Santander Customer Transaction Prediction

Problem Statement:-

In this challenge, we need to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

In [2]:
```
#Loading Libraries:-

import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import lightgbm as lgb
import eli5
from sklearn.model_selection import train_test_split,cross_val_predict,cross_val
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,roc_auc_score,roc_curve,classificat
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
random_state=42
np.random.seed(random_state)
```

In [4]:
```
#### set working directory###
os.chdir("C:/Users/Poo/Documents/edWisor/Project 02")
os.getcwd()
```

Out[4]:
```
'C:\\Users\\Poo\\Documents\\edWisor\\Project 02'
```

In [5]:
```
## load CSV file##
train=pd.read_csv("train.csv")

#to see all the coloums
pd.options.display.max_columns = None
```

In [6]:
```
#see top 5 observation
train.head()
```

Out[6]:

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	v
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	-4.9200	5.
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	3.1468	8.
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	-4.9193	5.
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	-5.8609	8.
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	6.2654	7.

In [7]:
```
#Shape of the dataset
train.shape
```

Out[7]: (200000, 202)

In [8]:
```
## check data type##
train.dtypes
```

Out[8]:
```
ID_code     object
target       int64
var_0      float64
var_1      float64
var_2      float64
var_3      float64
var_4      float64
var_5      float64
var_6      float64
var_7      float64
var_8      float64
var_9      float64
var_10     float64
var_11     float64
var_12     float64
var_13     float64
var_14     float64
var_15     float64
var_16     float64
var_17     float64
var_18     float64
var_19     float64
var_20     float64
var_21     float64
var_22     float64
var_23     float64
var_24     float64
var_25     float64
var_26     float64
var_27     float64
             ...
var_170    float64
var_171    float64
var_172    float64
var_173    float64
var_174    float64
var_175    float64
var_176    float64
var_177    float64
var_178    float64
var_179    float64
var_180    float64
var_181    float64
var_182    float64
var_183    float64
var_184    float64
var_185    float64
var_186    float64
var_187    float64
var_188    float64
var_189    float64
var_190    float64
var_191    float64
```

```
var_192    float64
var_193    float64
var_194    float64
var_195    float64
var_196    float64
var_197    float64
var_198    float64
var_199    float64
Length: 202, dtype: object
```

In [9]:
```
#Importing the test dataset:-
test=pd.read_csv("test.csv")
```

In [10]:
```
test.head()
```

Out[10]:

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9
0	test_0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	2.1337	8.8100
1	test_1	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	-4.4131	5.9739
2	test_2	5.4827	-10.3581	10.1407	7.0479	10.2628	9.8052	4.8950	20.2537	1.5233	8.3442
3	test_3	8.5374	-1.3222	12.0220	6.5749	8.8458	3.1744	4.9397	20.5660	3.3755	7.4578
4	test_4	11.7058	-0.1327	14.1295	7.7506	9.1035	-8.5848	6.8595	10.6048	2.9890	7.1437

In [11]:
```
test.shape
```

Out[11]:
```
(200000, 201)
```

In [12]:
```
#Summary of the train dataset
train.describe()
```

Out[12]:

	target	var_0	var_1	var_2	var_3	var_4
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000
mean	0.100490	10.679914	-1.627622	10.715192	6.796529	11.078333
std	0.300653	3.040051	4.050044	2.640894	2.043319	1.623150
min	0.000000	0.408400	-15.043400	2.117100	-0.040200	5.074800
25%	0.000000	8.453850	-4.740025	8.722475	5.254075	9.883175
50%	0.000000	10.524750	-1.608050	10.580000	6.825000	11.108250
75%	0.000000	12.758200	1.358625	12.516700	8.324100	12.261125
max	1.000000	20.315000	10.376800	19.353000	13.188300	16.671400

In [13]:
```
#Summary of the test dataset
test.describe()
```

Out[13]:

	var_0	var_1	var_2	var_3	var_4	var_5
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000
mean	10.658737	-1.624244	10.707452	6.788214	11.076399	-5.050558
std	3.036716	4.040509	2.633888	2.052724	1.616456	7.869293
min	0.188700	-15.043400	2.355200	-0.022400	5.484400	-27.767000
25%	8.442975	-4.700125	8.735600	5.230500	9.891075	-11.201400
50%	10.513800	-1.590500	10.560700	6.822350	11.099750	-4.834100
75%	12.739600	1.343400	12.495025	8.327600	12.253400	0.942575
max	22.323400	9.385100	18.714100	13.142000	16.037100	17.253700

We can make few observations here:

- standard deviation is relatively large for both train and test variable data;
- min, max, mean, sdt values for train and test data looks quite close;
- mean values are distributed over a large range.

Analysing Target class

1.Check Target class count

In [14]:
```
#Target Class Count
target_class=train['target'].value_counts()
print('Count of the target class :\n',target_class)
```

