Santander Bank Customer Transaction Prediction Model

Santander Bank 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.

Content

In [1]:
```python
import numpy as np
import pandas as pd
import lightgbm as lgb
import matplotlib
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold,KFold
import warnings
from six.moves import urllib
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
warnings.filterwarnings('ignore')
%matplotlib inline
plt.style.use('seaborn')
from scipy.stats import norm, skew
```

Import the Data

In [2]:
```python
#Load the Data
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
features = [c for c in train.columns if c not in ['ID_code', 'target']]
```

Data Exploration

In [3]:
```python
train.describe()
```

Out[3]:

	target	var_0	var_1	var_2	var_3	var_4	
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	2000(
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	'

8 rows × 201 columns

In [4]:
```python
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Columns: 202 entries, ID_code to var_199
dtypes: float64(200), int64(1), object(1)
memory usage: 308.2+ MB
```

In [5]:
```
train.shape
```
Out[5]: (200000, 202)

In [6]:
```
train.head(5)
```
Out[6]:

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	...	var_190	var_
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	...	4.4354	3.
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	...	7.6421	7.
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	...	2.9057	9.
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	...	4.4666	4.
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	...	-1.4905	9.

5 rows × 202 columns

Check for the Missing Values.

In [7]:
```
#Check for Missing Values after Concatination

obs = train.isnull().sum().sort_values(ascending = False)
percent = round(train.isnull().sum().sort_values(ascending = False)/len(train)*100, 2)
pd.concat([obs, percent], axis = 1,keys= ['Number of Observations', 'Percent'])
```
Out[7]:

	Number of Observations	Percent
var_199	0	0.0
var_61	0	0.0
var_71	0	0.0
var_70	0	0.0
var_69	0	0.0
...	...	...
var_129	0	0.0
var_128	0	0.0
var_127	0	0.0
var_126	0	0.0
ID_code	0	0.0

202 rows × 2 columns

There are no missing values in the dataset

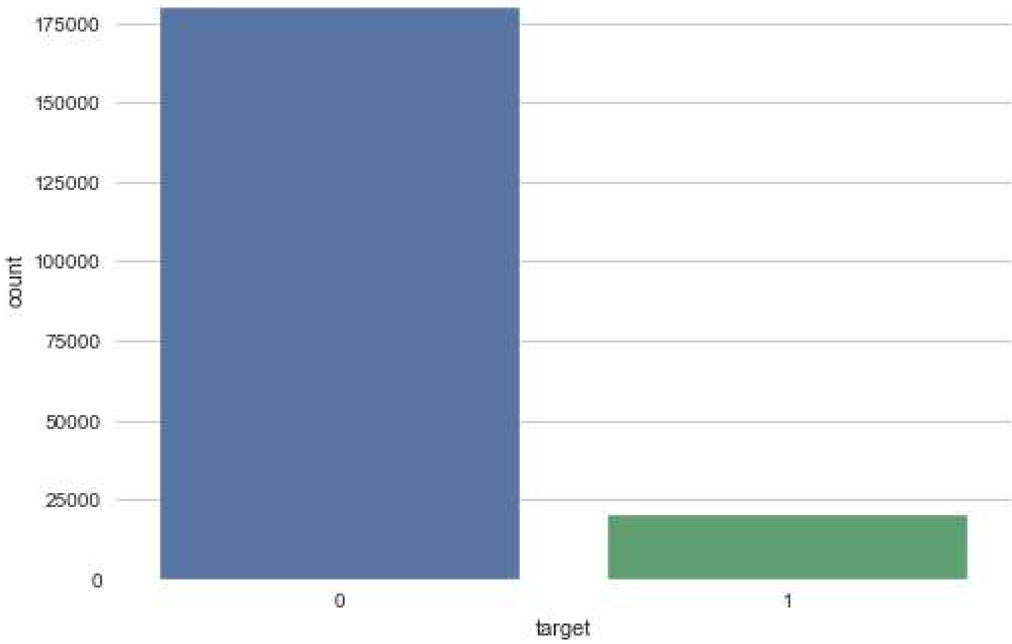
Visualizing the Satendar Customer Transactions Data

Check for Class Imbalance

In [8]:

```
target = train['target']
train = train.drop(["ID_code", "target"], axis=1)
sns.set_style('whitegrid')
sns.countplot(target)
```

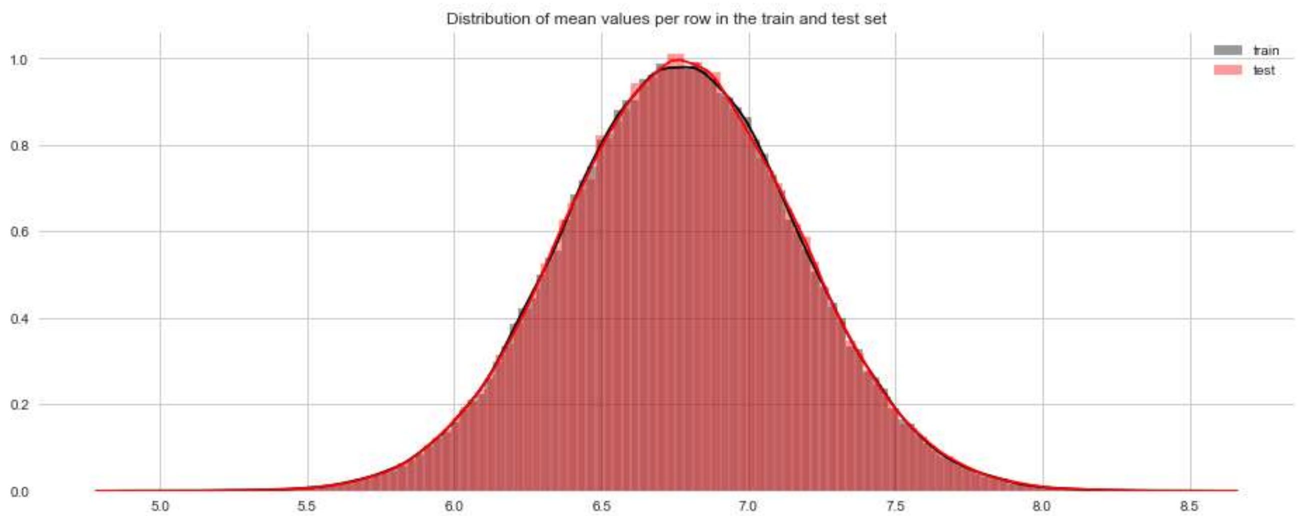
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1bd96ab8760>




```
<pre><a id = 32 ><b>Distribution of Mean and Standard Deviation</b></a></pre>
```

In [9]:

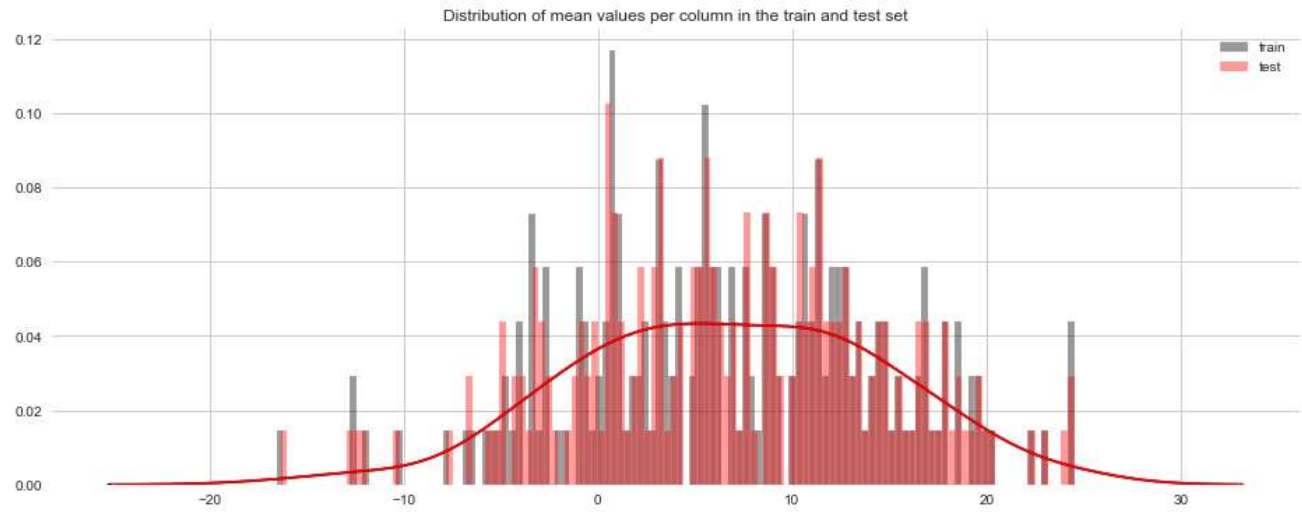
```
plt.figure(figsize=(16,6))
plt.title("Distribution of mean values per row in the train and test set")
sns.distplot(train[features].mean(axis=1),color="black", kde=True,bins=120, label='train')
sns.distplot(test[features].mean(axis=1),color="red", kde=True,bins=120, label='test')
plt.legend()
plt.show()
```



Let's check the distribution of the mean of values per columns in the train and test datasets.

In [10]:

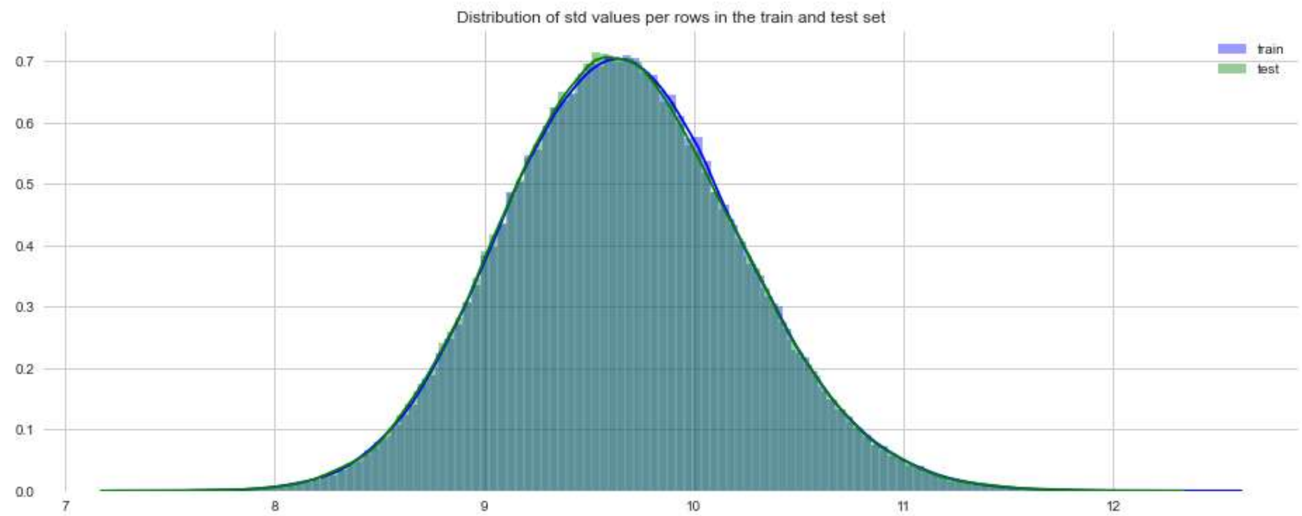
```
plt.figure(figsize=(16,6))
plt.title("Distribution of mean values per column in the train and test set")
sns.distplot(train[features].mean(axis=0),color="black", kde=True,bins=120, label='train')
sns.distplot(test[features].mean(axis=0),color="red", kde=True,bins=120, label='test')
plt.legend();plt.show()
```



Distribution for Standard Deviation

In [11]:

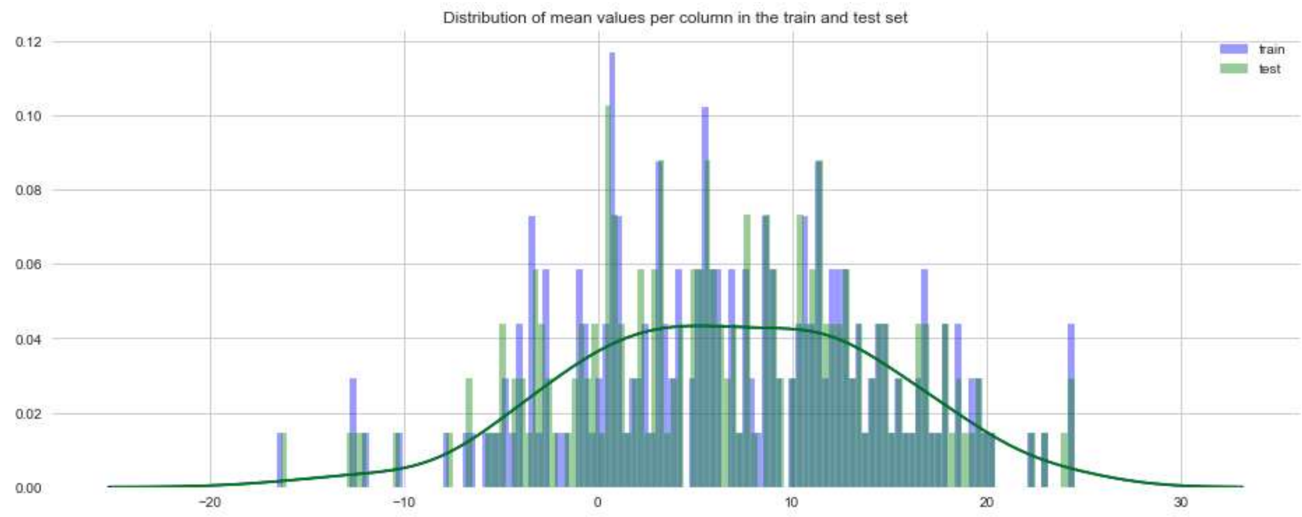
```
plt.figure(figsize=(16,6))
plt.title("Distribution of std values per rows in the train and test set")
sns.distplot(train[features].std(axis=1),color="blue",kde=True,bins=120, label='train')
sns.distplot(test[features].std(axis=1),color="green", kde=True,bins=120, label='test')
plt.legend(); plt.show()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



Let's check the distribution of the standard deviation of values per columns in the train and test datasets.

In [12]:

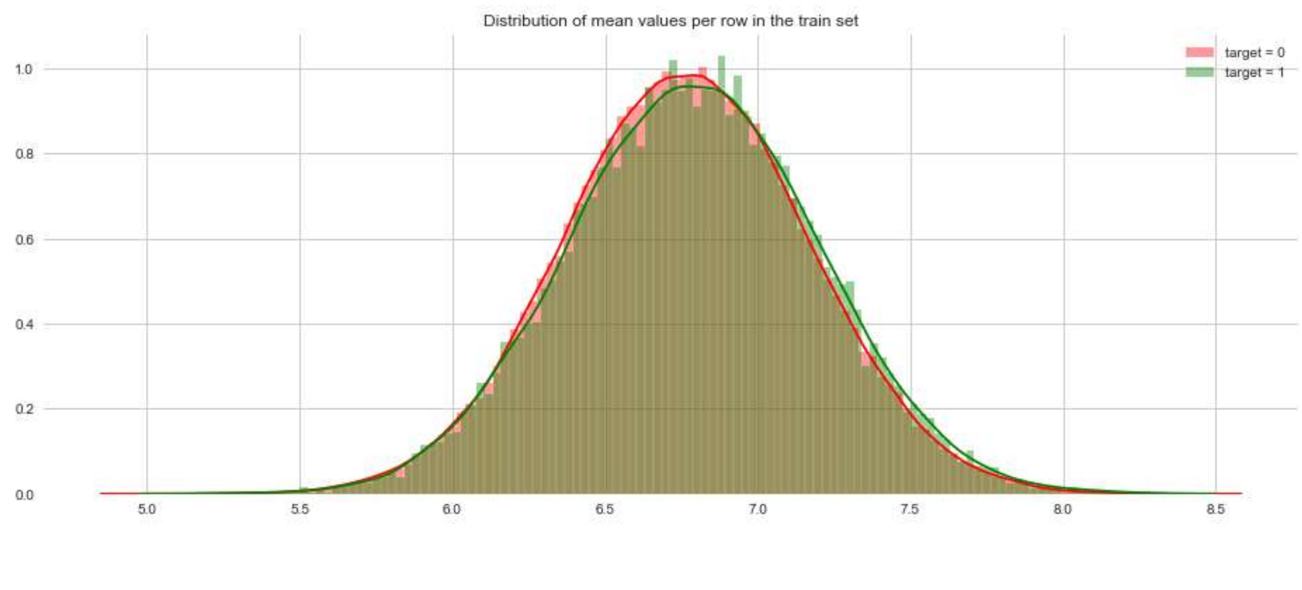
```
plt.figure(figsize=(16,6))
plt.title("Distribution of mean values per column in the train and test set")
sns.distplot(train[features].mean(axis=0),color="blue", kde=True,bins=120, label='train')
sns.distplot(test[features].mean(axis=0),color="green", kde=True,bins=120, label='test')
plt.legend();plt.show()
```



Let's check now the distribution of the mean value per row in the train dataset, grouped by value of target

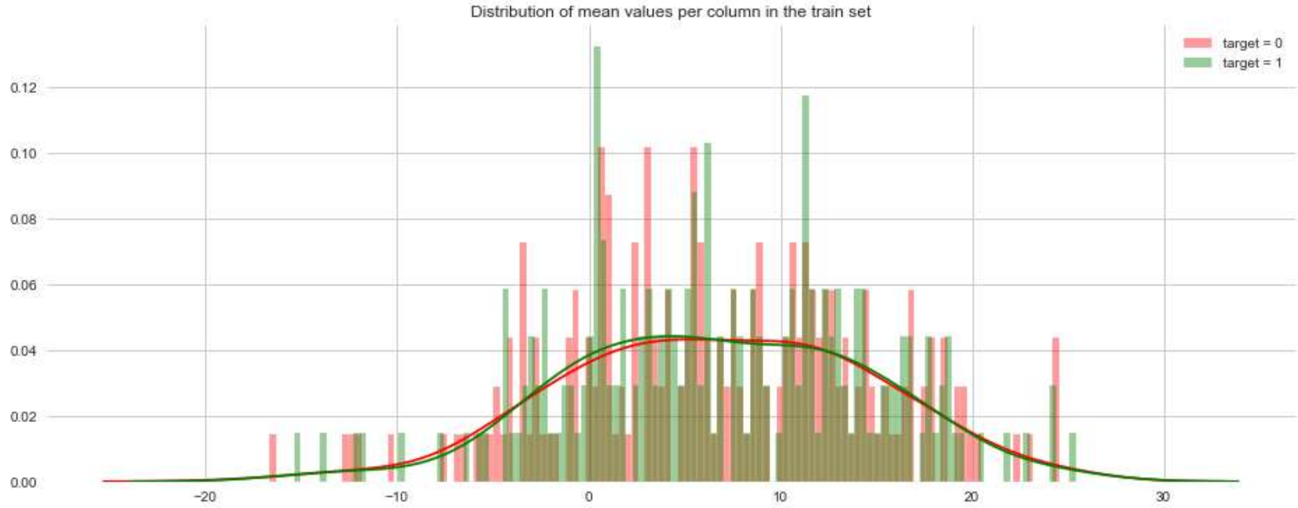
In [13]:

```
t0 = train.loc[target == 0]
t1 = train.loc[target == 1]
plt.figure(figsize=(16,6))
plt.title("Distribution of mean values per row in the train set")
sns.distplot(t0[features].mean(axis=1),color="red", kde=True,bins=120, label='target = 0')
sns.distplot(t1[features].mean(axis=1),color="green", kde=True,bins=120, label='target = 1')
plt.legend(); plt.show()
```



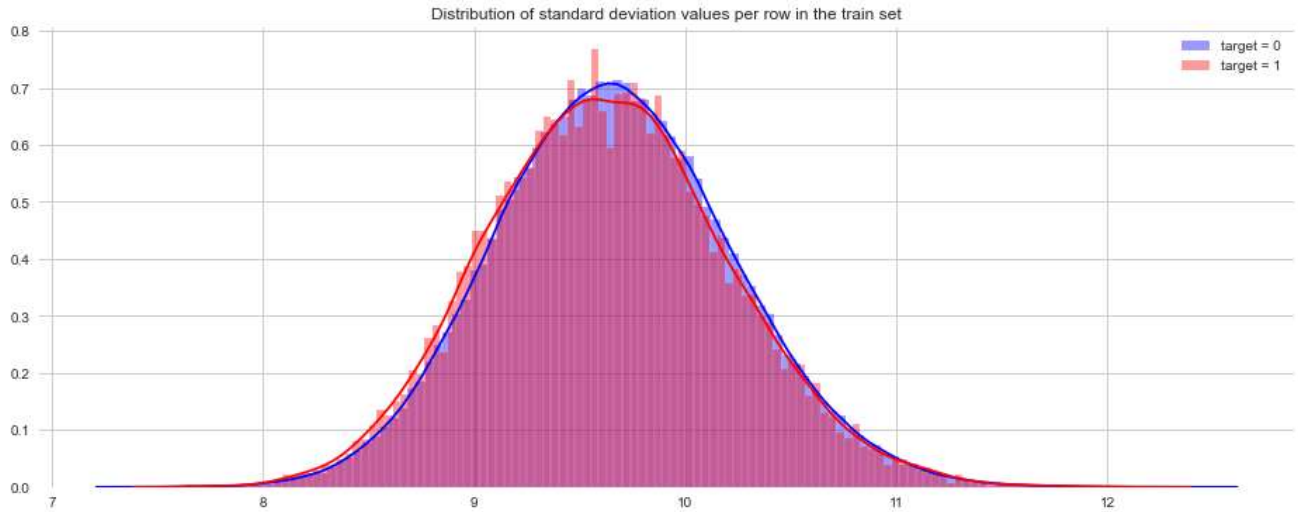
Let's check now the distribution of the mean values per columns in the train and test da tasets.

In [14]:
```
t0 = train.loc[target == 0]
t1 = train.loc[target == 1]
plt.figure(figsize=(16,6))
plt.title("Distribution of mean values per column in the train set")
sns.distplot(t0[features].mean(axis=0),color="red", kde=True,bins=120, label='target = 0')
sns.distplot(t1[features].mean(axis=0),color="green", kde=True,bins=120, label='target = 1')
plt.legend(); plt.show()
```



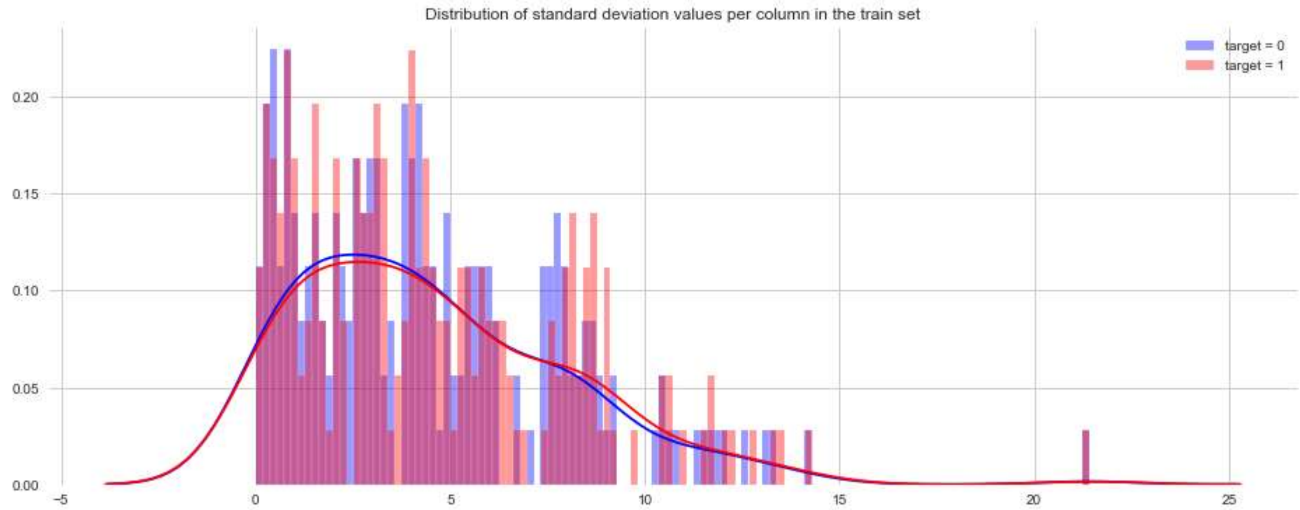
Let's check now the distribution of the standard deviation per row in the train datase t, grouped by value of target

In [15]:
```
t0 = train.loc[target == 0]
t1 = train.loc[target == 1]
plt.figure(figsize=(16,6))
plt.title("Distribution of standard deviation values per row in the train set")
sns.distplot(t0[features].std(axis=1),color="blue", kde=True,bins=120, label='target = 0')
sns.distplot(t1[features].std(axis=1),color="red", kde=True,bins=120, label='target = 1')
plt.legend(); plt.show()
```



Let's check now the distribution of standard deviation per columns in the train and test datasets.

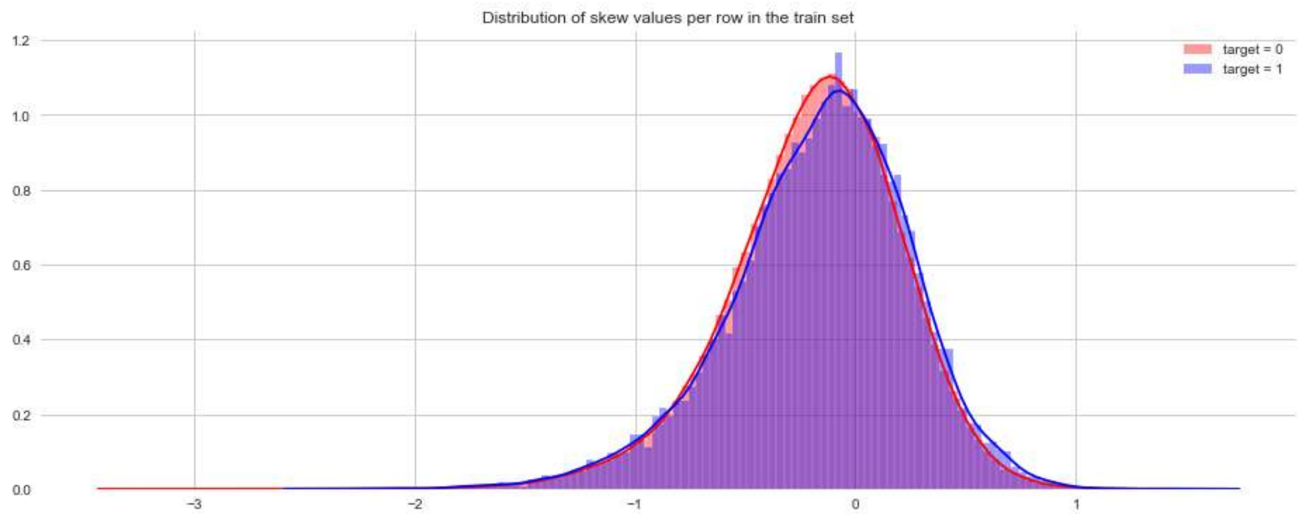
In [16]:
```
t0 = train.loc[target  == 0]
t1 = train.loc[target  == 1]
plt.figure(figsize=(16,6))
plt.title("Distribution of standard deviation values per column in the train set")
sns.distplot(t0[features].std(axis=0),color="blue", kde=True,bins=120, label='target = 0')
sns.distplot(t1[features].std(axis=0),color="red", kde=True,bins=120, label='target = 1')
plt.legend(); plt.show()
```



Distribution of Skewness

Let's see now the distribution of skewness on rows in train separated for values of target 0 and 1. We found the distribution is left skewed

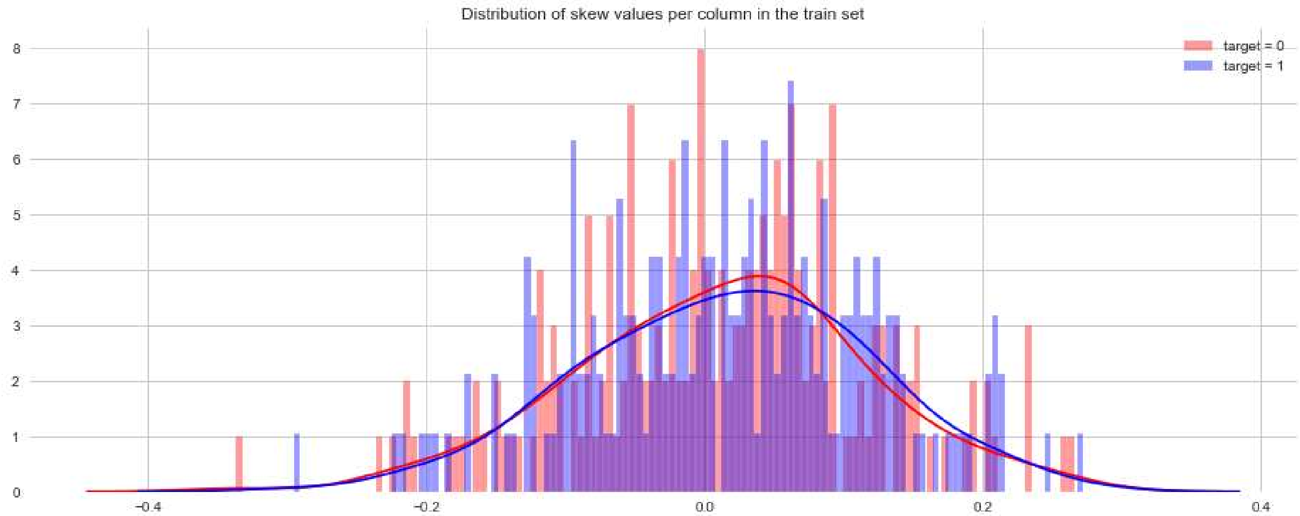
In [17]:
```
t0 = train.loc[target == 0]
t1 = train.loc[target == 1]
plt.figure(figsize=(16,6))
plt.title("Distribution of skew values per row in the train set")
sns.distplot(t0[features].skew(axis=1),color="red", kde=True,bins=120, label='target = 0')
sns.distplot(t1[features].skew(axis=1),color="blue", kde=True,bins=120, label='target = 1')
plt.legend(); plt.show()
```



Let's see now the distribution of skewness on columns in train separated for values of target 0 and 1.

In [18]:

```
t0 = train.loc[target == 0]
t1 = train.loc[target == 1]
plt.figure(figsize=(16,6))
plt.title("Distribution of skew values per column in the train set")
sns.distplot(t0[features].skew(axis=0),color="red", kde=True,bins=120, label='target = 0')
sns.distplot(t1[features].skew(axis=0),color="blue", kde=True,bins=120, label='target = 1')
plt.legend(); plt.show()
```

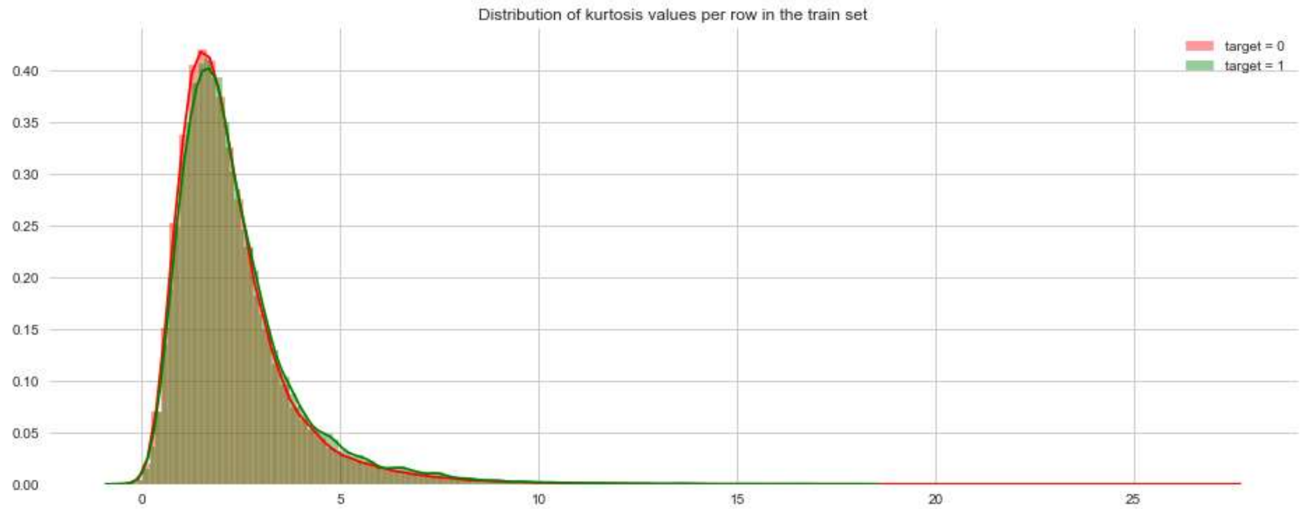


Distribution of Kurtosis

Let's see now the distribution of kurtosis on rows in train separated for values of target 0 and 1. We found the distribution to be Leptokurtic

In [19]:

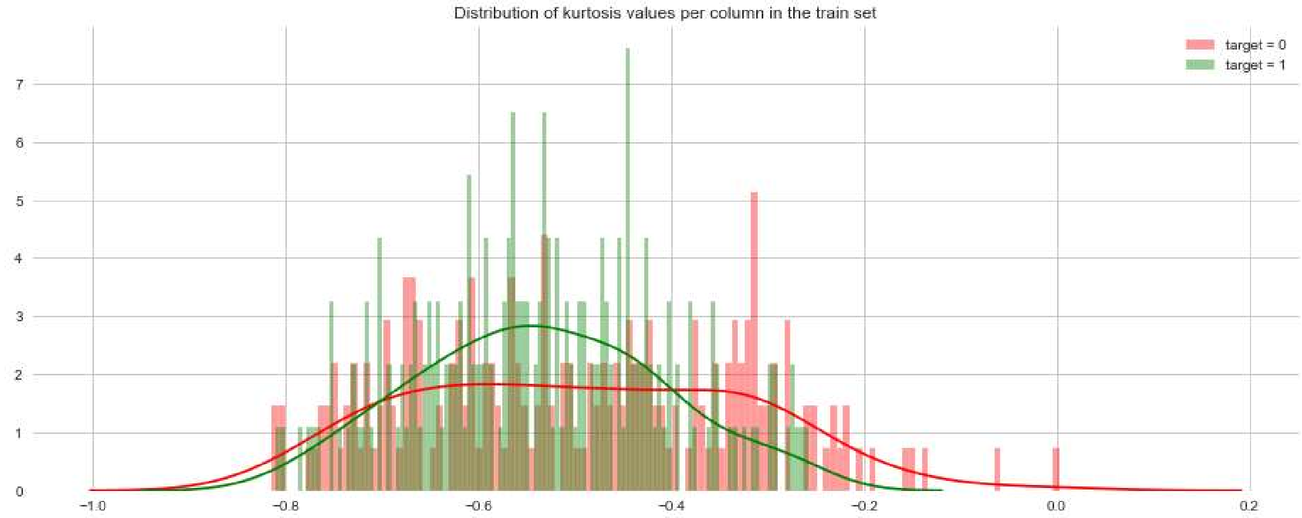
```
t0 = train.loc[target == 0]
t1 = train.loc[target == 1]
plt.figure(figsize=(16,6))
plt.title("Distribution of kurtosis values per row in the train set")
sns.distplot(t0[features].kurtosis(axis=1),color="red", kde=True,bins=120, label='target = 0'
sns.distplot(t1[features].kurtosis(axis=1),color="green", kde=True,bins=120, label='target =
plt.legend(); plt.show()
```



Let's see now the distribution of kurtosis on columns in train separated for values of target 0 and 1.

In [20]:

```
t0 = train.loc[target == 0]
t1 = train.loc[target == 1]
plt.figure(figsize=(16,6))
plt.title("Distribution of kurtosis values per column in the train set")
sns.distplot(t0[features].kurtosis(axis=0),color="red", kde=True,bins=120, label='target = 0'
sns.distplot(t1[features].kurtosis(axis=0),color="green", kde=True,bins=120, label='target =
plt.legend(); plt.show()
```

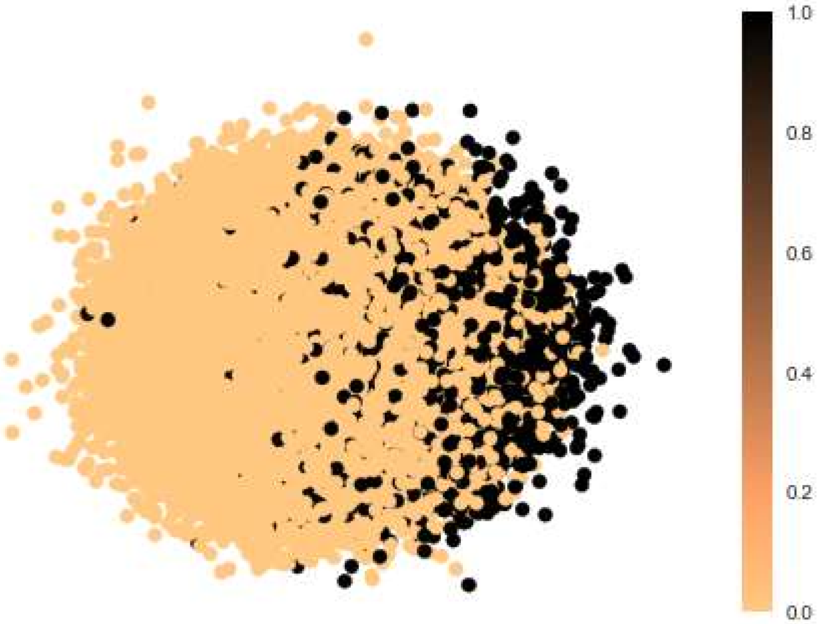


Principal Component Analysis to check Dimentionality Reduction

In [21]:

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
train_scaled = scaler.fit_transform(train)
PCA_train_x = PCA(2).fit_transform(train_scaled)
plt.scatter(PCA_train_x[:, 0], PCA_train_x[:, 1], c=target, cmap="copper_r")
plt.axis('off')
plt.colorbar()
plt.show()
```



Kernel PCA (Since the Graph above doesn't represent meaningful analysis)

In [22]:

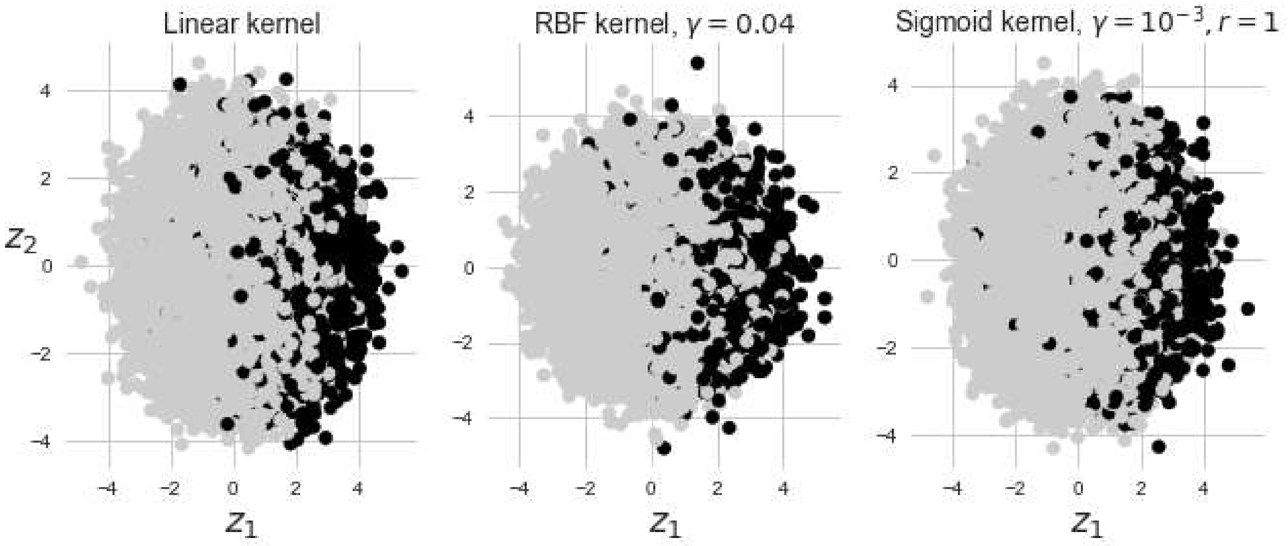
```
from sklearn.decomposition import KernelPCA

lin_pca = KernelPCA(n_components = 2, kernel="linear", fit_inverse_transform=True)
rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.0433, fit_inverse_transform=True)
sig_pca = KernelPCA(n_components = 2, kernel="sigmoid", gamma=0.001, coef0=1, fit_inverse_tra


plt.figure(figsize=(11, 4))
for subplot, pca, title in ((131, lin_pca, "Linear kernel"), (132, rbf_pca, "RBF kernel, $\ga
                            (133, sig_pca, "Sigmoid kernel, $\gamma=10^{-3}, r=1$")):

    PCA_train_x = PCA(2).fit_transform(train_scaled)
    plt.subplot(subplot)
    plt.title(title, fontsize=14)
    plt.scatter(PCA_train_x[:, 0], PCA_train_x[:, 1], c=target, cmap="nipy_spectral_r")
    plt.xlabel("$z_1$", fontsize=18)
    if subplot == 131:
        plt.ylabel("$z_2$", fontsize=18, rotation=0)
    plt.grid(True)

plt.show()
```



Since PCA hasn't been useful, I decided to proceed with the existing dataset

Data Augmentation

In [23]:

```python
def augment(x,y,t=2):
    xs,xn = [],[]
    for i in range(t):
        mask = y>0
        x1 = x[mask].copy()
        ids = np.arange(x1.shape[0])
        for c in range(x1.shape[1]):
            np.random.shuffle(ids)
            x1[:,c] = x1[ids][:,c]
        xs.append(x1)

    for i in range(t//2):
        mask = y==0
        x1 = x[mask].copy()
        ids = np.arange(x1.shape[0])
        for c in range(x1.shape[1]):
            np.random.shuffle(ids)
            x1[:,c] = x1[ids][:,c]
        xn.append(x1)

    xs = np.vstack(xs)
    xn = np.vstack(xn)
    ys = np.ones(xs.shape[0])
    yn = np.zeros(xn.shape[0])
    x = np.vstack([x,xs,xn])
    y = np.concatenate([y,ys,yn])
    return x,y
```

Build the Light GBM Model

In [24]:

```python
param = {
    'bagging_freq': 5,
    'bagging_fraction': 0.335,
    'boost_from_average':'false',
    'boost': 'gbdt',
    'feature_fraction': 0.041,
    'learning_rate': 0.0083,
    '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 [25]:

```python
train.shape
```

Out[25]:

```
(200000, 200)
```

In [28]:

```
num_folds = 11
features = [c for c in train.columns if c not in ['ID_code', 'target']]

folds = KFold(n_splits=num_folds)
oof = np.zeros(len(train))
getVal = np.zeros(len(train))
predictions = np.zeros(len(target))
feature_importance_df = pd.DataFrame()

print('Light GBM Model')
for fold_, (trn_idx, val_idx) in enumerate(folds.split(train.values, target.values)):

    X_train, y_train = train.iloc[trn_idx][features], target.iloc[trn_idx]
    X_valid, y_valid = train.iloc[val_idx][features], target.iloc[val_idx]

    X_tr, y_tr = augment(X_train.values, y_train.values)
    X_tr = pd.DataFrame(X_tr)

    print("Fold idx:{}".format(fold_ + 1))
    trn_data = lgb.Dataset(X_tr, label=y_tr)
    val_data = lgb.Dataset(X_valid, label=y_valid)

    clf = lgb.train(param, trn_data, 1000000, valid_sets = [trn_data, val_data], verbose_eval=
    oof[val_idx] = clf.predict(train.iloc[val_idx][features], num_iteration=clf.best_iteration
    getVal[val_idx]+= clf.predict(train.iloc[val_idx][features], num_iteration=clf.best_itera

