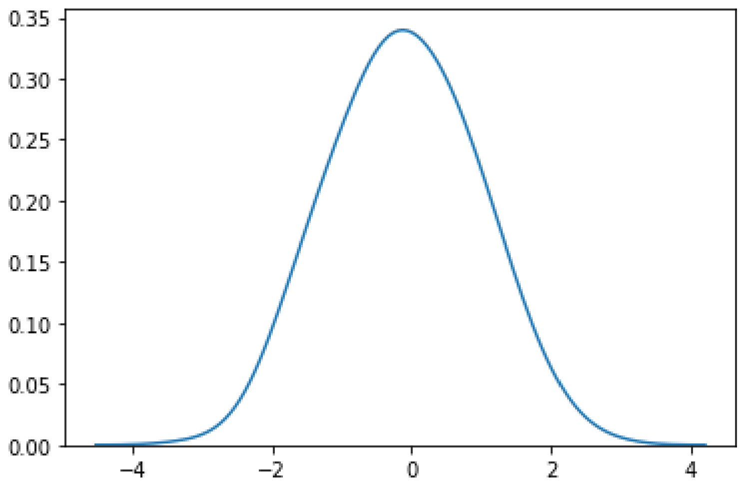

```
KstestResult(statistic=0.03918239942009802, pvalue=0.8374181676483494)
```



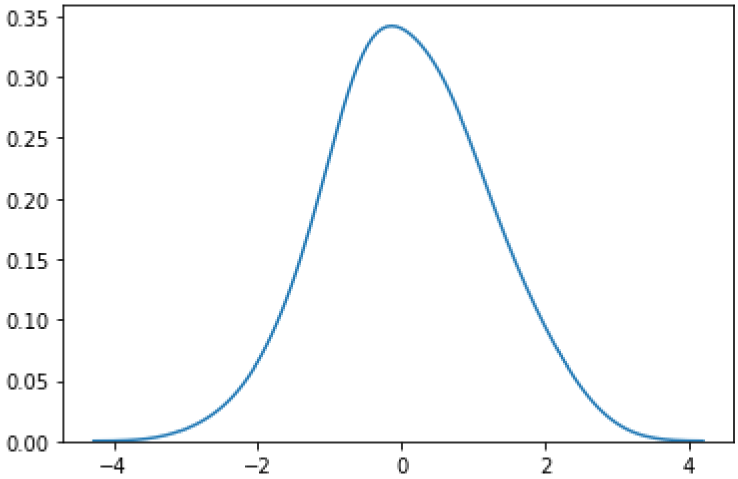
KstestResult(statistic=0.035467031395539805, pvalue=0.9115799161732249)



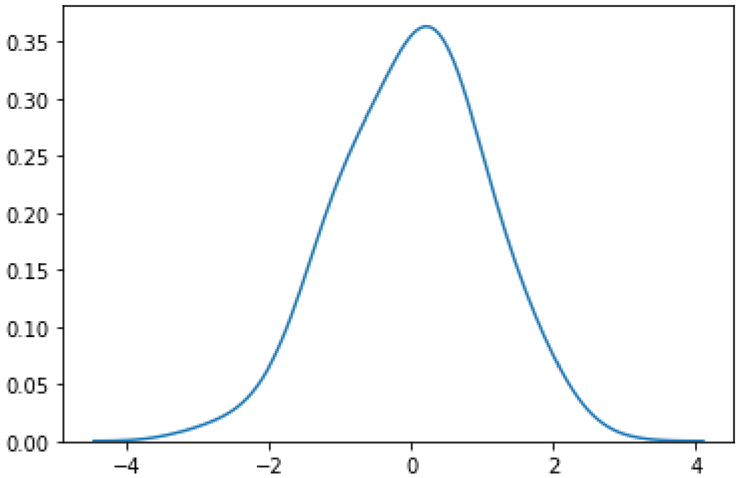
KstestResult(statistic=0.06915888374918011, pvalue=0.17466266738156241)