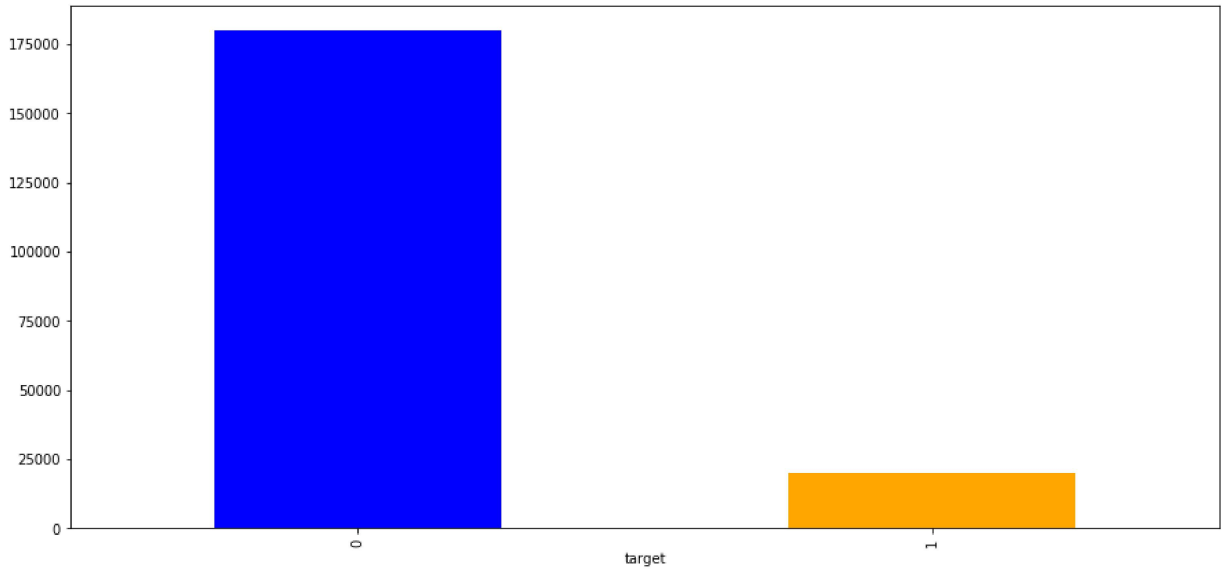
```
Count of the target class :
 0    179902
1     20098
Name: target, dtype: int64
```

2. Percentage of Target class

In [15]:
```
#Percentage of target class count
per_target_class=train['target'].value_counts()/len(train)*100
print('Percentage of target class count :\n',per_target_class)
```

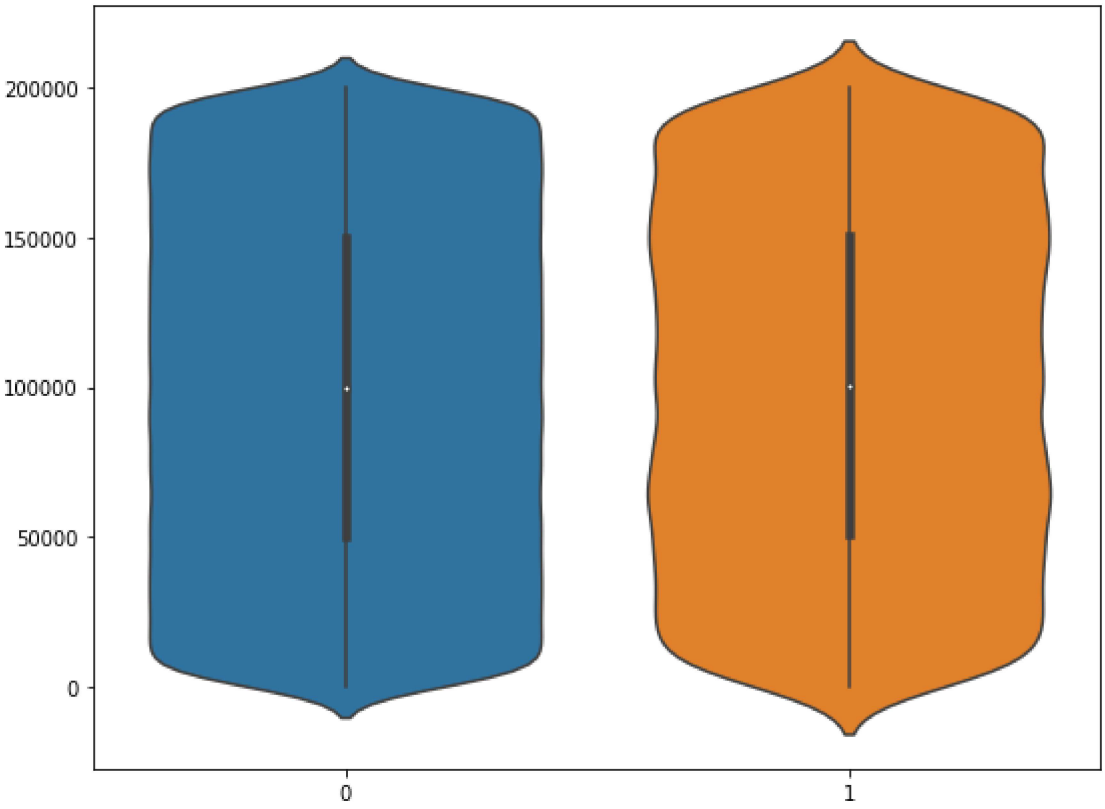
```
Percentage of target class count :
 0    89.951
1    10.049
Name: target, dtype: float64
```

In [16]:
```
## Bar plot for Target class
plt.figure(figsize=(15,7))
train.groupby(train["target"])['target'].count().plot.bar(color=['blue','orange'
plt.show()
```



In [17]:
```
## Violin plot for Target class
from matplotlib import pyplot
fig, ax = pyplot.subplots(figsize =(9, 7))
sns.violinplot(x=train.target.values,y=train.index.values,ax=ax)
```

Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2541484e710>
```



We can make few observation here

- The data is unbalanced with respect with target value.
- We are having a unbalanced data, where 90% of the data is no. of customers who will not make a transaction & 10 % of the data are those who will make a transaction.
- From the violin plots, it seems that there is no relationship between the target and index of the data frame, it is more dominated by zero compare to one's.

Density plots of features

- Let's show now the density plot of variables in train dataset.
- We represent with different colors the distribution for values with target value 0 and 1.

In [18]:
```
def plot_feature_distribution(t0, t1, label1, label2, features):
    i = 0
    sns.set_style('whitegrid')
    plt.figure()
    fig, ax = plt.subplots(10,10,figsize=(18,22))

    for feature in features:
        i += 1
        plt.subplot(10,10,i)
        sns.distplot(t0[feature], hist=False,label=label1)
        sns.distplot(t1[feature], hist=False,label=label2)
        plt.xlabel(feature, fontsize=9)
        locs, labels = plt.xticks()
        plt.tick_params(axis='x', which='major', labelsize=6, pad=-6)
        plt.tick_params(axis='y', which='major', labelsize=6)
    plt.show();
```

In [19]:
```
#Corresponding to negative class-
t0=train[train.target.values==0]

#Corresponding to possitive class-
t1=train[train.target.values==1]
```

1. For fisrt 100 observations

In [20]:
```
#train attributes from 2 to 102 -
features=train.columns.values[2:102]

#Plot distribution of train attributes-
plot_feature_distribution(t0,t1,'0','1',features)
```

```
<Figure size 432x288 with 0 Axes>
```

2. For next 100 observations

In [21]:
```
#train attributes from 102 to 202 -
features = train.columns.values[102:202]

#Plot distribution of train attributes-
plot_feature_distribution(t0, t1, '0', '1', features)
```

```
<Figure size 432x288 with 0 Axes>
```

Observation

- We can observe that there is a considerable number of features with significant different distribution for the two target values. **For example, var_0, var_1, var_2, var_5, var_9, var_13, var_106, var_109, var_139 and many others.**
- Also some features, like **var_2, var_13, var_26, var_55, var_175, var_184, var_196** shows a distribution that resambles to a bivariate distribution.

We will take this into consideration in the future for the selection of the features for our prediction model.

Outlier Analysis

In this project, we haven't perform outlier analysis due to imbalanced data.

Missing Value Analysis

In [22]:
```
#Create dataframe with missing percentage
missing_val = pd.DataFrame(train.isnull().sum())
#Reset index
missing_val = missing_val.reset_index()
missing_val.head()
```

Out[22]:

	index	0
0	ID_code	0
1	target	0
2	var_0	0
3	var_1	0
4	var_2	0

In [23]:
```
train.isnull().values.any()
```

Out[23]:
```
False
```

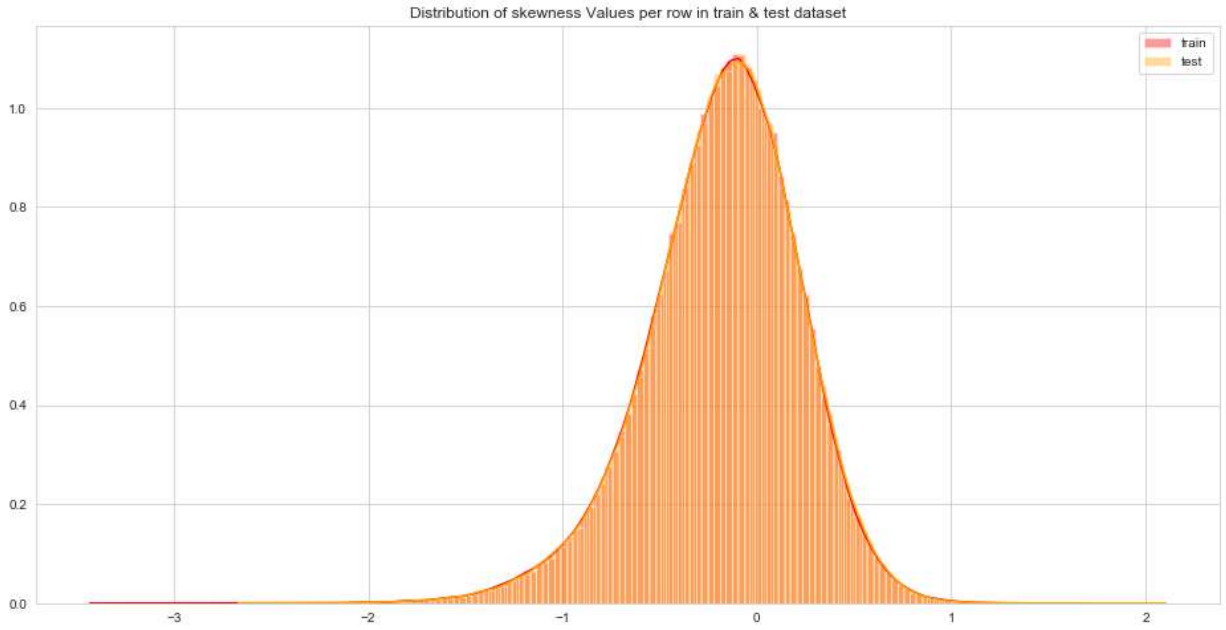
In [24]:
```
test.isnull().values.any()
```

Out[24]:
```
False
```

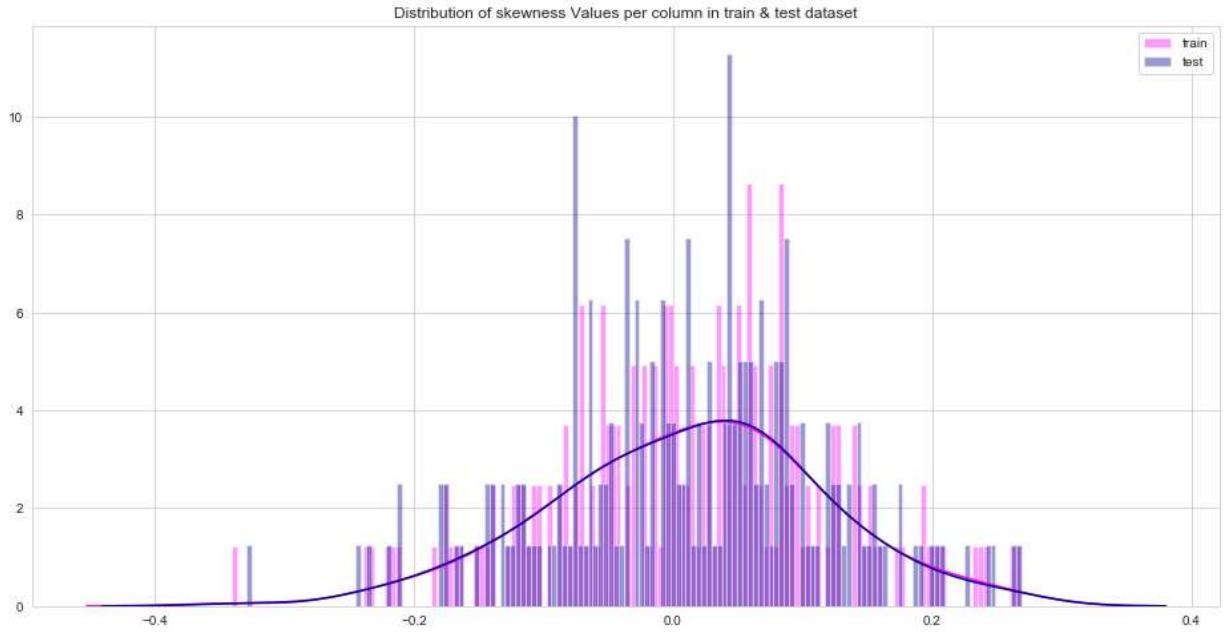
Observation :- There are no missing data in train and test datasets.

Let's see now what is the distribution of skew values per rows and columns.

In [25]:
```
#Normality check of training and test data##
plt.figure(figsize=(16,8))

