Santander Customer Transaction Prediction

Dataset used: Santander Customer Transaction Prediction

Introduction

In this challenge, Santander invites Kagglers to help them identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted. The data provided for this competition has the same structure as the real data they have available to solve this problem.

The data is anonimyzed, each row containing 200 numerical values identified just with a number.

In the following we will explore the data, prepare it for a model, train a model and predict the target value for the test set, then prepare a submission.

Stay tuned, I will frequently update this Kernel in the next days.

Prepare for data analysis

In [1]:

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

import warnings
warnings.filterwarnings("ignore")
```

Load Data and Reducing Memory Usage

In [2]:

```
#taken from https://github.com/kranthik13/Santander-Customer-Transaction-Prediction/blob/ma

def reduce_mem_usage(df):
    """ iterate through all the columns of a dataframe and modify the data type
        to reduce memory usage.
    """
    start_mem = df.memory_usage().sum() / 1024 ** 2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            df[col] = df[col].astype('category')

    end_mem = df.memory_usage().sum() / 1024 ** 2
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))

    return df

def import_data(file):
    """create a dataframe and optimize its memory usage"""
    df = pd.read_csv(file, parse_dates=True, keep_date_col=True)
    df = reduce_mem_usage(df)
    return df
```

Shape of Training and Test Data

In [40]:

```
train = import_data("train.csv")
test = import_data("test.csv")

print("\n\nTrain Size : \t{}\nTest Size : \t{}".format(train.shape, test.shape))
```

```
Memory usage of dataframe is 308.23 MB
Memory usage after optimization is: 83.77 MB
Decreased by 72.8%
Memory usage of dataframe is 306.70 MB
Memory usage after optimization is: 83.58 MB
Decreased by 72.7%


Train Size :    (200000, 202)
Test Size :     (200000, 201)
```

We can see that the train Dataset has 202 columns while the test Dataset has 201 Columns. The extra column in the Train Dataset is the target data set which is not present in the Test Dataset

In [4]:

```
train.head(2)
```

Out[4]:

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6
0	train_0	0	8.921875	-6.785156	11.906250	5.093750	11.460938	-9.281250	5.117188
1	train_1	0	11.500000	-4.148438	13.859375	5.390625	12.359375	7.042969	5.621094

2 rows × 202 columns

The data obtained is entirely masked so with even domain knowledge we will not be able to find out any significant features. We can try with basic features like mean, standard deviation, counts, median, etc. We will do feature engineering later.

Basic Stats

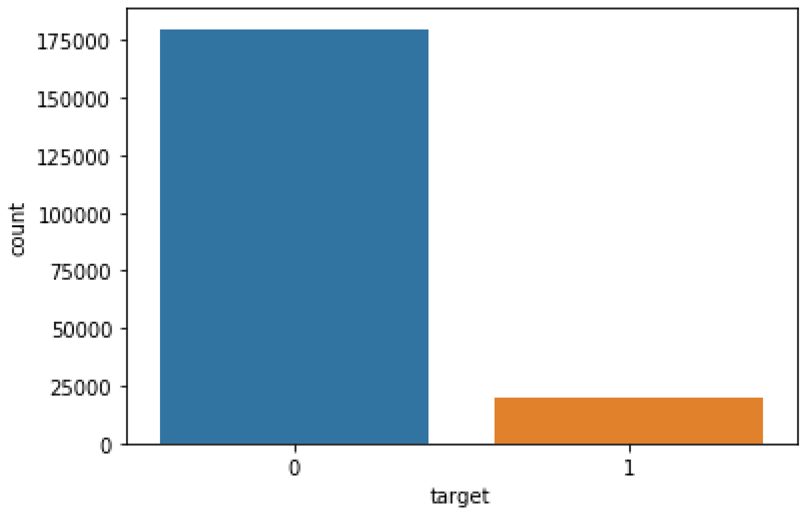
Target Distribution

In [5]:

```
sns.countplot(train['target'])
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a4300b0208>
```



In [6]:

```
train.target.value_counts()
```

Out[6]:

```
0    179902
1     20098
Name: target, dtype: int64
```

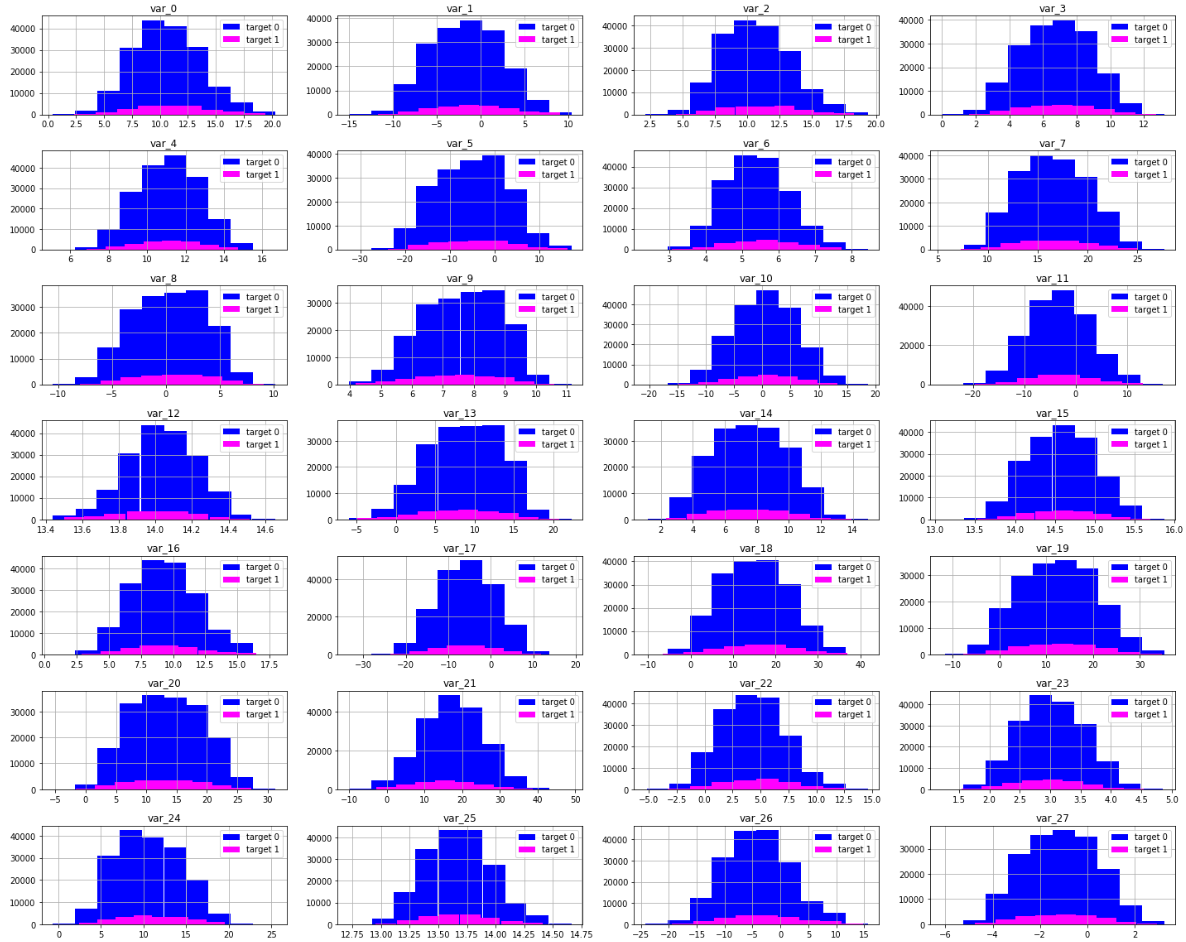
In [7]:

```
t0=train[train['target']==0]
t1=train[train['target']==1]
```

In [8]:

```
print('Distributions of 1st 100 features')
plt.figure(figsize=(20,16))
for i, col in enumerate(list(train.columns)[2:30]):
    plt.subplot(7,4,i + 1)
    plt.hist(t0[col],label='target 0',color='blue')
    plt.hist(t1[col],label='target 1',color='magenta')
    plt.title(col)
    plt.grid()
    plt.legend(loc='upper right')
    plt.tight_layout()
```

```
Distributions of 1st 100 features
```



We can see from the above that nearly 90% of the Target value is 0(we assume that 0 stands for Customer didnot do transaction) and only 10% is 1(we assume 1 stands for Customer did a Transaction).

This makes the data significantly imbalanced

In [9]:

```
train.drop(['ID_code'],axis=1,inplace=True)
labels=train['target']
train.drop(['target'],axis=1,inplace=True)
```

In [10]:

```
train.select_dtypes(include='float16')
```

Out[10]:

	var_0	var_1	var_2	var_3	var_4	var_5	var_6	va
0	8.921875	-6.785156	11.906250	5.093750	11.460938	-9.281250	5.117188	18.625
1	11.500000	-4.148438	13.859375	5.390625	12.359375	7.042969	5.621094	16.531
2	8.609375	-2.746094	12.078125	7.894531	10.585938	-9.085938	6.941406	14.617
3	11.062500	-2.152344	8.953125	7.195312	12.585938	-1.835938	5.843750	14.921
4	9.835938	-1.483398	12.875000	6.636719	12.273438	2.449219	5.941406	19.250
5	11.476562	-2.318359	12.609375	8.625000	10.960938	3.560547	4.531250	15.226
6	11.812500	-0.083191	9.351562	4.292969	11.132812	-8.023438	6.195312	12.078
7	13.554688	-7.988281	13.875000	7.597656	8.656250	0.831055	5.687500	22.328
8	16.109375	2.443359	13.929688	5.632812	8.804688	6.164062	4.453125	10.187
9	12.507812	1.974609	8.898438	5.449219	13.601562	-16.281250	6.062500	16.843
10	5.070312	-0.544922	9.593750	4.296875	12.390625	-18.875000	6.039062	14.382
11	12.718750	-7.976562	10.375000	9.007812	12.859375	-12.085938	5.644531	11.835
12	8.765625	-4.617188	9.726562	7.425781	9.023438	1.424805	6.281250	12.312
13	16.375000	1.593750	16.734375	7.332031	12.148438	5.902344	4.820312	20.968
14	13.804688	5.050781	17.265625	8.515625	12.851562	-9.164062	5.734375	21.046
15	3.941406	2.656250	13.367188	6.890625	12.281250	-16.156250	5.699219	14.460
16	5.062500	0.268799	15.132812	3.658203	13.531250	-6.546875	5.277344	9.867
17	8.421875	-1.812500	8.117188	5.394531	9.718750	-17.843750	4.097656	15.289
18	4.875000	1.264648	11.921875	8.468750	10.718750	-0.670898	5.609375	16.468
19	4.410156	-0.786133	15.179688	8.062500	11.281250	-0.735840	6.378906	16.015
20	12.671875	-2.021484	6.894531	6.914062	9.570312	-11.265625	5.605469	16.234
21	8.390625	1.480469	12.976562	7.554688	11.156250	-14.781250	6.000000	15.515
22	10.203125	0.192505	14.023438	7.035156	11.851562	13.882812	6.402344	18.000
23	15.000000	-9.343750	10.382812	8.320312	13.023438	-5.074219	5.250000	11.687
24	5.925781	-3.728516	11.101562	4.695312	11.734375	-20.406250	5.812500	15.906
25	8.273438	-5.683594	12.687500	7.277344	12.375000	-7.753906	6.726562	18.421
26	15.656250	-4.496094	10.484375	3.818359	8.882812	-6.031250	5.523438	17.703
27	10.718750	-9.976562	10.953125	6.765625	10.679688	-12.929688	4.500000	17.390
28	7.800781	4.527344	8.929688	8.492188	12.843750	-1.263672	5.039062	13.632
29	5.332031	-2.605469	13.187500	3.119141	6.648438	-6.566406	5.906250	15.234
...	...	...	...	...	...	...	...	
199970	15.578125	-2.568359	9.882812	4.257812	10.750000	-2.974609	6.261719	19.656
199971	14.578125	-3.917969	13.164062	9.281250	10.960938	7.730469	5.160156	14.882
199972	7.421875	-2.597656	12.023438	8.789062	14.203125	0.064880	4.742188	19.921

	var_0	var_1	var_2	var_3	var_4	var_5	var_6	va
199973	8.773438	-0.460449	8.085938	6.453125	13.000000	0.187256	4.347656	20.046
199974	16.203125	-5.785156	15.265625	6.523438	11.601562	-7.312500	5.785156	18.453
199975	7.523438	1.054688	9.625000	4.867188	7.183594	-16.406250	5.335938	13.109
199976	7.964844	-2.847656	9.093750	7.328125	9.671875	-16.781250	4.507812	12.437
199977	7.386719	-0.809082	8.929688	7.722656	14.320312	-11.320312	5.230469	12.062
199978	12.203125	-0.878906	16.812500	9.109375	10.320312	7.019531	6.257812	19.828
199979	10.820312	-2.933594	15.093750	9.781250	9.398438	-4.570312	5.320312	16.078
199980	7.960938	6.226562	13.335938	7.535156	12.562500	1.120117	5.449219	22.156
199981	12.812500	0.638672	14.164062	7.105469	8.937500	-0.327393	6.593750	14.609
199982	11.820312	-2.359375	9.593750	10.562500	11.164062	-0.756836	4.894531	19.609
199983	15.304688	2.826172	11.406250	4.800781	11.453125	5.375000	4.507812	15.328
199984	11.320312	0.374512	4.558594	5.886719	10.367188	0.718262	5.441406	17.453
199985	9.023438	-3.638672	12.390625	5.191406	12.171875	-8.898438	5.257812	17.984
199986	12.031250	-8.781250	7.707031	7.402344	9.234375	-16.218750	5.906250	17.921
199987	8.046875	-1.917969	13.148438	9.203125	8.992188	2.753906	4.218750	18.109
199988	10.867188	-8.351562	6.171875	7.605469	9.554688	-15.765625	5.871094	18.984
199989	11.757812	-4.535156	9.226562	7.773438	10.625000	-2.248047	5.476562	12.445
199990	14.148438	1.856445	11.007812	3.677734	12.195312	-16.593750	5.320312	14.851
199991	9.992188	2.552734	11.968750	6.394531	13.546875	-9.531250	6.085938	14.179
199992	12.281250	2.691406	15.468750	6.425781	10.984375	9.968750	4.503906	9.921
199993	13.218750	-5.800781	9.726562	6.589844	12.460938	-7.164062	6.066406	12.992
199994	12.390625	-5.882812	11.234375	3.923828	10.453125	10.726562	7.050781	18.703
199995	11.484375	-0.495605	8.265625	3.513672	10.343750	11.609375	5.671875	15.148
199996	4.914062	-2.449219	16.703125	6.632812	8.312500	-10.562500	5.878906	21.593
199997	11.226562	-5.050781	10.515625	5.644531	9.343750	-5.410156	4.554688	21.562
199998	9.710938	-8.609375	13.609375	5.792969	12.515625	0.533691	6.046875	17.015
199999	10.875000	-5.710938	12.117188	8.031250	11.554688	0.348877	5.285156	15.203

200000 rows × 200 columns

In [11]:

```
train.astype(np.float64).describe()
```

Out[11]:

	var_0	var_1	var_2	var_3	var_4	va
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.0000
mean	10.679915	-1.627622	10.715197	6.796529	11.078332	-5.065
std	3.040059	4.050044	2.640890	2.043315	1.623149	7.863
min	0.408447	-15.046875	2.117188	-0.040192	5.074219	-32.562
25%	8.453125	-4.738281	8.718750	5.253906	9.882812	-11.203
50%	10.523438	-1.608398	10.578125	6.824219	11.109375	-4.832
75%	12.757812	1.358398	12.515625	8.320312	12.257812	0.924
max	20.312500	10.375000	19.359375	13.187500	16.671875	17.250

8 rows × 200 columns

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;

Missing Values:

In [12]:

```
def missing_data(data):
    total = data.isnull().sum()
    percent = (data.isnull().sum()/data.isnull().count()*100)
    tt = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
    types = []
    for col in data.columns:
        dtype = str(data[col].dtype)
        types.append(dtype)
    tt['Types'] = types
    return(np.transpose(tt))
```

In [13]:

```
missing_data(train)
```

Out[13]:

	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	...	var_
Total	0	0	0	0	0	0	0	0	0	0	...	
Percent	0	0	0	0	0	0	0	0	0	0	...	
Types	float16	float16	float16	float16	float16	float16	float16	float16	float16	float16	...	flo

3 rows × 200 columns

In [14]:

```
missing_data(test)
```

Out[14]:

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	...	va
Total	0	0	0	0	0	0	0	0	0	0	...	
Percent	0	0	0	0	0	0	0	0	0	0	...	
Types	category	float16	float16	float16	float16	float16	float16	float16	float16	float16	...	f

3 rows × 201 columns

We can notice that there is no missing values in both the Train and the Test Dataset

Performing EDA

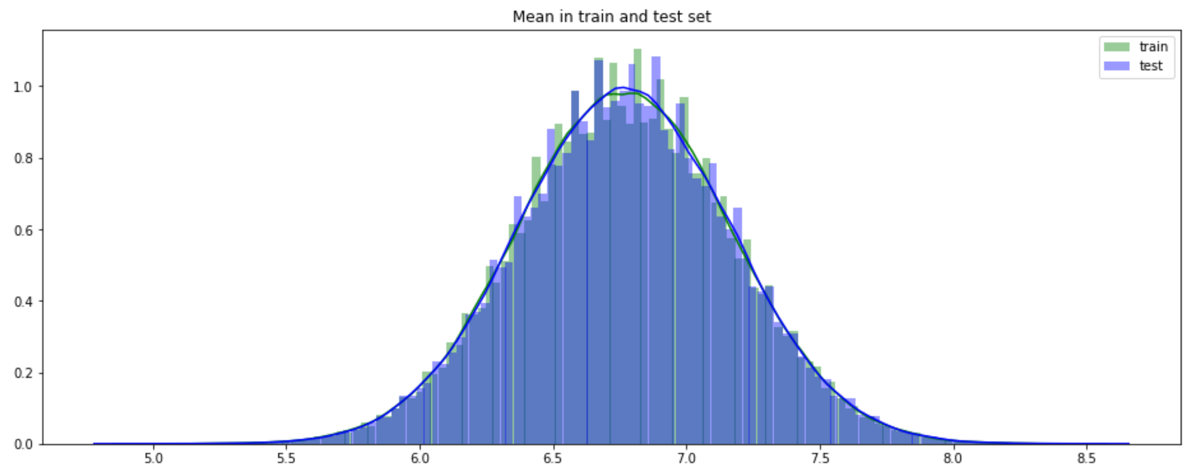
Mean

In [15]:

```
features = train.columns.tolist()
```

In [16]:

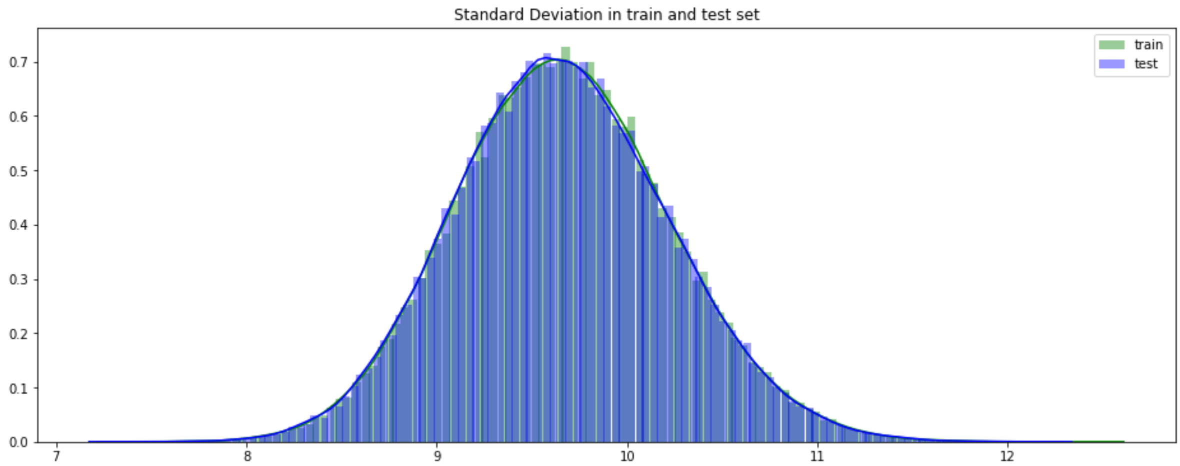
```
plt.figure(figsize=(16,6))
plt.title("Mean in train and test set")
sns.distplot(train[features].mean(axis=1), color="green", kde=True, bins=120, label='train'
sns.distplot(test[features].mean(axis=1), color="blue", kde=True, bins=120, label='test')
plt.legend()
plt.show()
```



Standard Deviation

In [17]:

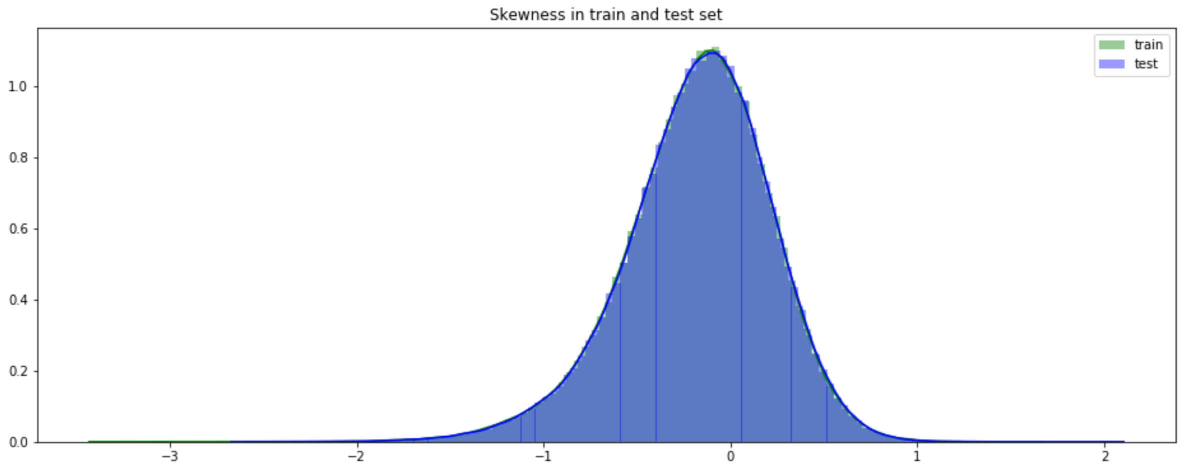
```
plt.figure(figsize=(16,6))
plt.title("Standard Deviation in train and test set")
sns.distplot(train[features].std(axis=1), color="green", kde=True, bins=120, label='train')
sns.distplot(test[features].std(axis=1), color="blue", kde=True, bins=120, label='test')
plt.legend()
plt.show()
```



Skewness

In [18]:

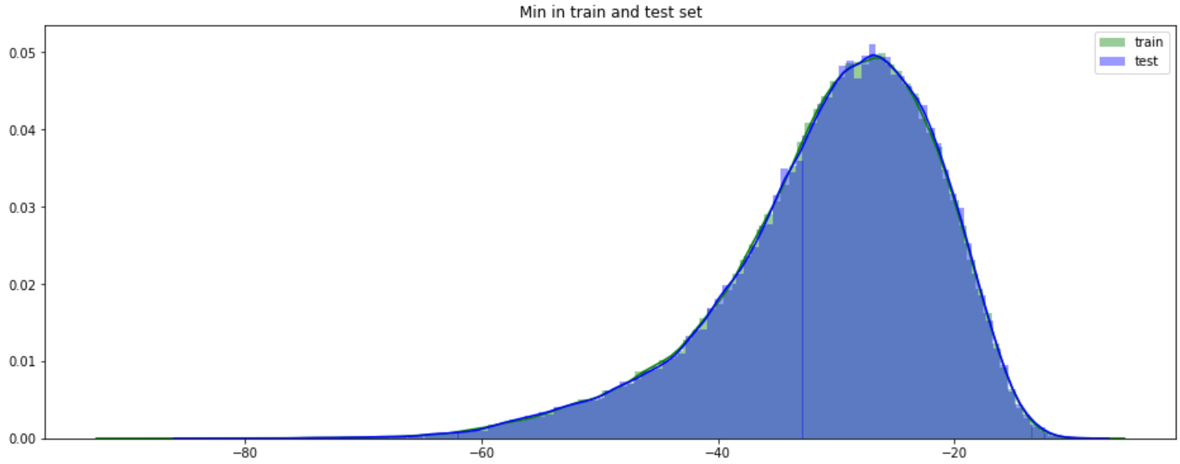
```
plt.figure(figsize=(16,6))
plt.title("Skewness in train and test set")
sns.distplot(train[features].skew(axis=1), color="green", kde=True, bins=120, label='train'
sns.distplot(test[features].skew(axis=1), color="blue", kde=True, bins=120, label='test')
plt.legend()
plt.show()
```



Min

In [19]:

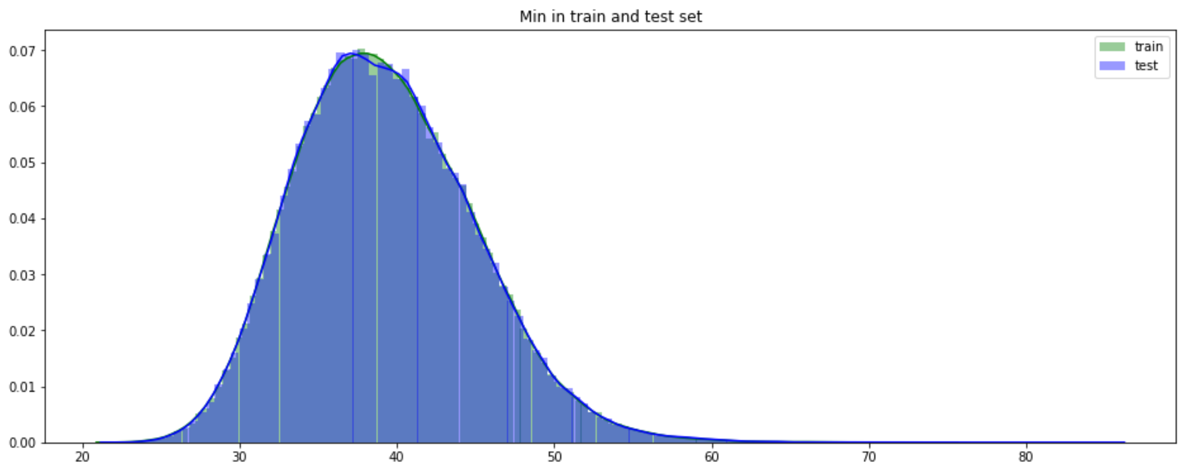
```
plt.figure(figsize=(16,6))
plt.title("Min in train and test set")
sns.distplot(train[features].min(axis=1), color="green", kde=True, bins=120, label='train')
sns.distplot(test[features].min(axis=1), color="blue", kde=True, bins=120, label='test')
plt.legend()
plt.show()
```



Max

In [20]:

```
plt.figure(figsize=(16,6))
plt.title("Min in train and test set")
sns.distplot(train[features].max(axis=1), color="green", kde=True, bins=120, label='train')
sns.distplot(test[features].max(axis=1), color="blue", kde=True, bins=120, label='test')
plt.legend()
plt.show()
```



Comparing Distribution of Feature

We can see from above that all the variables have nearly same distribution with the same scales

Duplicate Values

In [21]:

```
features = train.columns.values[2:202]
unique_max_train = []
unique_max_test = []
for feature in features:
    values = train[feature].value_counts()
    unique_max_train.append([feature, values.max(), values.idxmax()])
    values = test[feature].value_counts()
    unique_max_test.append([feature, values.max(), values.idxmax()])
```

In [22]:

```
np.transpose((pd.DataFrame(unique_max_train, columns=['Feature', 'Max duplicates', 'Value']
            sort_values(by = 'Max duplicates', ascending=False).head(15))
```

Out[22]:

	66	106	10	124	23	41	89	123	146	
Feature	var_68	var_108	var_12	var_126	var_25	var_43	var_91	var_125	var_148	var
Max duplicates	40233	6127	3221	2746	2087	1836	1811	1780	1578	1
Value	5.01953	14.2031	13.9766	11.5391	13.6875	11.5078	6.98438	12.5547	4.02344	14.

In [23]:

```
np.transpose((pd.DataFrame(unique_max_test, columns=['Feature', 'Max duplicates', 'Value'])
            sort_values(by = 'Max duplicates', ascending=False).head(15))
```

Out[23]:

	66	106	10	124	23	41	89	123	146	
Feature	var_68	var_108	var_12	var_126	var_25	var_43	var_91	var_125	var_148	va
Max duplicates	39964	5987	3164	2747	2116	1944	1848	1824	1617	
Value	5.01953	14.2031	13.9766	11.5391	13.6406	11.4609	7.03125	12.5391	4.00781	14.

Same columns in train and test set have the same or very close number of duplicates of same or very close values. This is an interesting pattern that we might be able to use in the future.

Feature Engineering

In [24]:

```
idx = features = train.columns.values[2:202]
for df in [test, train]:
    df['sum'] = df[idx].sum(axis=1)
    df['min'] = df[idx].min(axis=1)
    df['max'] = df[idx].max(axis=1)
    df['mean'] = df[idx].mean(axis=1)
    df['std'] = df[idx].std(axis=1)
    df['skew'] = df[idx].skew(axis=1)
    df['kurt'] = df[idx].kurtosis(axis=1)
    df['med'] = df[idx].median(axis=1)
```

In [25]:

```
train.head(2)
```

Out[25]:

	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_
0	8.921875	-6.785156	11.906250	5.093750	11.460938	-9.281250	5.117188	18.62500	-4.9218
1	11.500000	-4.148438	13.859375	5.390625	12.359375	7.042969	5.621094	16.53125	3.1464

2 rows × 208 columns

In [26]:

```
train.drop(['kurt'],axis=1,inplace=True)
```

In [27]:

```
train.head()
```

Out[27]:

	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	va
0	8.921875	-6.785156	11.906250	5.093750	11.460938	-9.281250	5.117188	18.625000	-4.921
1	11.500000	-4.148438	13.859375	5.390625	12.359375	7.042969	5.621094	16.531250	3.146
2	8.609375	-2.746094	12.078125	7.894531	10.585938	-9.085938	6.941406	14.617188	-4.917
3	11.062500	-2.152344	8.953125	7.195312	12.585938	-1.835938	5.843750	14.921875	-5.859
4	9.835938	-1.483398	12.875000	6.636719	12.273438	2.449219	5.941406	19.250000	6.265

5 rows × 207 columns

In [28]:

```
test.head(2)
```

Out[28]:

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7
0	test_0	11.06250	7.781250	12.953125	9.429688	11.429688	-2.380859	5.847656	18.265625
1	test_1	8.53125	1.253906	11.304688	5.187500	9.195312	-4.011719	6.019531	18.625000

2 rows × 209 columns

In [29]:

```
test.drop(['kurt','ID_code'],axis=1,inplace=True)
```

TSNE

In [30]:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
train_data = scaler.fit_transform(train)
```

In [31]:

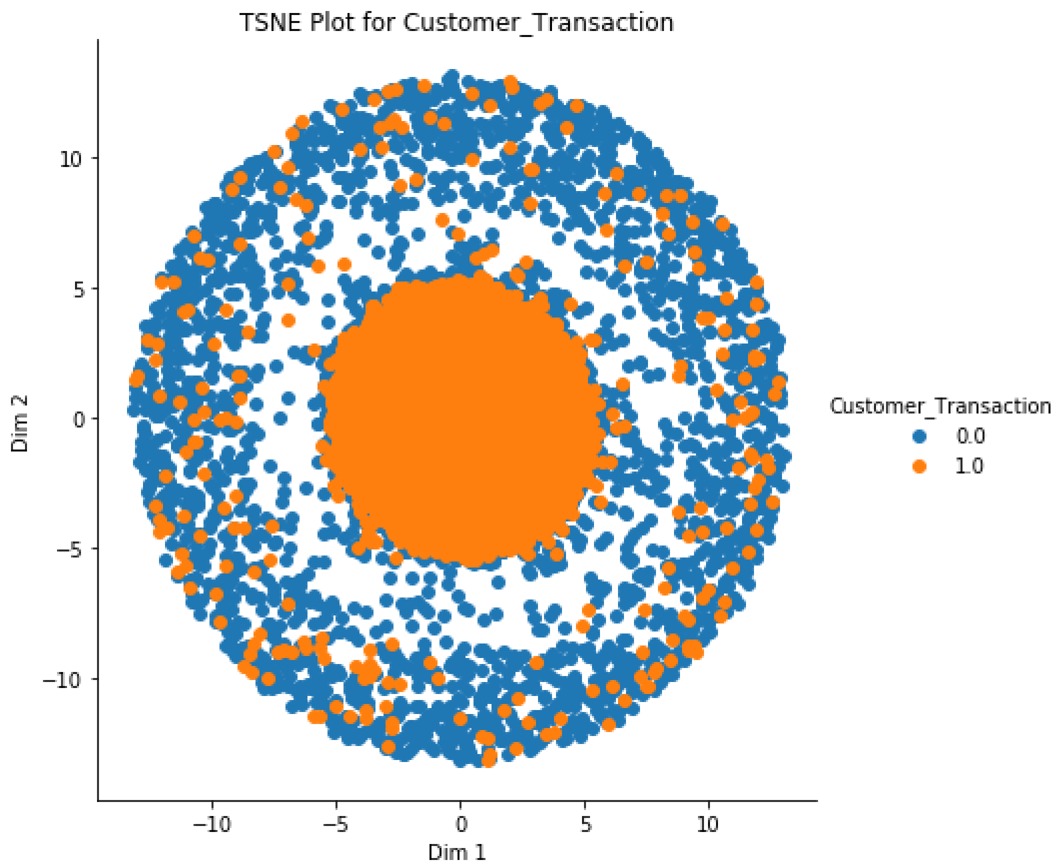
```
train_data.shape
```

Out[31]:

```
(200000, 207)
```

In [32]:

```
from sklearn.manifold import TSNE
model=TSNE(n_components=2 , random_state=0, perplexity=40, n_iter=1000)
tsne_data=model.fit_transform(train_data) # Calculation of TSNE
tsne_data=np.vstack((tsne_data.T,labels)).T
tsne_df=pd.DataFrame(data=tsne_data, columns=("Dim 1","Dim 2","Customer_Transaction"))
#plotting TSE
#labels=["Rejected","Accepted"]
sns.FacetGrid(tsne_df, hue="Customer_Transaction", size=6).map(plt.scatter, 'Dim 1', 'Dim 2
plt.title("TSNE Plot for Customer_Transaction")
plt.show()
```



As we can see from above the data cannot be separated using TSNE. The points are massively overlapped with positive points concentrated in the middle and the negative points surrounding it.

Modelling

Regression Model

Light GBM

In [33]:

```
!pip install lightgbm
```

```
Requirement already satisfied: lightgbm in c:\users\anirban\anaconda3\lib\si
te-packages (2.2.3)
Requirement already satisfied: scikit-learn in c:\users\anirban\anaconda3\li
b\site-packages (from lightgbm) (0.20.3)
Requirement already satisfied: numpy in c:\users\anirban\appdata\roaming\pyt
hon\python36\site-packages (from lightgbm) (1.16.4)
Requirement already satisfied: scipy in c:\users\anirban\anaconda3\lib\site-
packages (from lightgbm) (1.2.1)
```

In [34]:

```
import lightgbm as lgb
```

In [35]:

```
param = {
    'bagging_freq': 5,
    'bagging_fraction': 0.4,
    'boost_from_average':'false',
    'boost': 'gbdt',
    'feature_fraction': 0.05,
    'learning_rate': 0.01,
    'max_depth': -1,
    'metric':'auc',
    'min_data_in_leaf': 80,
    'min_sum_hessian_in_leaf': 10.0,
    'num_leaves': 13,
    'num_threads': 8,
    'tree_learner': 'serial',
    'objective': 'binary',
    'verbosity': 1
}
```

In [37]:

```
#https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.train.html#lightgbm.train
#https://www.kaggle.com/ashishpatel26/kfold-lightgbm/code
#(learned from here how to use stratified k-fold with model)
#https://github.com/KazukiOnodera/Santander-Customer-Transaction-Prediction/blob/master/fin

from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_auc_score, roc_curve
folds = StratifiedKFold(n_splits=10, shuffle=False, random_state=44000)
oof = np.zeros(len(train))
predictions = np.zeros(len(test))
feature_importance_df = pd.DataFrame()

for fold_, (trn_idx, val_idx) in enumerate(folds.split(train.values, labels.values)):
    print("Fold {}".format(fold_))
    trn_data = lgb.Dataset(train.iloc[trn_idx][features], label=labels.iloc[trn_idx])
    val_data = lgb.Dataset(train.iloc[val_idx][features], label=labels.iloc[val_idx])

    num_round = 1000000
    clf = lgb.train(param, trn_data, num_round, valid_sets = [trn_data, val_data], verbose_
    oof[val_idx] = clf.predict(train.iloc[val_idx][features], num_iteration=clf.best_iterat

    fold_importance_df = pd.DataFrame()
    fold_importance_df["Feature"] = features
    fold_importance_df["importance"] = clf.feature_importance()
    fold_importance_df["fold"] = fold_ + 1
    feature_importance_df = pd.concat([feature_importance_df, fold_importance_df], axis=0)

    predictions += clf.predict(test[features], num_iteration=clf.best_iteration) / folds.n_

print("CV score: {:<8.5f}".format(roc_auc_score(labels, oof)))
```

```
Fold 0
Training until validation scores don't improve for 3000 rounds.
[1000]  training's auc: 0.898416        valid_1's auc: 0.879423
[2000]  training's auc: 0.909095        valid_1's auc: 0.886415
[3000]  training's auc: 0.916569        valid_1's auc: 0.890816
[4000]  training's auc: 0.922256        valid_1's auc: 0.893047
[5000]  training's auc: 0.926935        valid_1's auc: 0.894655
[6000]  training's auc: 0.931187        valid_1's auc: 0.895585
[7000]  training's auc: 0.935173        valid_1's auc: 0.896432
[8000]  training's auc: 0.938821        valid_1's auc: 0.896624
[9000]  training's auc: 0.942428        valid_1's auc: 0.896811
[10000] training's auc: 0.945787        valid_1's auc: 0.896703
[11000] training's auc: 0.949007        valid_1's auc: 0.896715
[12000] training's auc: 0.952091        valid_1's auc: 0.896705
Early stopping, best iteration is:
[9295]  training's auc: 0.94345 valid_1's auc: 0.896898
Fold 1
Training until validation scores don't improve for 3000 rounds.
[1000]  training's auc: 0.898197        valid_1's auc: 0.880849
[2000]  training's auc: 0.908856        valid_1's auc: 0.888719
[3000]  training's auc: 0.916275        valid_1's auc: 0.892485
[4000]  training's auc: 0.921915        valid_1's auc: 0.895032
[5000]  training's auc: 0.926672        valid_1's auc: 0.896269
[6000]  training's auc: 0.930977        valid_1's auc: 0.897043
[7000]  training's auc: 0.934918        valid_1's auc: 0.897488
```

