In [4]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split,cross_val_predict,cross_val_score
from sklearn.metrics import
roc_auc_score,confusion_matrix,make_scorer,classification_report,roc_curve,auc
from sklearn.model_selection import StratifiedKFold
from imblearn.over_sampling import SMOTE, RandomOverSampler
from imblearn.under_sampling import ClusterCentroids,NearMiss, RandomUnderSampler
import lightgbm as lgb
import eli5
from eli5.sklearn import PermutationImportance
from sklearn import tree
import graphviz
from pdpbox import pdp, get_dataset, info_plots
import scikitplot as skplt
from scikitplot.metrics import plot_confusion_matrix,plot_precision_recall_curve


from scipy.stats import randint as sp_randint
import warnings
warnings.filterwarnings('ignore')

import os
#print(os.listdir("C:/Users/apurv/Desktop/Sadaf/PROJECT 1"))
#print(os.listdir("C:/Users/Sadaf.Mehdi/Desktop/Project 1"))

random_state=42
np.random.seed(random_state)
```

In [5]:

```
#importing the train dataset
train_df=pd.read_csv('train.csv')
train_df.head()
```

Out[5]:

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	...	var_190	var_191	var_192	var_193	var_19
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	...	4.4354	3.9642	3.1364	1.6910	18.522
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	...	7.6421	7.7214	2.5837	10.9516	15.430
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	...	2.9057	9.7905	1.6704	1.6858	21.604
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	...	4.4666	4.7433	0.7178	1.4214	23.034
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	...	-1.4905	9.5214	-0.1508	9.1942	13.287

5 rows × 202 columns

In [6]:

```
#Shape of the train dataset
train_df.shape
```

Out[6]:

```
(200000, 202)
```

In [7]:

```
#Summary of the dataset
train_df.describe()
```

Out[7]:

	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200(
mean	0.100490	10.679914	-1.627622	10.715192	6.796529	11.078333	-5.065317	5.408949	
std	0.300653	3.040051	4.050044	2.640894	2.043319	1.623150	7.863267	0.866607	
min	0.000000	0.408400	-15.043400	2.117100	-0.040200	5.074800	-32.562600	2.347300	
25%	0.000000	8.453850	-4.740025	8.722475	5.254075	9.883175	-11.200350	4.767700	
50%	0.000000	10.524750	-1.608050	10.580000	6.825000	11.108250	-4.833150	5.385100	
75%	0.000000	12.758200	1.358625	12.516700	8.324100	12.261125	0.924800	6.003000	
max	1.000000	20.315000	10.376800	19.353000	13.188300	16.671400	17.251600	8.447700	

8 rows × 201 columns

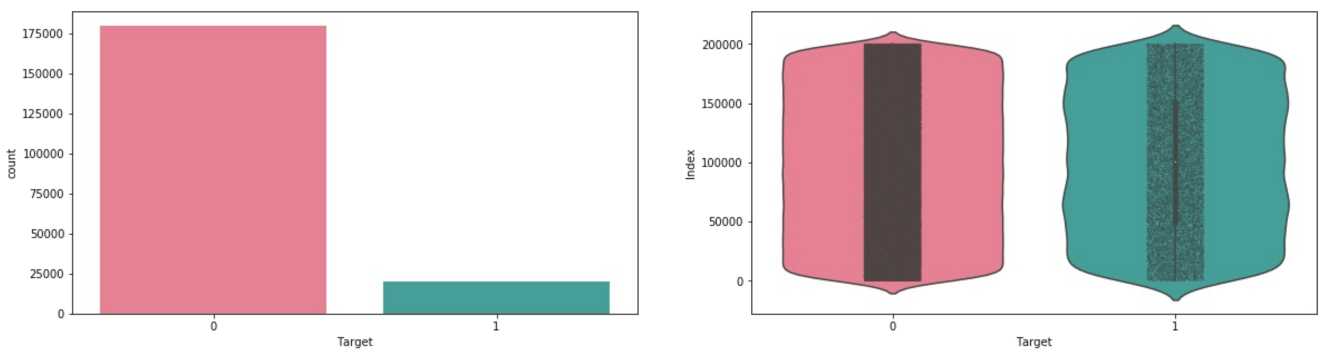
In [8]:

```
%%time
#target classes count
target_class=train_df['target'].value_counts()
print('Count of target classes :\n',target_class)
#Percentage of target classes count
per_target_class=train_df['target'].value_counts()/len(train_df)*100
print('percentage of count of target classes :\n',per_target_class)