#Train attributes-
train_attributes=train.columns.values[2:202]

#Test attributes-
test_attributes=test.columns.values[1:201]

#Distribution plot for skew values per rows in train attributes:
sns.distplot(train[train_attributes].skew(axis=1),color='red',kde=True,bins=150,

#Distribution plot for skew values per rows in test attributes:
sns.distplot(test[test_attributes].skew(axis=1),color='orange',kde=True,bins=150

plt.title('Distribution of skewness Values per row in train & test dataset')
plt.legend()
plt.show()
```



In [26]:
```
plt.figure(figsize=(16,8))

#Distribution plot for skew values per column in train attributes:
sns.distplot(train[train_attributes].skew(axis=0),color='magenta',kde=True,bins=

#Distribution plot for skew values per column in test attributes:
sns.distplot(test[test_attributes].skew(axis=0),color='darkblue',kde=True,bins=1

plt.title('Distribution of skewness Values per column in train & test dataset')
plt.legend()
plt.show()
```



Feature Selection

1. Correlation in Train and Test dataset respectiviely

In [27]:
```
#Correlation in train attiributes-
train_attributes2=train.columns.values[2:202]
train_correlation=train[train_attributes2].corr().abs().unstack().sort_values(ki
train_correlation=train_correlation[train_correlation['level_0']!=train_correlat
print('----Least correlated features---- :\n',train_correlation.head(10))
print('----Top most correlated features---- :\n',train_correlation.tail(10))
```