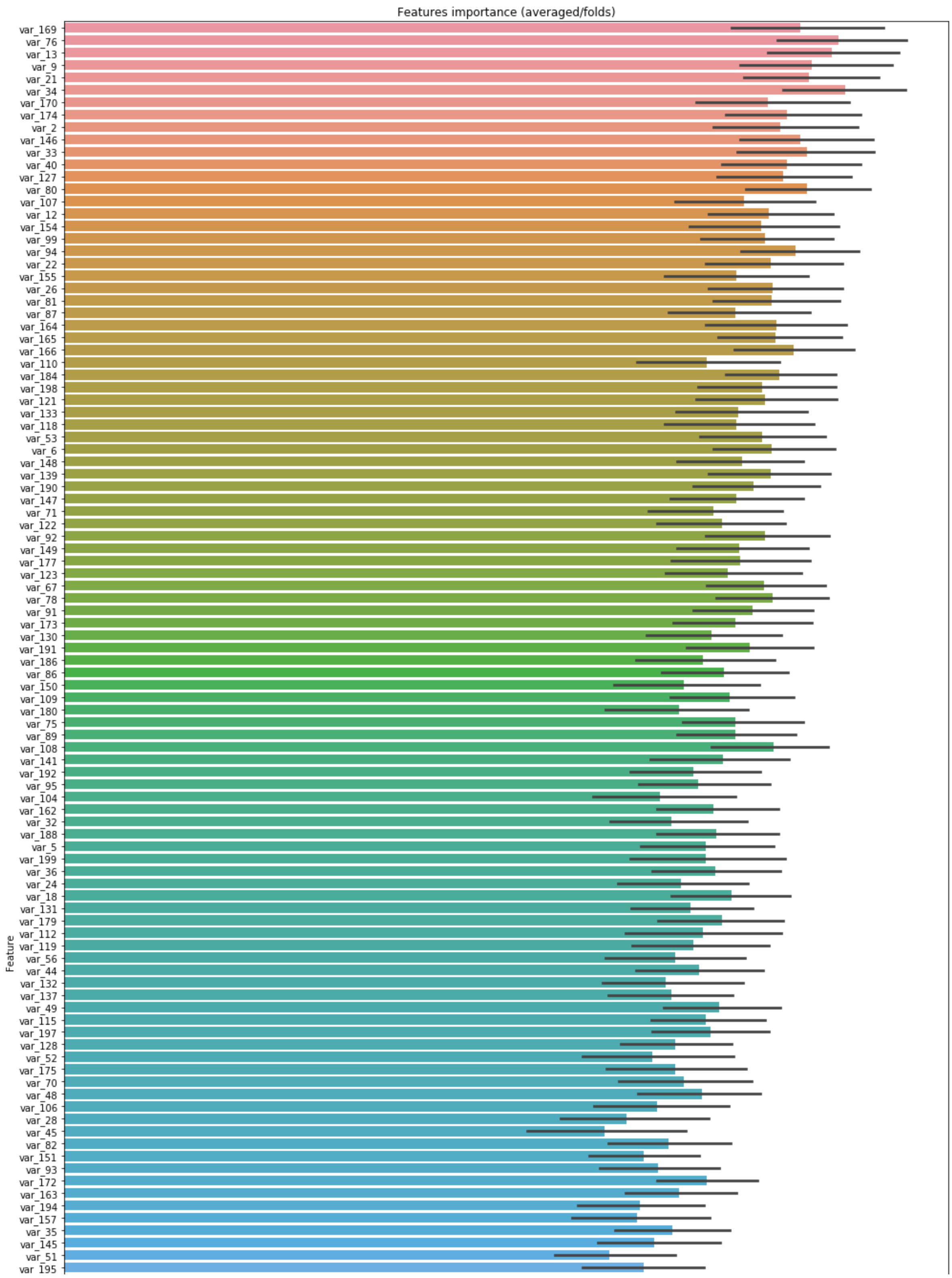
```
[8000]  training's auc: 0.938576        valid_1's auc: 0.897793
[9000]  training's auc: 0.942212        valid_1's auc: 0.897983
[10000] training's auc: 0.945579        valid_1's auc: 0.897846
[11000] training's auc: 0.948802        valid_1's auc: 0.897674
Early stopping, best iteration is:
[8671]  training's auc: 0.941043        valid_1's auc: 0.898076
Fold 2
Training until validation scores don't improve for 3000 rounds.
[1000]  training's auc: 0.898698        valid_1's auc: 0.874817
[2000]  training's auc: 0.909551        valid_1's auc: 0.882419
[3000]  training's auc: 0.916888        valid_1's auc: 0.886291
[4000]  training's auc: 0.922455        valid_1's auc: 0.88898
[5000]  training's auc: 0.927235        valid_1's auc: 0.89033
[6000]  training's auc: 0.931533        valid_1's auc: 0.891383
[7000]  training's auc: 0.935447        valid_1's auc: 0.891797
[8000]  training's auc: 0.93912 valid_1's auc: 0.891946
[9000]  training's auc: 0.942637        valid_1's auc: 0.891871
[10000] training's auc: 0.945984        valid_1's auc: 0.891904
[11000] training's auc: 0.949188        valid_1's auc: 0.892048
[12000] training's auc: 0.952309        valid_1's auc: 0.892106
[13000] training's auc: 0.955278        valid_1's auc: 0.892128
[14000] training's auc: 0.958132        valid_1's auc: 0.892199
[15000] training's auc: 0.96092 valid_1's auc: 0.891831
[16000] training's auc: 0.963544        valid_1's auc: 0.891542
Early stopping, best iteration is:
[13501] training's auc: 0.956716        valid_1's auc: 0.89222
Fold 3
Training until validation scores don't improve for 3000 rounds.
[1000]  training's auc: 0.898074        valid_1's auc: 0.879978
[2000]  training's auc: 0.908974        valid_1's auc: 0.887306
[3000]  training's auc: 0.916442        valid_1's auc: 0.891179
[4000]  training's auc: 0.922096        valid_1's auc: 0.893237
[5000]  training's auc: 0.926876        valid_1's auc: 0.894588
[6000]  training's auc: 0.931224        valid_1's auc: 0.895011
[7000]  training's auc: 0.935145        valid_1's auc: 0.895287
[8000]  training's auc: 0.938871        valid_1's auc: 0.895565
[9000]  training's auc: 0.942427        valid_1's auc: 0.895676
[10000] training's auc: 0.945821        valid_1's auc: 0.895622
[11000] training's auc: 0.949014        valid_1's auc: 0.895412
[12000] training's auc: 0.95214 valid_1's auc: 0.895332
Early stopping, best iteration is:
[9015]  training's auc: 0.942479        valid_1's auc: 0.895704
Fold 4
Training until validation scores don't improve for 3000 rounds.
[1000]  training's auc: 0.897869        valid_1's auc: 0.880826
[2000]  training's auc: 0.908624        valid_1's auc: 0.88775
[3000]  training's auc: 0.916318        valid_1's auc: 0.89196
[4000]  training's auc: 0.922001        valid_1's auc: 0.894451
[5000]  training's auc: 0.926837        valid_1's auc: 0.895422
[6000]  training's auc: 0.93124 valid_1's auc: 0.89594
[7000]  training's auc: 0.935152        valid_1's auc: 0.896049
[8000]  training's auc: 0.938868        valid_1's auc: 0.895956
[9000]  training's auc: 0.942355        valid_1's auc: 0.895889
[10000] training's auc: 0.945694        valid_1's auc: 0.895764
Early stopping, best iteration is:
[7197]  training's auc: 0.935881        valid_1's auc: 0.896168
Fold 5
Training until validation scores don't improve for 3000 rounds.
[1000]  training's auc: 0.897321        valid_1's auc: 0.884385
[2000]  training's auc: 0.908219        valid_1's auc: 0.891861
[3000]  training's auc: 0.915943        valid_1's auc: 0.895536
```