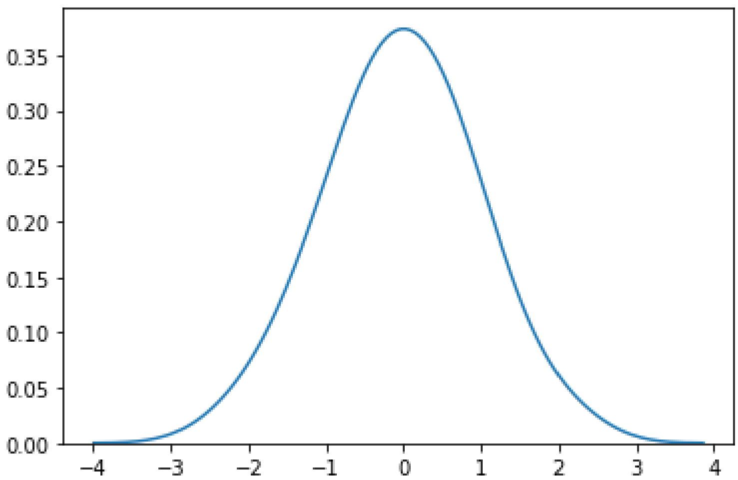
KstestResult(statistic=0.0501110886094247, pvalue=0.551333849953259)



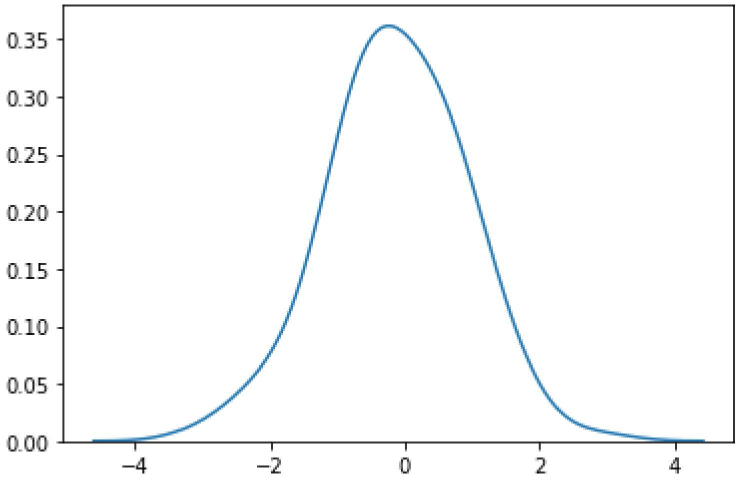

```
KstestResult(statistic=0.05690330975007296, pvalue=0.38154472397148226)
```




```
KstestResult(statistic=0.037660945578836036, pvalue=0.8702211833921591)
```




```
KstestResult(statistic=0.08189582753034802, pvalue=0.06607087878107515)
```



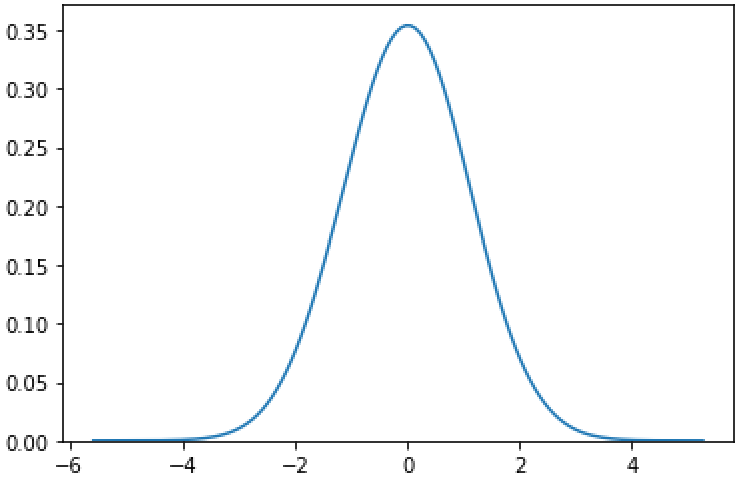
- **By seeing the above plots of KS-Test we can see that most of our plots follow a gaussian like curve (Bell Curve) with a high p value.We can conclude saying that most of our features follow a Gaussian Distribution with mean close to 0 and std-dev close to 1.Most importantly this plot is just over a sample of 250 pts.As no of pts increases,This will follow Gaussian. Let's ensure this by plotting a KS PLOT FOR THE TEST SET Feature which has 19750 Points.**