```
----Least correlated features---- :
    level_0  level_1            0
0   var_75  var_191  2.703975e-08
1  var_191   var_75  2.703975e-08
2  var_173    var_6  5.942735e-08
3    var_6  var_173  5.942735e-08
4  var_126  var_109  1.313947e-07
5  var_109  var_126  1.313947e-07
6  var_144   var_27  1.772502e-07
7   var_27  var_144  1.772502e-07
8  var_177  var_100  3.116544e-07
9  var_100  var_177  3.116544e-07
----Top most correlated features---- :
        level_0  level_1         0
39790  var_183  var_189  0.009359
39791  var_189  var_183  0.009359
39792  var_174   var_81  0.009490
39793   var_81  var_174  0.009490
39794   var_81  var_165  0.009714
39795  var_165   var_81  0.009714
39796   var_53  var_148  0.009788
39797  var_148   var_53  0.009788
39798   var_26  var_139  0.009844
39799  var_139   var_26  0.009844
```

Observation: Its visible that correlation between train attributes is very small.

In [28]:
```
#Correlation in test attiributes-
test_attributes=test.columns.values[1:201]
test_correlation=test[test_attributes].corr().abs().unstack().sort_values(kind='
test_correlation=test_correlation[test_correlation['level_0']!=test_correlation[
print('-----Least correlated features---- :\n',test_correlation.head(10))
print('-----Top most correlated features---- :\n',test_correlation.tail(10))
```

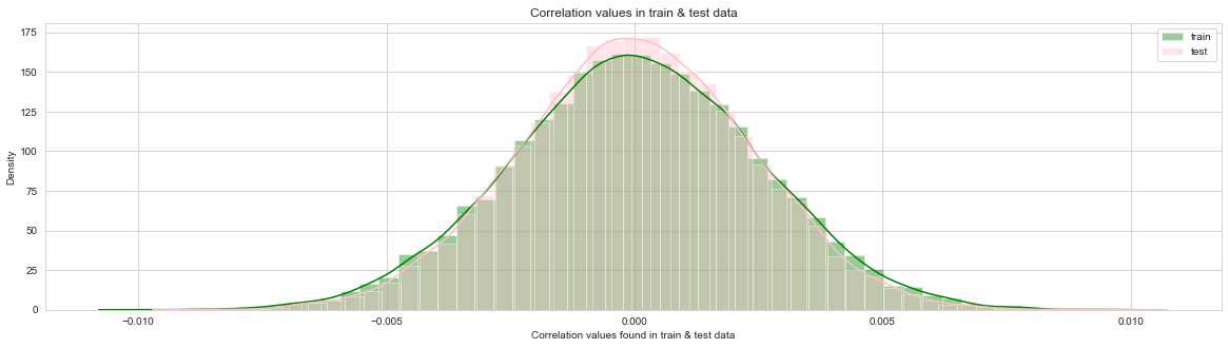
```
-----Least correlated features---- :
    level_0  level_1            0
0  var_154  var_175  1.477268e-07
1  var_175  var_154  1.477268e-07
2  var_188  var_113  1.639749e-07
3  var_113  var_188  1.639749e-07
4  var_131    var_8  4.695407e-07
5    var_8  var_131  4.695407e-07
6   var_60  var_189  9.523709e-07
7  var_189   var_60  9.523709e-07
8  var_159   var_96  1.147835e-06
9   var_96  var_159  1.147835e-06
-----Top most correlated features---- :
        level_0  level_1         0
39790  var_122  var_164  0.008513
39791  var_164  var_122  0.008513
39792  var_164    var_2  0.008614
39793    var_2  var_164  0.008614
39794   var_31  var_132  0.008714
39795  var_132   var_31  0.008714
39796   var_96  var_143  0.008829
39797  var_143   var_96  0.008829
39798  var_139   var_75  0.009868
39799   var_75  var_139  0.009868
```

Observation: Its visible that correlation between test attributes is very small.

Correlation plot for Train and Teast data

In [81]:
```
train_correlation=train[train_attributes2].corr()
train_correlation=train_correlation.values.flatten()
train_correlation=train_correlation[train_correlation!=1]


test_correlation=test[test_attributes].corr()
test_correlation=test_correlation.values.flatten()
test_correlation=test_correlation[test_correlation!=1]

plt.figure(figsize=(20,5))
sns.distplot(train_correlation,color="green",label="train")
sns.distplot(test_correlation,color="pink",label="test")
plt.xlabel("Correlation values found in train & test data")
plt.ylabel("Density")
plt.title ("Correlation values in train & test data")
plt.legend()
```

Out[81]: <matplotlib.legend.Legend at 0x1a683575160>



Observation: The correlation between the train and test data is very small, its completely visible from the above graph

2. Feature Engineering :-

Performing feature engineering by using **Permutation Importance**

In [29]:
```
#Training & testing data:
X=train.drop(columns=['ID_code','target'],axis=1)
y=train['target']
test_01=test.drop(columns=['ID_code'],axis=1)

#Split the train data:-
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size = 0.25,random_state
print('Train data size:',train.shape)
print('X_train size :',X_train.shape)
print('X_test size :',X_test.shape)
print('y_train size :',y_train.shape)
print('y_test size :', y_test.shape)
```

```
Train data size: (200000, 202)
X_train size : (150000, 200)
X_test size : (50000, 200)
y_train size : (150000,)
y_test size : (50000,)
```

In [83]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [30]:
```
rf_model=RandomForestClassifier(n_estimators=10,random_state=42)

#fitting the model:-
rf_model.fit(X_test,y_test)

#Permutation Importance:-
from eli5.sklearn import PermutationImportance
perm_imp=PermutationImportance(rf_model,random_state=42)