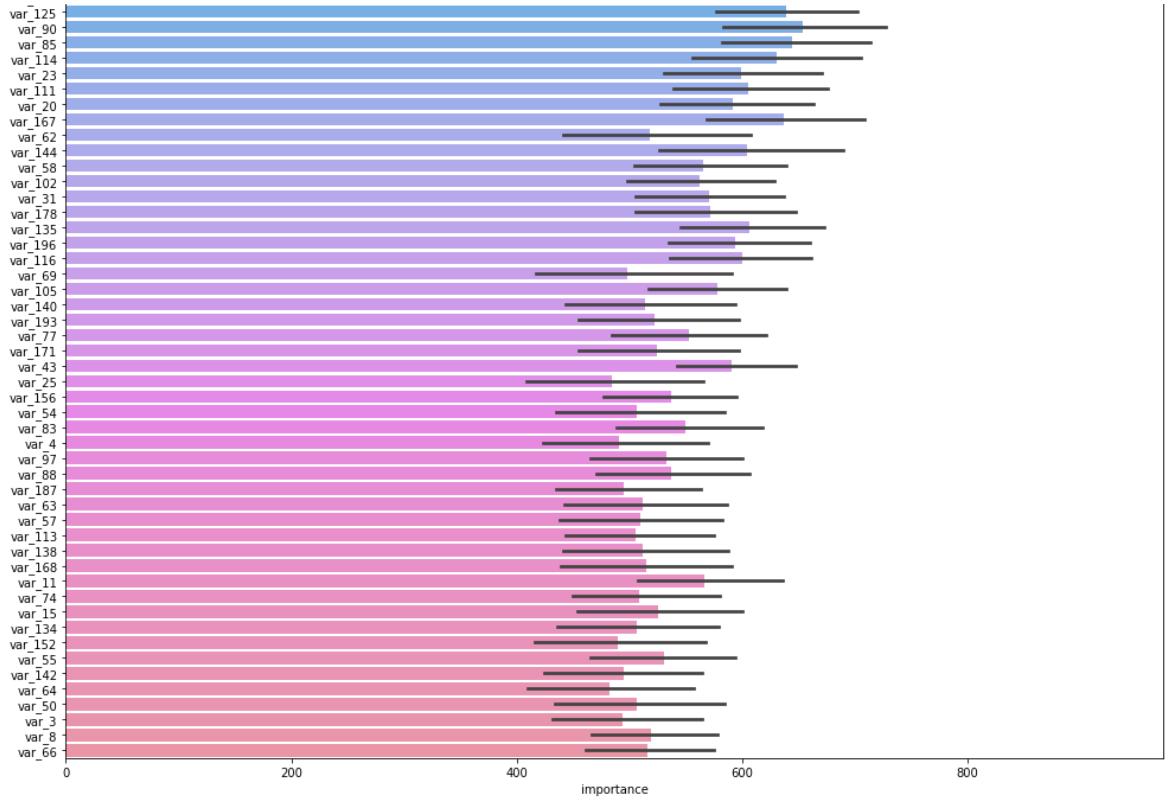
```
[4000]  training's auc: 0.921697        valid_1's auc: 0.897753
[5000]  training's auc: 0.926515        valid_1's auc: 0.898986
[6000]  training's auc: 0.93088 valid_1's auc: 0.899574
[7000]  training's auc: 0.934857        valid_1's auc: 0.899988
[8000]  training's auc: 0.938543        valid_1's auc: 0.900122
[9000]  training's auc: 0.942113        valid_1's auc: 0.90001
[10000] training's auc: 0.945446        valid_1's auc: 0.900061
Early stopping, best iteration is:
[7794]  training's auc: 0.937813        valid_1's auc: 0.900173
Fold 6
Training until validation scores don't improve for 3000 rounds.
[1000]  training's auc: 0.898039        valid_1's auc: 0.882774
[2000]  training's auc: 0.908613        valid_1's auc: 0.889829
[3000]  training's auc: 0.916228        valid_1's auc: 0.894245
[4000]  training's auc: 0.921928        valid_1's auc: 0.896211
[5000]  training's auc: 0.926702        valid_1's auc: 0.897454
[6000]  training's auc: 0.931064        valid_1's auc: 0.898198
[7000]  training's auc: 0.935019        valid_1's auc: 0.89841
[8000]  training's auc: 0.938748        valid_1's auc: 0.898878
[9000]  training's auc: 0.942362        valid_1's auc: 0.898877
[10000] training's auc: 0.945688        valid_1's auc: 0.898932
[11000] training's auc: 0.948935        valid_1's auc: 0.899101
[12000] training's auc: 0.952032        valid_1's auc: 0.898925
[13000] training's auc: 0.955029        valid_1's auc: 0.898734
[14000] training's auc: 0.95785 valid_1's auc: 0.898607
Early stopping, best iteration is:
[11453] training's auc: 0.950384        valid_1's auc: 0.899167
Fold 7
Training until validation scores don't improve for 3000 rounds.
[1000]  training's auc: 0.898184        valid_1's auc: 0.877027
[2000]  training's auc: 0.908809        valid_1's auc: 0.886122
[3000]  training's auc: 0.91632 valid_1's auc: 0.890461
[4000]  training's auc: 0.921962        valid_1's auc: 0.893008
[5000]  training's auc: 0.92679 valid_1's auc: 0.89456
[6000]  training's auc: 0.931147        valid_1's auc: 0.895747
[7000]  training's auc: 0.935056        valid_1's auc: 0.896056
[8000]  training's auc: 0.938739        valid_1's auc: 0.896521
[9000]  training's auc: 0.942309        valid_1's auc: 0.896645
[10000] training's auc: 0.945717        valid_1's auc: 0.896822
[11000] training's auc: 0.948954        valid_1's auc: 0.896833
[12000] training's auc: 0.952075        valid_1's auc: 0.896886
[13000] training's auc: 0.955046        valid_1's auc: 0.896669
[14000] training's auc: 0.957926        valid_1's auc: 0.896623
Early stopping, best iteration is:
[11895] training's auc: 0.951758        valid_1's auc: 0.896924
Fold 8
Training until validation scores don't improve for 3000 rounds.
[1000]  training's auc: 0.898088        valid_1's auc: 0.885564
[2000]  training's auc: 0.908586        valid_1's auc: 0.893243
[3000]  training's auc: 0.916126        valid_1's auc: 0.896857
[4000]  training's auc: 0.921795        valid_1's auc: 0.898947
[5000]  training's auc: 0.926536        valid_1's auc: 0.899975
[6000]  training's auc: 0.930824        valid_1's auc: 0.900483
[7000]  training's auc: 0.934771        valid_1's auc: 0.900676
[8000]  training's auc: 0.938519        valid_1's auc: 0.900848
[9000]  training's auc: 0.942027        valid_1's auc: 0.900899
[10000] training's auc: 0.945489        valid_1's auc: 0.900989
[11000] training's auc: 0.948704        valid_1's auc: 0.901046
[12000] training's auc: 0.951867        valid_1's auc: 0.901043
[13000] training's auc: 0.9549  valid_1's auc: 0.900809
Early stopping, best iteration is:
```

```
[10488] training's auc: 0.947108        valid_1's auc: 0.901166
Fold 9
Training until validation scores don't improve for 3000 rounds.
[1000]  training's auc: 0.898214        valid_1's auc: 0.883317
[2000]  training's auc: 0.908928        valid_1's auc: 0.890293
[3000]  training's auc: 0.916448        valid_1's auc: 0.893621
[4000]  training's auc: 0.922226        valid_1's auc: 0.895815
[5000]  training's auc: 0.926962        valid_1's auc: 0.897078
[6000]  training's auc: 0.931292        valid_1's auc: 0.897827
[7000]  training's auc: 0.935287        valid_1's auc: 0.898296

[8000]  training's auc: 0.938976        valid_1's auc: 0.898245
[9000]  training's auc: 0.942555        valid_1's auc: 0.898286
[10000] training's auc: 0.946015        valid_1's auc: 0.898298
[11000] training's auc: 0.949268        valid_1's auc: 0.89821
[12000] training's auc: 0.952388        valid_1's auc: 0.898177
Early stopping, best iteration is:
[9874]  training's auc: 0.945576        valid_1's auc: 0.898367
CV score: 0.89728
```

In [38]:

```
cols = (feature_importance_df[["Feature", "importance"]]
        .groupby("Feature")
        .mean()
        .sort_values(by="importance", ascending=False)[:150].index)
best_features = feature_importance_df.loc[feature_importance_df.Feature.isin(cols)]

plt.figure(figsize=(14,28))
sns.barplot(x="importance", y="Feature", data=best_features.sort_values(by="importance",asc
plt.title('Features importance (averaged/folds)')
plt.tight_layout()
plt.savefig('FI.png')
```





In [41]:

```
sub_df =  pd.DataFrame({"ID_code":test["ID_code"].values})
sub_df["target"] = predictions
sub_df.to_csv("lgbm.csv", index=False)
```

In [42]:

```
lgbm=pd.read_csv('lgbm.csv')
lgbm.head()
```

Out[42]:

	ID_code	target
0	test_0	0.054296
1	test_1	0.214405
2	test_2	0.210391
3	test_3	0.220053
4	test_4	0.042658

Classification Model

1. Logistic Regression

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
from random import randrange, uniform
from scipy.stats import chi2_contingency
%matplotlib inline
```

In [2]:

```
trans = pd.read_csv("train.csv")
```

Detect and delete outliers from data

In [3]:

```
for i in range(2,202):
        #print(i)
        q75, q25 = np.percentile(trans.iloc[:,i], [75 ,25])
        iqr = q75 - q25

        min = q25 - (iqr*1.5)
        max = q75 + (iqr*1.5)
        #print(min)
        #print(max)

        trans = trans.drop(trans[trans.iloc[:,i] < min].index)
        trans = trans.drop(trans[trans.iloc[:,i] > max].index)
```

In [4]:

```
trans.shape
```

Out[4]:

```
(175073, 202)
```

In [5]:

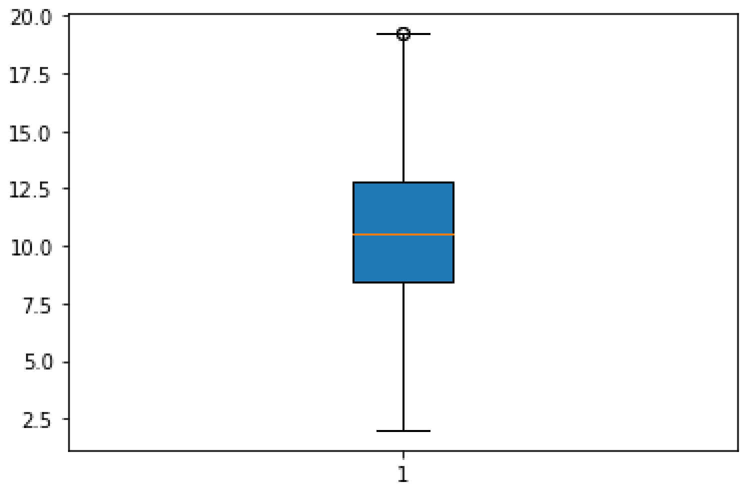
```
trans.to_csv("outlier values.csv")
```

In [6]:

```
plt.boxplot(trans['var_0'] ,vert=True,patch_artist=True)
```

Out[6]:

```
{'whiskers': [<matplotlib.lines.Line2D at 0x1ea8f073908>,
  <matplotlib.lines.Line2D at 0x1ea8f073a58>],
 'caps': [<matplotlib.lines.Line2D at 0x1ea8f073da0>,
  <matplotlib.lines.Line2D at 0x1ea8f29c128>],
 'boxes': [<matplotlib.patches.PathPatch at 0x1ea8f0736d8>],
 'medians': [<matplotlib.lines.Line2D at 0x1ea8f29c470>],
 'fliers': [<matplotlib.lines.Line2D at 0x1eac8070080>],
 'means': []}
```



In [7]:

```
trans = trans.drop(trans.columns[0], axis = 1)
```

In [8]:

```
from  sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(trans.drop('target',axis=1),
                                                    trans['target'], test_size=0.30,
                                                    random_state=101)
```

In [9]:

```
print(x_train.shape)
print(x_test.shape)
```

```
(122551, 200)
(52522, 200)
```

In [10]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from math import log
```

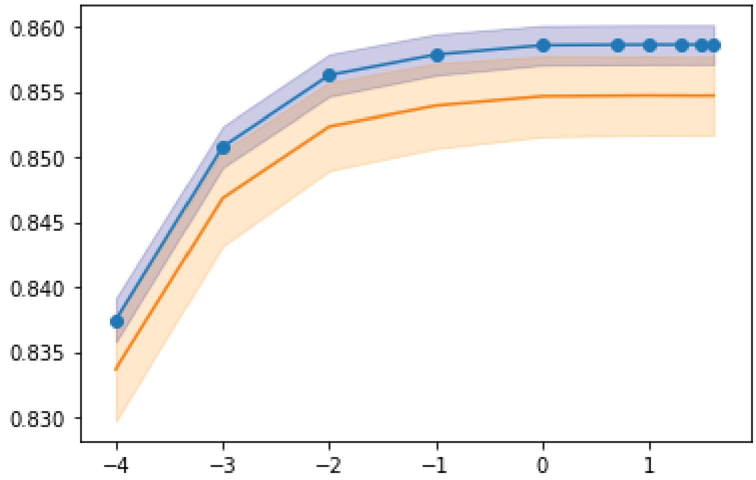
In [11]:

```
from sklearn.linear_model import LogisticRegression

import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import RandomizedSearchCV

C = LogisticRegression()

import math

parameter_data = [0.0001,0.001,0.01,0.1,1,5,10,20,30,40]

log_my_data = [math.log10(x) for x in parameter_data]

#print(log_my_data)
print("Printing parameter Data and Corresponding Log value")
data={'Parameter value':parameter_data,'Corresponding Log Value':log_my_data}
param=pd.DataFrame(data)
print("="*100)
print(param)
parameters = {'C':parameter_data}
clf = RandomizedSearchCV(C, parameters, cv=3, scoring='roc_auc', return_train_score=True, r
clf.fit(x_train, y_train)

#data={'Parameter value':[0.0001,0.001,0.01,0.1,1,5,10,20,30,40],'Corresponding Log Value':

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(log_my_data, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_my_data,train_auc - train_auc_std,train_auc + train_auc_std,alph

plt.plot(log_my_data, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_my_data,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color=

plt.scatter(log_my_data, train_auc, label='Train AUC points')
```

```
Printing parameter Data and Corresponding Log value
============================================================================
========================
   Parameter value  Corresponding Log Value
0           0.0001                -4.000000
1           0.0010                -3.000000
2           0.0100                -2.000000
3           0.1000                -1.000000
4           1.0000                 0.000000
5           5.0000                 0.698970
6          10.0000                 1.000000
7          20.0000                 1.301030
8          30.0000                 1.477121
9          40.0000                 1.602060
```