#Countplot and violin plot for target classes
fig,ax=plt.subplots(1,2,figsize=(20,5))
sns.countplot(train_df.target.values,ax=ax[0],palette='husl')
sns.violinplot(x=train_df.target.values,y=train_df.index.values,ax=ax[1],palette='husl')
sns.stripplot(x=train_df.target.values,y=train_df.index.values,jitter=True,color='black',linewidth=
0.5,size=0.5,alpha=0.5,ax=ax[1],palette='husl')
ax[0].set_xlabel('Target')
ax[1].set_xlabel('Target')
ax[1].set_ylabel('Index')
```

```
Count of target classes :
 0    179902
1     20098
Name: target, dtype: int64
percentage of count of target classes :
 0    89.951
1    10.049
Name: target, dtype: float64
Wall time: 1.84 s
```

Out[8]:

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



In [10]:

```
print("""
We have a unbalanced data,where 90% of the data is the number of customers those will not make a t
ransaction and 10% of the data is those who will make a transaction.

""")
```

```
We have a unbalanced data,where 90% of the data is the number of customers those will not make a t
ransaction and 10% of the data is those who will make a transaction.
```

In [11]:

```
print("Let us look distribution of train attributes")
```

```
Let us look distribution of train attributes
```

In [12]:

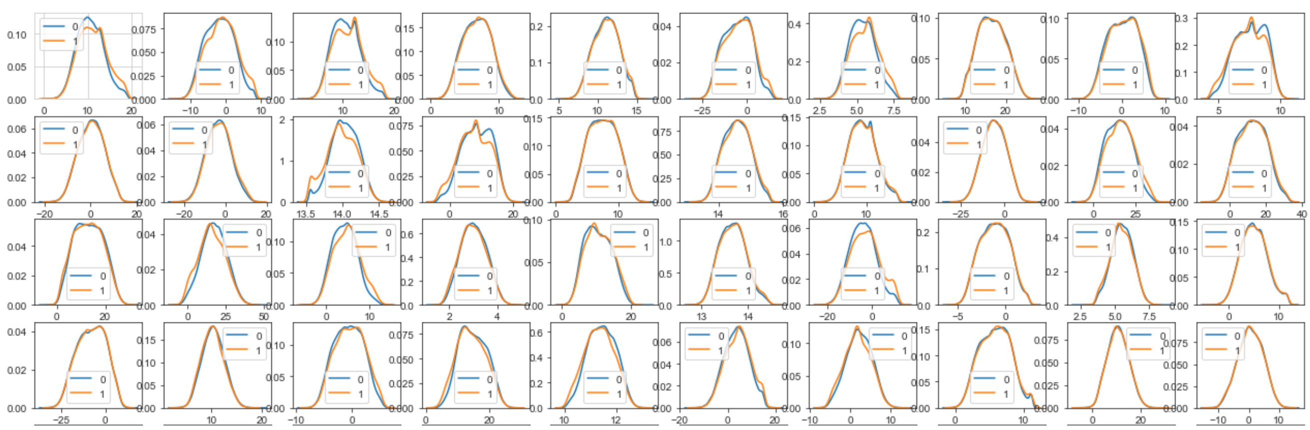
```
%%time
#Distribution of train attributes
def plot_train_attribute_distribution(t0,t1,label1,label2,train_attributes):
    i=0
    sns.set_style('whitegrid')

    fig=plt.figure()
    ax=plt.subplots(10,10,figsize=(22,18))