Test-Data

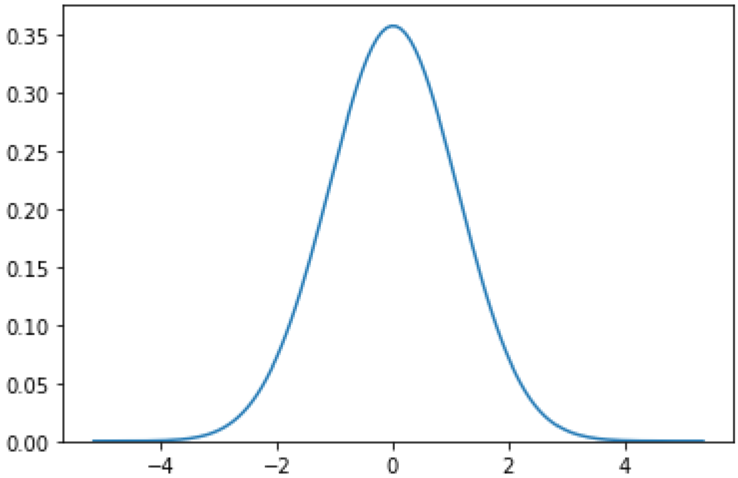
In [12]:

```
for i in range(0,2):
    sns.kdeplot(np.array(test_data[str(i)]), bw=0.5)
    print(stats.kstest(test_data[str(i)],"norm"))
    plt.show()
```

```
KstestResult(statistic=0.008675831943463025, pvalue=0.10226846130903085)
```




```
KstestResult(statistic=0.0046406094845576895, pvalue=0.7886496588520177)
```



EDA SUMMARY

- **We only have 250 data points as our whole data to train our model.**
- **We have a total of 300 features and by EDA we get to know that each feature follow a gaussian disb with mean close to 0 and std-dev very close to 1**

Simple First-Cut Solution using Logistic Regression and RepeatedStratifiedKFold

Data Split

In [13]:

```
train_data["target"]=train_data["target"].astype(int)
train_data=train_data.drop("id",axis=1)
X=train_data.drop("target",axis=1)
Y=train_data['target']
test=test_data.drop("id",axis=1)
```

Random Search and Stratified-K-Fold Strategy

In [14]:

```
def rskf_func(n_splits,n_repeats): #Creating a function for repeated stratified K-Fold Object
    rskf_var=RepeatedStratifiedKFold(n_splits=n_splits,n_repeats=n_repeats)
    return rskf_var
```

In [18]:

```
def final_submission_csv(final_prediction,name): #Getting csv for kaggle submission
    sub_df=pd.read_csv("sample_submission.csv")
    final_df=pd.concat([sub_df["id"],pd.DataFrame(final_prediction)],axis=1)
    final_df.columns=["id","target"]
    final_df.to_csv(name + ".csv",index=False)
```