#fitting the model:-
perm_imp.fit(X_test,y_test)
```

Out[30]:
```
PermutationImportance(cv='prefit',
                      estimator=RandomForestClassifier(bootstrap=True,
                                                       ccp_alpha=0.0,
                                                       class_weight=None,
                                                       criterion='gini',
                                                       max_depth=None,
                                                       max_features='auto',
                                                       max_leaf_nodes=None,
                                                       max_samples=None,
                                                       min_impurity_decrease=0.
0,
                                                       min_impurity_split=None,
                                                       min_samples_leaf=1,
                                                       min_samples_split=2,
                                                       min_weight_fraction_leaf
=0.0,
                                                       n_estimators=10,
                                                       n_jobs=None,
                                                       oob_score=False,
                                                       random_state=42,
                                                       verbose=0,
                                                       warm_start=False),
                      n_iter=5, random_state=42, refit=True, scoring=None)
```

In [85]:

Out[85]:
```
PermutationImportance(cv='prefit',
                      estimator=RandomForestClassifier(bootstrap=True,
                                                       ccp_alpha=0.0,
                                                       class_weight=None,
                                                       criterion='gini',
                                                       max_depth=None,
                                                       max_features='auto',
                                                       max_leaf_nodes=None,
                                                       max_samples=None,
                                                       min_impurity_decrease=0.
0,
                                                       min_impurity_split=None,
                                                       min_samples_leaf=1,
                                                       min_samples_split=2,
                                                       min_weight_fraction_leaf
=0.0,
                                                       n_estimators=10,
                                                       n_jobs=None,
                                                       oob_score=False,
                                                       random_state=42,
                                                       verbose=0,
                                                       warm_start=False),
                      n_iter=5, random_state=42, refit=True, scoring=None)
```

In [31]:
```
#Important Features:-
eli5.show_weights(perm_imp,feature_names=X_test.columns.tolist(),top=200)
```

0.0076 ± 0.0003	var_170
0.0076 ± 0.0003	var_12
0.0075 ± 0.0004	var_26
0.0075 ± 0.0002	var_166
0.0073 ± 0.0004	var_169
0.0070 ± 0.0004	var_21
0.0063 ± 0.0002	var_6
0.0057 ± 0.0002	var_165
0.0057 ± 0.0001	var_80
0.0056 ± 0.0005	var_179
0.0055 ± 0.0004	var_146
0.0052 ± 0.0002	var_109
0.0051 ± 0.0002	var_174
0.0051 ± 0.0003	var_78
0.0049 ± 0.0003	var_99
0.0046 ± 0.0005	var_94
0.0045 ± 0.0004	var_149
0.0044 ± 0.0003	var_0
0.0042 ± 0.0005	var_33
0.0040 ± 0.0004	var_13
0.0040 ± 0.0002	var_198
0.0040 ± 0.0003	var_108

Observation:-

- Importance of features is decreasing as we move down the top of column.
- Features showing in green indicates they are having positive impact on our prediction.
- Features showing in white showing they have no impact on prediction.

- Most improtant feature is var_81.

Handling of imblanced data:-

We are going to use multiple approaches for dealing with imbalanced datasets.

- Oversample minority class.
- Undersample majority class.
- Change of performance matrix.
- SMOTE (Synthetic Minority Oversampling technique)
- Change of algorithm.

Apply logistic Regression on imbalanced dataset and check how the model works.

In [32]:
```
from sklearn.metrics import accuracy_score as score
```

In [33]:
```
#Training & testing data:
X1=train.drop(columns=['ID_code','target'],axis=1)
Y1=train['target']


#Stratified KFold Cross Validator:-
skf=StratifiedKFold(n_splits=5, random_state=42, shuffle=True)
for train_index, valid_index in skf.split(X1,Y1):
    X1_train, X1_valid = X1.iloc[train_index], X1.iloc[valid_index]
    Y1_train, Y1_valid = Y1.iloc[train_index], Y1.iloc[valid_index]

print('Shape of X_train :',X1_train.shape)
print('Shape of X_valid :',X1_valid.shape)
print('Shape of y_train :',Y1_train.shape)
print('Shape of y_valid :',Y1_valid.shape)

#Split the train data:-
#X1_train,X1_test,Y1_train,Y1_test=train_test_split(X1,Y1,test_size = 0.25,rando
#print('Train data size:',train.shape)
#print('X1_train size :',X1_train.shape)
#print('X1_test size :',X1_test.shape)
#print('Y1_train size :',Y1_train.shape)
#print('Y1_test size :', Y1_test.shape)
```

```
Shape of X_train : (160000, 200)
Shape of X_valid : (40000, 200)
Shape of y_train : (160000,)
Shape of y_valid : (40000,)
```

In [34]:
```
## apply logistic regression
lr_model_main=LogisticRegression(random_state=42)
#fitting the model-
lr_model_main.fit(X1_train,Y1_train)

#Accuracy of model-
lr_score=lr_model_main.score(X1_train,Y1_train)
print('Accuracy of lr_main_model :',lr_score)
```

```
Accuracy of lr_main_model : 0.91219375
```

1.Applying Up-sampling

In [35]:
```
#import library
from sklearn.utils import resample

# Separate majority and minority classes
train_majority = train[train.target==0]
train_minority = train[train.target==1]
# Upsample minority class
train_minority_upsampled = resample(train_minority,
                                    replace=True,     # sample with replacement
                                    n_samples=179902,    # to match majority class
                                    random_state=42) # reproducible results

# Combine majority class with upsampled minority class
upsampled = pd.concat([train_majority, train_minority_upsampled])

# Display new class counts
upsampled.target.value_counts()
```

Out[35]:
```
1    179902
0    179902
Name: target, dtype: int64
```

Apply Logistic Regression model on Up-Sampled data

In [36]:
```
## spliting test and train data
X2=upsampled.drop(columns=['ID_code','target'],axis=1)
Y2=upsampled['target']
X2_train,X2_test,Y2_train,Y2_test=train_test_split(X2,Y2,test_size = 0.25,random_

## apply losgistic regression
lr_model_upsample=LogisticRegression(random_state=42)
lr_model_upsample.fit(X2_train,Y2_train)

##Acuracy after upsampling
lr_score=lr_model_upsample.score(X2_train,Y2_train)
print('Accuracy of lr_upsample_model :',lr_score)
```

```
Accuracy of lr_upsample_model : 0.7703638647708196
```

Observation:-

- Adding more copies of minority class.
- It can be a good option we dont have that much large data to work.
- Drawback of this process is we are adding info. That can lead to overfitting or poor performance on test data.

2.Applying Down-Sampling

In [37]:
```
# Separate majority and minority classes
train_majority = train[train.target==0]
train_minority = train[train.target==1]
# Upsample minority class
train_majority_down = resample(train_majority,
                                 replace=True,     # sample with replacement
                                 n_samples=20098,    # to match majority class
                                 random_state=42) # reproducible results

# Combine majority class with upsampled minority class
Down_sampled = pd.concat([train_minority, train_majority_down])

# Display new class counts
Down_sampled.target.value_counts()
```

Out[37]:
```
1    20098
0    20098
Name: target, dtype: int64
```

Apply Logistic Regression model on Down-sampled data

In [38]:
```
## spliting test and train data
X3=Down_sampled.drop(columns=['ID_code','target'],axis=1)
Y3=Down_sampled['target']
X3_train,X3_test,Y3_train,Y3_test=train_test_split(X3,Y3,test_size = 0.25,random_