    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_sp

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

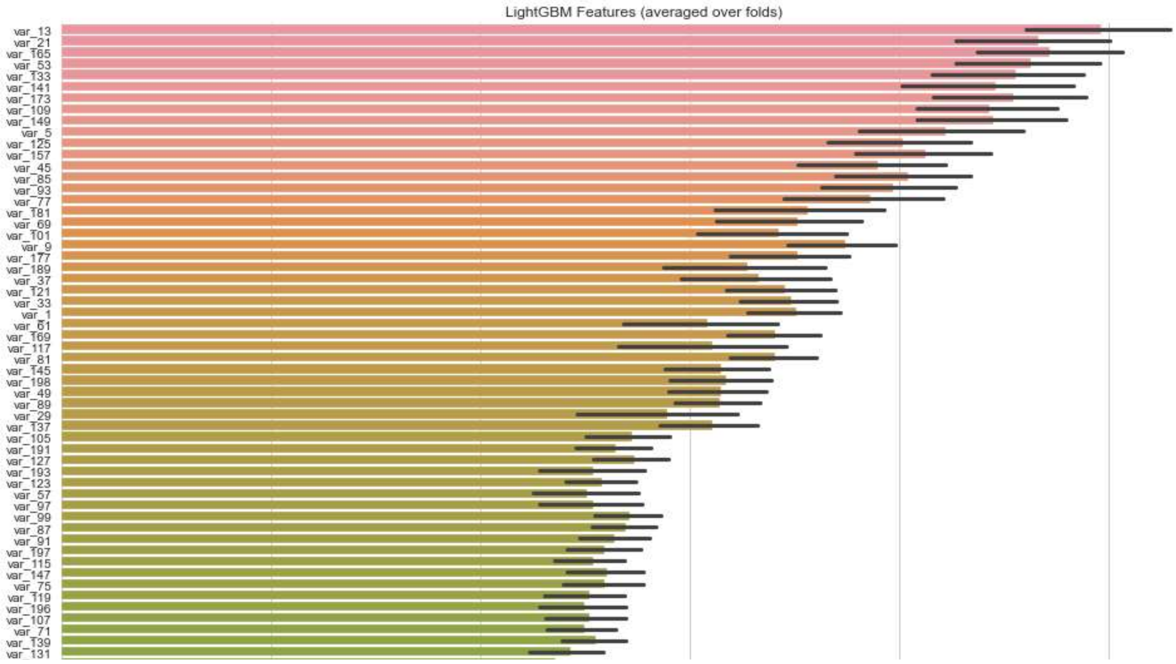
```
Light GBM Model
Fold idx:1
Training until validation scores don't improve for 4000 rounds
[5000]  training's auc: 0.90899 valid_1's auc: 0.8934
[10000] training's auc: 0.921198        valid_1's auc: 0.899973
[15000] training's auc: 0.929112        valid_1's auc: 0.901135
[20000] training's auc: 0.936042        valid_1's auc: 0.900976
Early stopping, best iteration is:
[16679] training's auc: 0.931525        valid_1's auc: 0.901338
Fold idx:2
Training until validation scores don't improve for 4000 rounds
[5000]  training's auc: 0.90959 valid_1's auc: 0.891068
[10000] training's auc: 0.921765        valid_1's auc: 0.897038
[15000] training's auc: 0.929677        valid_1's auc: 0.898043
[20000] training's auc: 0.936498        valid_1's auc: 0.897726
Early stopping, best iteration is:
[16058] training's auc: 0.931178        valid_1's auc: 0.898115
Fold idx:3
Training until validation scores don't improve for 4000 rounds
[5000]  training's auc: 0.910229        valid_1's auc: 0.885835
[10000] training's auc: 0.922399        valid_1's auc: 0.89189
[15000] training's auc: 0.930175        valid_1's auc: 0.892519
Early stopping, best iteration is:
[15246] training's auc: 0.930528        valid_1's auc: 0.892589
Fold idx:4
Training until validation scores don't improve for 4000 rounds
[5000]  training's auc: 0.909218        valid_1's auc: 0.895167
[10000] training's auc: 0.921484        valid_1's auc: 0.901369
```

```
[15000] training's auc: 0.929364	valid_1's auc: 0.902368
[20000] training's auc: 0.936328	valid_1's auc: 0.902688
Early stopping, best iteration is:
[19399] training's auc: 0.935526	valid_1's auc: 0.9028
Fold idx:5
Training until validation scores don't improve for 4000 rounds
[5000]	training's auc: 0.910046	valid_1's auc: 0.894027
[10000] training's auc: 0.922115	valid_1's auc: 0.898665
[15000] training's auc: 0.929861	valid_1's auc: 0.898896
Early stopping, best iteration is:
[11901] training's auc: 0.925245	valid_1's auc: 0.899008
Fold idx:6
Training until validation scores don't improve for 4000 rounds
[5000]	training's auc: 0.90949 valid_1's auc: 0.896594
[10000] training's auc: 0.921717	valid_1's auc: 0.90164
[15000] training's auc: 0.929695	valid_1's auc: 0.902496
Early stopping, best iteration is:
[15472] training's auc: 0.930381	valid_1's auc: 0.902526
Fold idx:7
Training until validation scores don't improve for 4000 rounds
[5000]	training's auc: 0.909546	valid_1's auc: 0.898875
[10000] training's auc: 0.921673	valid_1's auc: 0.904299
[15000] training's auc: 0.929507	valid_1's auc: 0.905297
[20000] training's auc: 0.936381	valid_1's auc: 0.905621
Early stopping, best iteration is:
[19076] training's auc: 0.935171	valid_1's auc: 0.905691
Fold idx:8
Training until validation scores don't improve for 4000 rounds
[5000]	training's auc: 0.90989 valid_1's auc: 0.893758
[10000] training's auc: 0.922017	valid_1's auc: 0.898745
[15000] training's auc: 0.929833	valid_1's auc: 0.899375
[20000] training's auc: 0.936699	valid_1's auc: 0.899487
Early stopping, best iteration is:
[19503] training's auc: 0.93603 valid_1's auc: 0.899593
Fold idx:9
Training until validation scores don't improve for 4000 rounds
[5000]	training's auc: 0.909496	valid_1's auc: 0.896646
[10000] training's auc: 0.921713	valid_1's auc: 0.902544
[15000] training's auc: 0.929546	valid_1's auc: 0.903464
[20000] training's auc: 0.936423	valid_1's auc: 0.903514
Early stopping, best iteration is:
[18610] training's auc: 0.934564	valid_1's auc: 0.903672
Fold idx:10
Training until validation scores don't improve for 4000 rounds
[5000]	training's auc: 0.909188	valid_1's auc: 0.89784
[10000] training's auc: 0.921441	valid_1's auc: 0.903906
[15000] training's auc: 0.929344	valid_1's auc: 0.905076
[20000] training's auc: 0.936173	valid_1's auc: 0.905287
Early stopping, best iteration is:
[19840] training's auc: 0.935965	valid_1's auc: 0.90536
Fold idx:11
Training until validation scores don't improve for 4000 rounds
[5000]	training's auc: 0.909737	valid_1's auc: 0.894735
[10000] training's auc: 0.92194 valid_1's auc: 0.899571
[15000] training's auc: 0.929759	valid_1's auc: 0.900475
[20000] training's auc: 0.936589	valid_1's auc: 0.900646
Early stopping, best iteration is:
[18815] training's auc: 0.935016	valid_1's auc: 0.900737
CV score: 0.90098
```

In [29]:

```python
cols = (feature_importance_df[["feature", "importance"]]
        .groupby("feature")
        .mean()
        .sort_values(by="importance", ascending=False)[:1000].index)
best_features = feature_importance_df.loc[feature_importance_df.feature.isin(cols)]

plt.figure(figsize=(14,26))
sns.barplot(x="importance", y="feature", data=best_features.sort_values(by="importance",ascen
plt.title('LightGBM Features (averaged over folds)')
plt.tight_layout()
plt.savefig('lgbm_importances.png')
```



In [30]:

```python
num_sub = 26
print('Saving the Submission File')
sub = pd.DataFrame({"ID_code": test.ID_code.values})
sub["target"] = predictions
sub.to_csv('submission{}.csv'.format(num_sub), index=False)
getValue = pd.DataFrame(getVal)
getValue.to_csv("Validation_kfold.csv")
```

```
Saving the Submission File
```

In [ ]:

```python
```