Out[11]:

```
<matplotlib.collections.PathCollection at 0x1ea8f06d320>
```

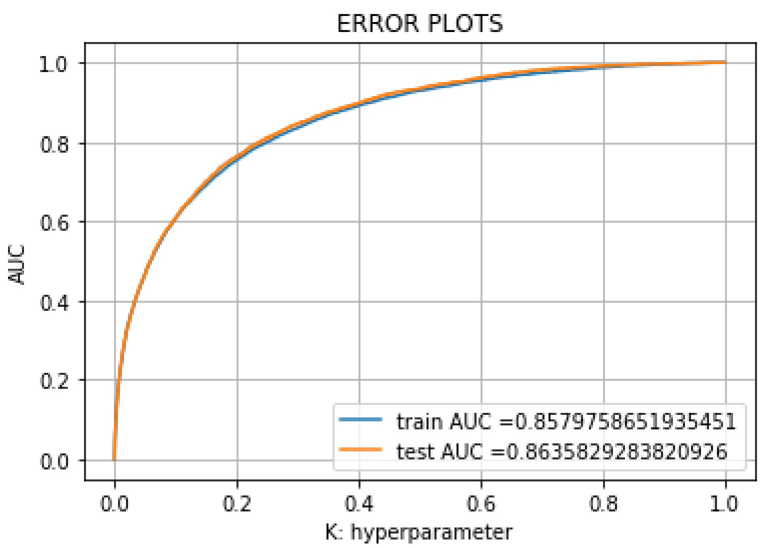


In [14]:

```python
def model_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs

    y_data_pred = []
    y_data_pred.extend(clf.predict_proba(data[:])[:,1])

    return y_data_pred
```

In [23]:

```
from sklearn.metrics import roc_curve, auc


neigh = LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
neigh.fit(x_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_pred = model_predict(neigh, x_train)
y_test_pred = model_predict(neigh, x_test)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [41]:

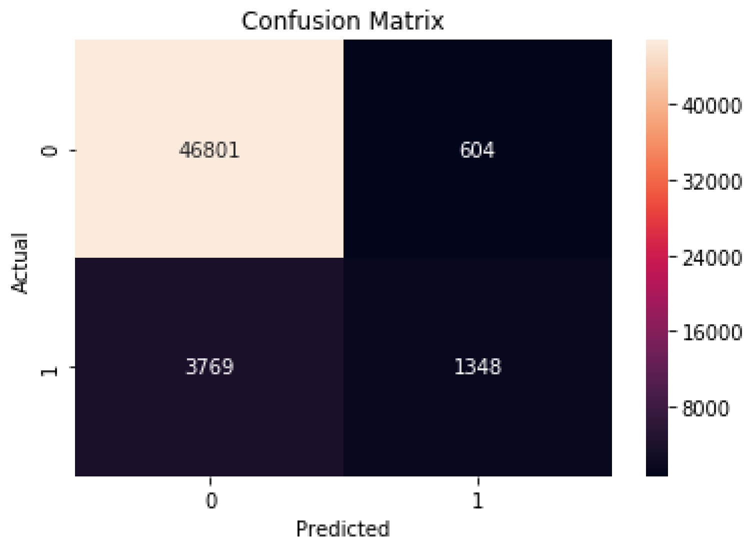
```
from sklearn.metrics import confusion_matrix
CM = confusion_matrix(y_test, y_test_pred)
CM = pd.crosstab(y_test, y_test_pred)

#let us save TP, TN, FP, FN
TN = CM.iloc[0,0]
FN = CM.iloc[1,0]
TP = CM.iloc[1,1]
FP = CM.iloc[0,1]
```

In [43]:

```
sns.heatmap(CM, annot=True, fmt="d" )
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title("Confusion Matrix")
```

Out[43]:

```
Text(0.5, 1, 'Confusion Matrix')
```



In [44]:

```
test =pd.read_csv("test.csv")
```

In [45]:

```
id_code = test.iloc[:,0]
```

In [46]:

```
test = test.drop("ID_code" ,axis=1)
predictions_test = neigh.predict(test)
df = pd.DataFrame({"ID_code" :id_code ,"target": predictions_test})
df.head()
```

Out[46]:

	ID_code	target
0	test_0	0
1	test_1	0
2	test_2	0
3	test_3	0
4	test_4	0

In [47]:

```
test_logistic = df.join(test)
test_logistic.to_csv('logisticmodelpred.csv')
test_logistic.head()
```

Out[47]:

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	...	va
0	test_0	0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	...	-:
1	test_1	0	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	...	1(
2	test_2	0	5.4827	-10.3581	10.1407	7.0479	10.2628	9.8052	4.8950	20.2537	...	-(
3	test_3	0	8.5374	-1.3222	12.0220	6.5749	8.8458	3.1744	4.9397	20.5660	...	!
4	test_4	0	11.7058	-0.1327	14.1295	7.7506	9.1035	-8.5848	6.8595	10.6048	...	‹

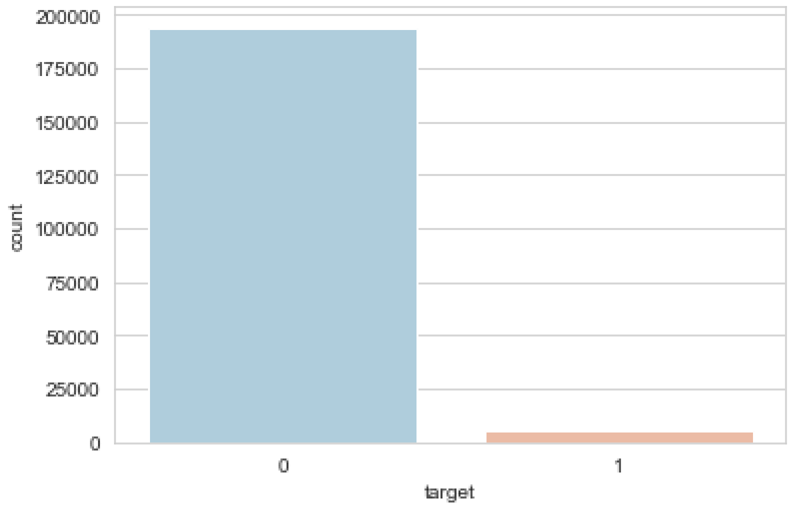
5 rows × 202 columns

In [48]:

```
sns.set_style('whitegrid')
sns.countplot(x='target',data=test_logistic,palette='RdBu_r')
test_logistic['target'].value_counts()
```

Out[48]:

```
0    194098
1      5902
Name: target, dtype: int64
```



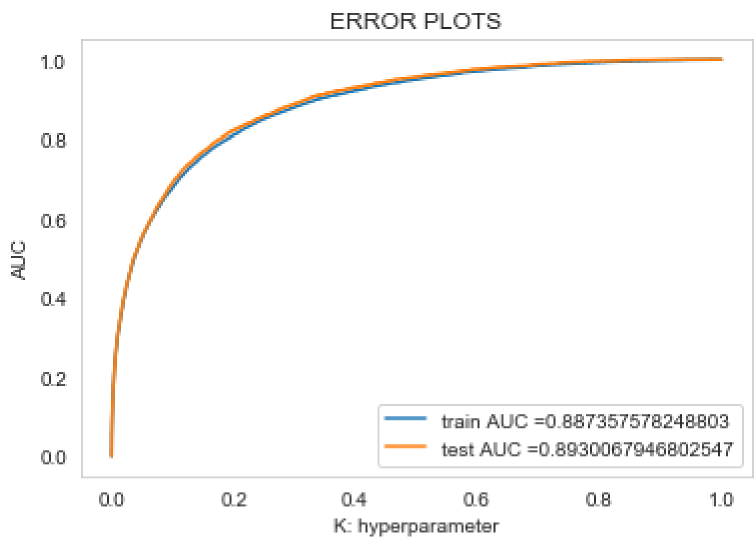
Naive Bayes

In [49]:

```
from sklearn.naive_bayes import GaussianNB
```

In [58]:

```
neigh = GaussianNB()
neigh.fit(x_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_pred = model_predict(neigh, x_train)
y_test_pred = model_predict(neigh, x_test)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

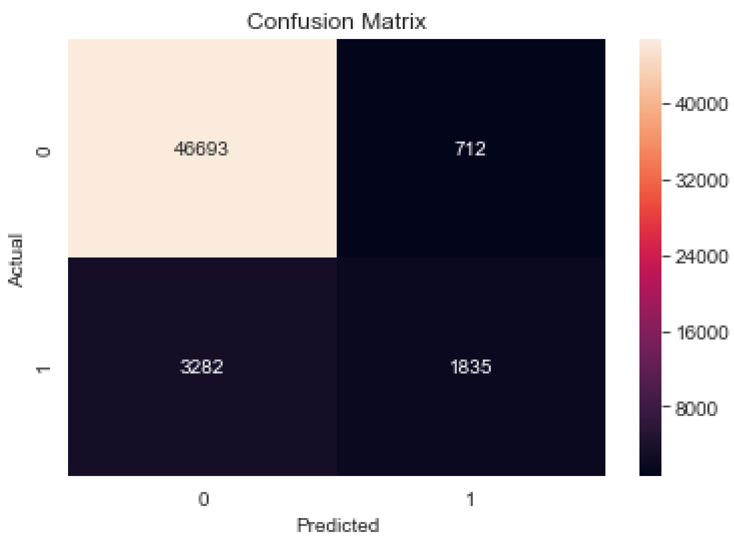


In [63]:

```
CM = confusion_matrix(y_test, y_test_pred)
CM = pd.crosstab(y_test, y_test_pred)

#let us save TP, TN, FP, FN
TN = CM.iloc[0,0]
FN = CM.iloc[1,0]
TP = CM.iloc[1,1]
FP = CM.iloc[0,1]
sns.heatmap(CM, annot=True, fmt="d" )
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title("Confusion Matrix")
```