#apply logistic regression
lr_model_Down=LogisticRegression(random_state=42)
lr_model_Down.fit(X3_train,Y3_train)

##Acuracy after Down sapmling
lr_score=lr_model_Down.score(X3_train,Y3_train)
print('Accuracy of lr_Downsample_model :',lr_score)
```

```
Accuracy of lr_Downsample_model : 0.7693966232129233
```

Observation :-

- Removing some copies of majority class.
- It can be a good option if we have very large amount of data say in millions to work.
- Drawback of this process is we are removing some valuable info. that can leads to underfitting & poor performance on test data.

a poor performance on test data.

3. Change Permormance matrix

3.1 Accuracy

In [39]:
```
## apply logistic regression
lr_model_main=LogisticRegression(random_state=42)
#fitting the model-
lr_model_main.fit(X1_train,Y1_train)

#Accuracy of model-
lr_score=lr_model_main.score(X1_train,Y1_train)
print('Accuracy of lr_main_model :',lr_score)
```

```
Accuracy of lr_main_model : 0.91219375
```

In [40]:
```
#Cross validation prediction of lr_model-
cv_predict=cross_val_predict(lr_model_main,X1_valid,Y1_valid,cv=5)
#Cross validation score-
cv_score=cross_val_score(lr_model_main,X1_valid,Y1_valid,cv=5)
print('cross val score :',np.average(cv_score))



#Cross validation prediction of lr_model-
#cv_predict=cross_val_predict(lr_model_main,X1_test,Y1_test,cv=5)
#Cross validation score-
#cv_score=cross_val_score(lr_model_main,X1_test,Y1_test,cv=5)
#print('cross val score :',np.average(cv_score))
```

```
cross val score : 0.9102750000000001
```

Accuracy of the model is not th best metric to use while evaluating the imbalanced datasets as it may be misleading. We are going to change the performance metric.

3.2 Confusion Matrix:

In [41]:
```
#Confusion matrix:-
cm=confusion_matrix(Y1_valid,cv_predict)
cm=pd.crosstab(Y1_valid,cv_predict)
cm
```

Out[41]:

col_0	0	1
target		
0	35484	496
1	3093	927

3.2(a) ROC AUC Score

In [140]:
```
#ROC_AUC SCORE:-
roc_score=roc_auc_score(Y1_valid,cv_predict)
print('ROC Score:',roc_score)
```

```
ROC Score: 0.6084057892859217
```

Observation:- On comparing roc_auc_score and model accuracy, model is not performing well on imbalanced data.

3.2(b) Classification report(Precision, Recall, F1 score)

In [141]:
```
#Classification report:-
classification_scores=classification_report(Y1_valid,cv_predict)
print(classification_scores)
```

```
              precision    recall  f1-score   support

           0       0.92      0.99      0.95     35980
           1       0.65      0.23      0.34      4020

    accuracy                           0.91     40000
   macro avg       0.79      0.61      0.65     40000
weighted avg       0.89      0.91      0.89     40000
```

Observation:- As we see that f1 score is high for the customers who will not make a transaction, compare to those who will make a transaction. So, we are going to change the algorithm.

In [142]:
```
#Model performance on test data:-
X1_valid=test.drop(['ID_code'],axis=1)
lr_pred=lr_model_main.predict(X1_valid)
print(lr_pred)
```

```
[0 0 0 ... 0 0 0]
```

In [ ]:
```
#from sklearn.preprocessing import PolynomialFeatures
#pol = PolynomialFeatures
#X5=train.drop(columns=['ID_code','target'],axis=1)
#y5=train['target']
#X5_train, X5_test, y5_train, y5_test = train_test_split(X5, y5, test_size=0.25,
#poly_reg = PolynomialFeatures(degree = 4)
#X_poly = poly_reg.fit_transform(X5_train)
```

4. SMOTE (Synthetic Minority Oversampling technique):-

- As per the drawbacks of both Upsampling and Down-sampling models we will use SMOTE (Synthetic Minority Oversampling technique)
- This is a statistical technique for increasing the number of cases in your dataset in a balanced way. It uses a nearest neighbors algorithm to generate new and synthetic data to used for

training the model.

In [103]:
```
from imblearn.over_sampling import SMOTE
```

In [143]:
```
X5=train.drop(columns=['ID_code','target'],axis=1)
Y5=train['target']
skf=StratifiedKFold(n_splits=5, random_state=42, shuffle=True)
for train_index, valid_index in skf.split(X1,Y1):
    X5_train, X5_valid = X5.iloc[train_index], X5.iloc[valid_index]
    Y5_train, Y5_valid = Y5.iloc[train_index], Y5.iloc[valid_index]

print('Shape of X_train :',X5_train.shape)
print('Shape of X_valid :',X5_valid.shape)
print('Shape of y_train :',Y5_train.shape)
print('Shape of y_valid :',Y5_valid.shape)
```

```
Shape of X_train : (160000, 200)
Shape of X_valid : (40000, 200)
Shape of y_train : (160000,)
Shape of y_valid : (40000,)
```

In [144]:
```
#SMOTE:-
sm = SMOTE(random_state=42)
#Generating synthetic data points
X_smote,y_smote=sm.fit_sample(X5_train,Y5_train)
X_smote_v,y_smote_v=sm.fit_sample(X5_valid,Y5_valid)
```

4.1 Building Logistsic regression model on synthetic data points:-

In [145]:
```
#Logistic regression model for SMOTE:-
smote=LogisticRegression(random_state=42)
#fitting the smote model:-
smote.fit(X_smote,y_smote)
```

Out[145]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=42, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

4.2 Accuracy

In [146]:
```
#Accuracy of the model:-
smote_score=smote.score(X_smote,y_smote)
print('Accuracy of the smote_model :',smote_score)
```