In [17]:

```
test_val=np.zeros(len(test))
cnt=0
rskf=rskf_func(20,20)
for train_index,valid_index in rskf.split(X,Y):
    #Gets 19 chunks of splitted data for train and 1 chunk for the validation out of 20
chunks
    #Each chunk can be used for validation once.Hence 20 iterations for 20 chunks
    #And we are repeating the process for 20 times.So 400 iterations in total
    X_train,X_valid=X.loc[train_index],X.loc[valid_index]
    Y_train,Y_valid=Y.loc[train_index],Y.loc[valid_index]
    clf=SGDClassifier(loss='log',class_weight='balanced',n_jobs=-1)
    param={"penalty":["l1","l2","elasticnet"],
        "alpha":np.arange(0.1,0.9,0.01),
        "l1_ratio":np.arange(0.1,0.9,0.05)
      }
    grid_model=RandomizedSearchCV(clf,param,cv=15,scoring='roc_auc',n_jobs=-1,verbose=0
)
    grid_model.fit(X_train,Y_train)
    clf_calib=CalibratedClassifierCV(grid_model.best_estimator_,cv=20,method='sigmoid')
    clf_calib.fit(X_train,Y_train)
    valid_roc=roc_auc_score(Y_valid.values,clf_calib.predict_proba(X_valid)[:,1])
    if( valid_roc > 0.8):
        print("<---Model ok")
        test_val+=clf_calib.predict_proba(test)[:,1]
        cnt+=1
    else:
        print("Skipping Model for this Iteration")

final_prediction=test_val* (1./cnt)
```

```
Skipping Model for this Iteration
<---Model ok
Skipping Model for this Iteration
<---Model ok
<---Model ok
Skipping Model for this Iteration
Skipping Model for this Iteration
Skipping Model for this Iteration
Skipping Model for this Iteration
<---Model ok
Skipping Model for this Iteration
Skipping Model for this Iteration
<---Model ok
<---Model ok
Skipping Model for this Iteration
<---Model ok
<---Model ok
Skipping Model for this Iteration
Skipping Model for this Iteration
<---Model ok
Skipping Model for this Iteration
<---Model ok
Skipping Model for this Iteration
<---Model ok
Skipping Model for this Iteration
Skipping Model for this Iteration
Skipping Model for this Iteration
<---Model ok
Skipping Model for this Iteration
Skipping Model for this Iteration
Skipping Model for this Iteration
<---Model ok
Skipping Model for this Iteration
<---Model ok
```

In [19]:

```
final_submission_csv(final_prediction,"simple_logistic_regression")
```

Score

- **Simple Logistic Regression with Repeated K-Fold Validation gave us a score of 0.804 in Private LeaderBoard and 0.831 Public Leaderboard**