Out[63]:

```
Text(0.5, 1, 'Confusion Matrix')
```



In [65]:

```
predictions_test = neigh.predict(test)
```

In [67]:

```
df = pd.DataFrame({"ID_code" :id_code ,"target": predictions_test})
df.head()
```

Out[67]:

	ID_code	target
0	test_0	0
1	test_1	0
2	test_2	0
3	test_3	0
4	test_4	0

In [68]:

```
test_nb = df.join(test)
```

In [69]:

```
test_nb.head(2)
```

Out[69]:

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	...	var_
0	test_0	0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	...	-2.1
1	test_1	0	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	...	10.6

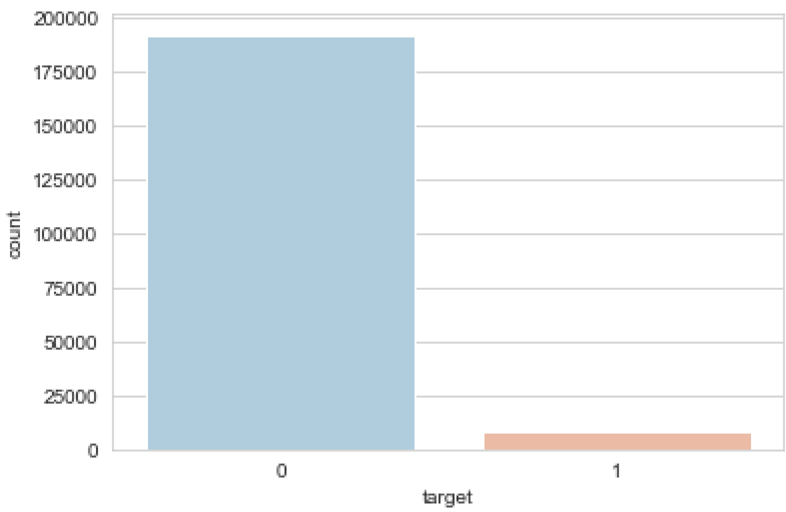
2 rows × 202 columns

In [70]:

```
sns.set_style('whitegrid')
sns.countplot(x='target',data=test_nb,palette='RdBu_r')
test_nb['target'].value_counts()
```

Out[70]:

```
0    192096
1      7904
Name: target, dtype: int64
```



In [71]:

```
test_nb.to_csv('Naive Bayes Prediction.csv')
```

Random Forest

In [73]:

```
from sklearn.ensemble import RandomForestClassifier

# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.ht
from sklearn.model_selection import GridSearchCV

C = RandomForestClassifier()

n_estimators=[10,50,100,200]
max_depth=[1, 5, 10, 50]

import math

log_max_depth = [math.log10(x) for x in max_depth]
log_n_estimators=[math.log10(x) for x in n_estimators]

print("Printing parameter Data and Corresponding Log value for Max Depth")
data={'Parameter value':max_depth,'Corresponding Log Value':log_max_depth}
param=pd.DataFrame(data)
print("="*100)
print(param)

print("Printing parameter Data and Corresponding Log value for Estimators")
data={'Parameter value':n_estimators,'Corresponding Log Value':log_n_estimators}
param=pd.DataFrame(data)
print("="*100)
print(param)

parameters = {'n_estimators':n_estimators, 'max_depth':max_depth}
clf = GridSearchCV(C, parameters, cv=3, scoring='roc_auc', return_train_score=True,n_jobs=-
clf.fit(x_train, y_train)

#data={'Parameter value':[0.0001,0.001,0.01,0.1,1,5,10,20,30,40],'Corresponding Log Value':

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

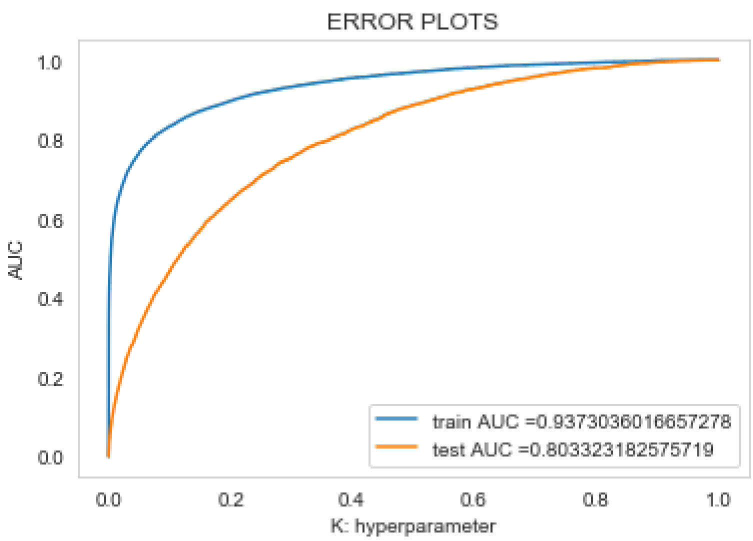
```
Printing parameter Data and Corresponding Log value for Max Depth
============================================================================
========================
   Parameter value  Corresponding Log Value
0                1                  0.00000
1                5                  0.69897
2               10                  1.00000
3               50                  1.69897
Printing parameter Data and Corresponding Log value for Estimators
============================================================================
========================
   Parameter value  Corresponding Log Value
0               10                  1.00000
1               50                  1.69897
2              100                  2.00000
3              200                  2.30103
```

In [74]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=log_n_estimators, y=log_max_depth, z=train_auc, name = 'train')
trace2 = go.Scatter3d(x=log_n_estimators, y=log_max_depth, z=cv_auc, name = 'Cross validati
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='n_estimators'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [76]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.
from sklearn.metrics import roc_curve, auc
#from sklearn.calibration import CalibratedClassifierCV


neigh = RandomForestClassifier(n_estimators=100,max_depth=10,class_weight='balanced')
neigh.fit(x_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_pred = model_predict(neigh, x_train)
y_test_pred = model_predict(neigh, x_test)


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

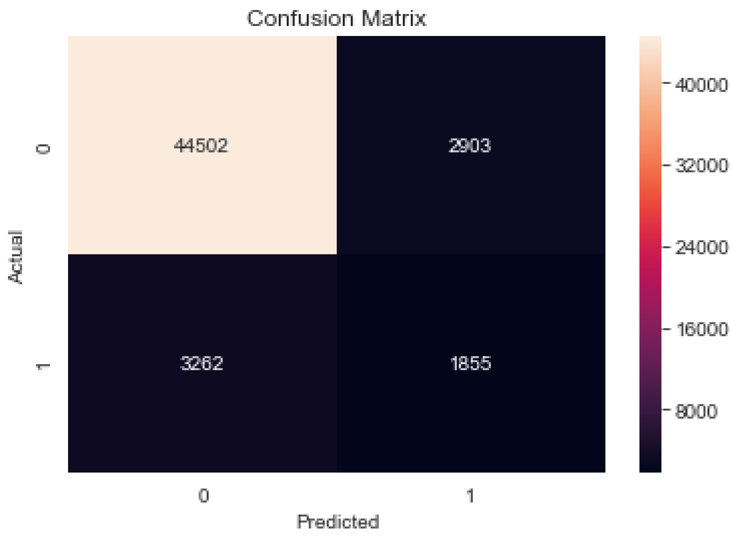


In [78]:

```
CM = confusion_matrix(y_test, y_test_pred)
CM = pd.crosstab(y_test, y_test_pred)

#let us save TP, TN, FP, FN
TN = CM.iloc[0,0]
FN = CM.iloc[1,0]
TP = CM.iloc[1,1]
FP = CM.iloc[0,1]
sns.heatmap(CM, annot=True, fmt="d" )
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title("Confusion Matrix")
```

Out[78]:

```
Text(0.5, 1, 'Confusion Matrix')
```



In [79]:

```
predictions_rfc = neigh.predict(test)
```

In [80]:

```
df = pd.DataFrame({"ID_code" :id_code ,"target": predictions_rfc})
df.head()
```

Out[80]:

	ID_code	target
0	test_0	1
1	test_1	0
2	test_2	0
3	test_3	0
4	test_4	0

In [81]:

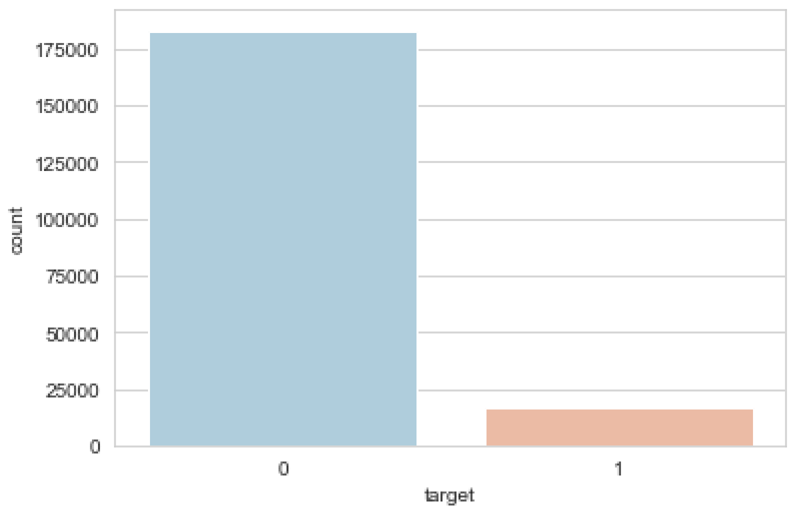
```
test_rfc = df.join(test)
```

In [82]:

```
sns.set_style('whitegrid')
sns.countplot(x='target',data=test_rfc,palette='RdBu_r')
test_rfc['target'].value_counts()
```

Out[82]:

```
0    182954
1     17046
Name: target, dtype: int64
```



In [83]:

```
test_rfc.to_csv('RandomForestPrediction.csv')
```