```
Accuracy of the smote_model : 0.7887779491669099
```

In [147]:
```
#Cross validation prediction for SMOTE:-
cv_pred=cross_val_predict(smote,X_smote_v,y_smote_v,cv=5)
#Cross validation score:-
cv_score=cross_val_score(smote,X_smote_v,y_smote_v,cv=5)
print('Cross validation score :',np.average(cv_score))
```

```
Cross validation score : 0.796247915508616
```

4.3 Confusion Matrix

In [148]:
```
#Confusion matrix:-
cm=confusion_matrix(y_smote_v,cv_pred)
cm=pd.crosstab(y_smote_v,cv_pred)
cm
```

Out[148]:

col_0	0	1
target		
0	28192	7788
1	6874	29106

4.3(a) ROC_AUC Score

In [149]:
```
#ROC_AUC SCORE:-
roc_score=roc_auc_score(y_smote_v,cv_pred)
print('ROC score:',roc_score)
```

```
ROC score: 0.7962479155086158
```

4.3(b) Classification report(Precision, Recall, F1 score)

In [150]:
```
#Classification Report:-
scores=classification_report(y_smote_v,cv_pred)
print(scores)
```

```
              precision    recall  f1-score   support

           0       0.80      0.78      0.79     35980
           1       0.79      0.81      0.80     35980

    accuracy                           0.80     71960
   macro avg       0.80      0.80      0.80     71960
weighted avg       0.80      0.80      0.80     71960
```

Observation:- As we see that f1 score is high for the customers who will not make a transaction, as well as who will make a transaction.

4.4 Model_performance on test data:-

In [112]:
```
#Predicting the model-
X1_test=test.drop(['ID_code'],axis=1)
smote_pred=smote.predict(X_test)
print(smote_pred)
```

```
[1 1 0 ... 0 0 1]
```

Observation:- We can observe that the smote model is performing well on imbalance data as compare to logistic regression.

5. Change of Algorithm(Light GBM):-

It is a gradient boosting framework that uses tree based learning algorithm.

In [113]:
```
#Training data-
lgb_train=lgb.Dataset(X_train,label=y_train)

#Validation data-
lgb_valid=lgb.Dataset(X_test,label=y_test)
```

In [117]:
```
#Selecting best hyperparameters by tuning of different parameters:-
params={'boosting_type': 'gbdt',
          'max_depth' : -1, #no limit for max_depth if <0
          'objective': 'binary',
          'boost_from_average':False,
          'nthread': 20,
          'metric':'auc',
          'num_leaves': 50,
          'learning_rate': 0.01,
          'max_bin': 100,      #default 255
          'subsample_for_bin': 100,
          'subsample': 1,
          'subsample_freq': 1,
          'colsample_bytree': 0.8,
          'bagging_fraction':0.5,
          'bagging_freq':5,
          'feature_fraction':0.08,
          'min_split_gain': 0.45, #>0
          'min_child_weight': 1,
          'min_child_samples': 5,
          'is_unbalance':True,
          }
```

In [119]:
```
#Training lgbm model:-
num_rounds=10000
lgbm= lgb.train(params,lgb_train,num_rounds,valid_sets=[lgb_train,lgb_valid],verb
lgbm
```

```
Training until validation scores don't improve for 5000 rounds
[1000]  training's auc: 0.94228 valid_1's auc: 1
[2000]  training's auc: 0.961604        valid_1's auc: 1
[3000]  training's auc: 0.974882        valid_1's auc: 1
[4000]  training's auc: 0.984071        valid_1's auc: 1
[5000]  training's auc: 0.990289        valid_1's auc: 1
Early stopping, best iteration is:
[1]     training's auc: 0.623702        valid_1's auc: 1
```

Out[119]:
```
<lightgbm.basic.Booster object at 0x000001A68336E860>
```

LGBM model performance on test data:-

In [121]:
```
X1_test=test.drop(['ID_code'],axis=1)
#Predict the model:-

#probability predictions
lgbm_predict_prob=lgbm.predict(X_test,random_state=42,num_iteration=lgbm.best_ite

#Convert to binary output 1 or 0
lgbm_predict=np.where(lgbm_predict_prob>=0.5,1,0)
print(lgbm_predict_prob)
print(lgbm_predict)
```

```
[0.49802717 0.49891065 0.49846851 ... 0.49939738 0.50047844 0.49908944]
[0 0 0 ... 0 1 0]
```

Plotting of important Features:-

In [124]:
```
lgb.plot_importance(lgbm,max_num_features=50,importance_type="split",figsize=(20
```

Out[124]:
```
<matplotlib.axes._subplots.AxesSubplot object at 0x000001A683130A20>
```

In [151]:
```
#Final submission:-
sub=pd.DataFrame({'ID_code':test['ID_code'].values})
sub['lgbm_predict_prob']=lgbm_predict_prob
sub['lgbm_predict']=lgbm_predict
sub.to_csv('Final.csv',index=False)
sub.head()
```

Out[151]:

	ID_code	lgbm_predict_prob	lgbm_predict
0	test_0	0.498027	0
1	test_1	0.498911	0
2	test_2	0.498469	0
3	test_3	0.500478	1
4	test_4	0.500855	1

In [ ]:
```

```