Solution-2 Stacking Classifier and Rigorous Feature Selection

In [21]:

```
iter_=1
test_prediction=np.zeros(len(test))
#'alpha' : [0.022, 0.021, 0.02, 0.019, 0.023, 0.024, 0.025, 0.026, 0.027, 0.029, 0.03
1],
#'tol'   : [0.0013, 0.0014, 0.001, 0.0015, 0.0011, 0.0012, 0.0016, 0.0017]
rskf=rskf_func(30,25)
for train_index,validation_index in rskf.split(X,Y):
    print("Iter:",iter_)
```

```
    X_train,X_CV=X.loc[train_index],X.loc[validation_index]
    Y_train,Y_CV=Y.loc[train_index],Y.loc[validation_index]
    feature_selector_model=Lasso()
    grid_par={"alpha":[0.022, 0.021, 0.02, 0.019, 0.023, 0.024, 0.025, 0.026, 0.027, 0.
029, 0.031]}
    gridcv=RandomizedSearchCV(feature_selector_model,grid_par,cv=10,scoring='roc_auc',n
_jobs=-1,verbose=0)
    gridcv.fit(X_train,Y_train)
    sfs = SFS(gridcv.best_estimator_,k_features=(10, 20),forward=True,floating=True,sco
ring='roc_auc',verbose=0,n_jobs=-1)
    sfs.fit(X_train,Y_train)
    X_train_imp=sfs.transform(X_train)
    X_Cv_imp=sfs.transform(X_CV)
    test_imp=sfs.transform(test)
    # Initializing models
    clf1=xgb.XGBClassifier(scale_pos_weight=0.5625)
    clf2 = GaussianNB(priors=[0.5,0.5])
    clf3 = Lasso()
    clf4 = SGDClassifier(loss='hinge',class_weight='balanced')
    lr = LogisticRegression(class_weight='balanced',solver='liblinear')
    sclf = StackingClassifier(classifiers=[clf1, clf2, clf3,clf4], meta_classifier=lr)
    sclf.fit(X_train_imp,Y_train)
# Estimate feature importance and time the whole process
    params = {
              'xgbclassifier__learning_rate':stats.uniform(0.01,0.3),
              'xgbclassifier__n_estimators':sp_randint(100,1000),
              'xgbclassifier__max_depth':sp_randint(1,10),
              'xgbclassifier__min_child_weight':sp_randint(1,8),
              'xgbclassifier__gamma':stats.uniform(0,0.02),
              'xgbclassifier__subsample':stats.uniform(0.6,0.3),
              'xgbclassifier__reg_alpha':sp_randint(0,200),
              'xgbclassifier__reg_lambda':stats.uniform(0,200),
              'xgbclassifier__colsample_bytree':stats.uniform(0.6,0.3),
              'lasso__alpha':  [0.022, 0.021, 0.02, 0.019, 0.023, 0.024, 0.025, 0.026,
0.027, 0.029, 0.031],
              'lasso__tol' :   [0.0013, 0.0014, 0.001, 0.0015, 0.0011, 0.0012, 0.0016,
0.0017],
              'sgdclassifier__penalty': ['l1','l2','elasticnet'],
              'sgdclassifier__alpha' : np.arange(0.01,1,0.001),
              'sgdclassifier__l1_ratio' : np.arange(0.1,0.9,0.05),
              'meta_classifier__penalty': ['l1','l2'],
              'meta_classifier__C': np.arange(0.1,1,0.01)
    }

    grid =RandomizedSearchCV(estimator=sclf,param_distributions=params,cv=20,scoring='r
oc_auc',n_jobs=-1,verbose=1)
    grid.fit(X_train_imp, Y_train)


    y_cv_pred=grid.best_estimator_.predict_proba(X_Cv_imp)[:,1]
    roc_cv=roc_auc_score(Y_CV.values,y_cv_pred)

    if(roc_cv > 0.85):
        print("<--------Model is performing well------->")
        test_prediction+=grid.best_estimator_.predict_proba(test_imp)[:,1]
        cnt+=1
    else:
        print("<------The Model is not performing as expected in this iteration--------
->")
    iter_+=1
final_stacked_prediction=test_prediction * (1./cnt)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:    1.1s
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed:    3.5s finished


<------The Model is not performing as expected in this iteration--------->
Iter: 750
Fitting 20 folds for each of 10 candidates, totalling 200 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  52 tasks      | elapsed:    0.9s

<--------Model is performing well------->

[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed:    2.5s finished
```

In [23]:

```
final_submission_csv(final_stacked_prediction,"stack_pred_final")
```

Final Score

- **With Stacked Model we get a Private Score of 0.825 and Public Score of 0.838**

****

Conclusion

- **Simple Logistic Regression with K-Fold Cross Validation we got 80 % ROC**
- **More Complex Stacked Classifer and Feature Selection Techniques help us achieve 83% ROC**
- **Without LB Probing we could get a good classifier which separates the data well in Private LB**

References

- **https://www.kaggle.com/featureblind/robust-lasso-patches-with-rfe-gs (https://www.kaggle.com/featureblind/robust-lasso-patches-with-rfe-gs)**
- **https://www.kaggle.com/rafjaa/dealing-with-very-small-datasets (https://www.kaggle.com/rafjaa/dealing-with-very-small-datasets)**
- **https://www.kaggle.com/iavinas/simple-short-solution-don-t-overfit-0-848 (https://www.kaggle.com/iavinas/simple-short-solution-don-t-overfit-0-848)**
- **https://www.appliedaicourse.com/lecture/11/applied-machine-learning-online-course/3096/stacking-models/4/module-4-machine-learning-ii-supervised-learning-models (https://www.appliedaicourse.com/lecture/11/applied-machine-learning-online-course/3096/stacking-models/4/module-4-machine-learning-ii-supervised-learning-models)**