    for attribute in train_attributes:
        i+=1
        plt.subplot(10,10,i)
        sns.distplot(t0[attribute],hist=False,label=label1)
        sns.distplot(t1[attribute],hist=False,label=label2)
        plt.legend()
        plt.xlabel('Attribute',)
        sns.set_style("ticks", {"xtick.major.size": 8, "ytick.major.size": 8})
    plt.show()
```

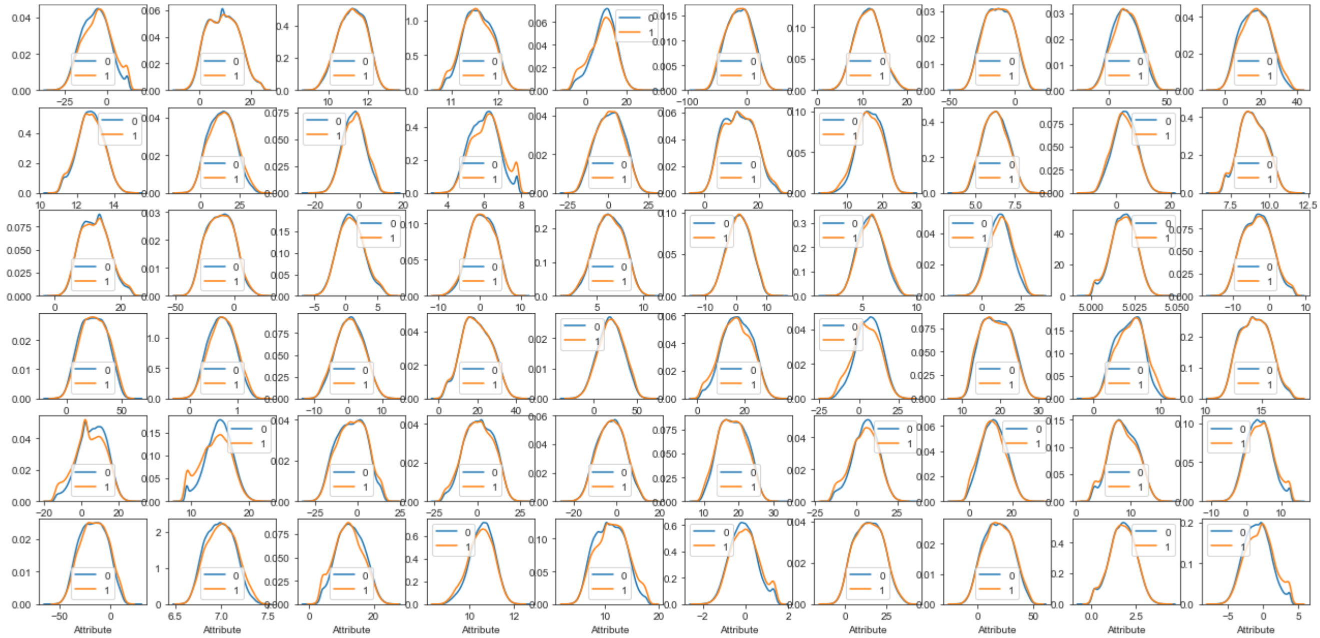
```
Wall time: 0 ns
```

In [13]:

```
%%time
#corresponding to negative class
t0=train_df[train_df.target.values==0]
#corresponding to positive class
t1=train_df[train_df.target.values==1]
#train attributes from 2 to 102
train_attributes=train_df.columns.values[2:102]
#plot distribution of train attributes
plot_train_attribute_distribution(t0,t1,'0','1',train_attributes)
```

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





```
Wall time: 18.7 s
```

In [14]:

```
%%time
#train attributes from 102 to 203
train_attributes=train_df.columns.values[102:203]
#plot distribution of train attributes
plot_train_attribute_distribution(t0,t1,'0','1',train_attributes)
```

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

```
Wall time: 17.4 s
```

In [16]:

```python
#importing the test dataset
test_df=pd.read_csv('test.csv')
test_df.head()
```

Out[16]:

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	...	var_190	var_191	var_192	var_193	var_1
0	test_0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	2.1337	...	-2.1556	11.8495	-1.4300	2.4508	13.7
1	test_1	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	-4.4131	...	10.6165	8.8349	0.9403	10.1282	15.5
2	test_2	5.4827	-10.3581	10.1407	7.0479	10.2628	9.8052	4.8950	20.2537	1.5233	...	-0.7484	10.9935	1.9803	2.1800	12.9
3	test_3	8.5374	-1.3222	12.0220	6.5749	8.8458	3.1744	4.9397	20.5660	3.3755	...	9.5702	9.0766	1.6580	3.5813	15.1
4	test_4	11.7058	-0.1327	14.1295	7.7506	9.1035	-8.5848	6.8595	10.6048	2.9890	...	4.2259	9.1723	1.2835	3.3778	19.5

5 rows × 201 columns

In [17]:

```python
#Shape of the test dataset
test_df.shape
```

Out[17]:

```
(200000, 201)
```

In [18]:

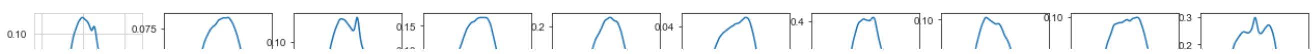
```python
%%time
#Distribution of test attributes
def plot_test_attribute_distribution(test_attributes):
    i=0
    sns.set_style('whitegrid')

    fig=plt.figure()
    ax=plt.subplots(10,10,figsize=(22,18))

    for attribute in test_attributes:
        i+=1
        plt.subplot(10,10,i)
        sns.distplot(test_df[attribute],hist=False)
        plt.xlabel('Attribute',)
        sns.set_style("ticks", {"xtick.major.size": 8, "ytick.major.size": 8})
    plt.show()
```

```
Wall time: 0 ns
```

In [19]:

```python
%%time
#test attribiutes from 1 to 101
test_attributes=test_df.columns.values[1:101]
#plot distribution of test attributes
plot_test_attribute_distribution(test_attributes)
```

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

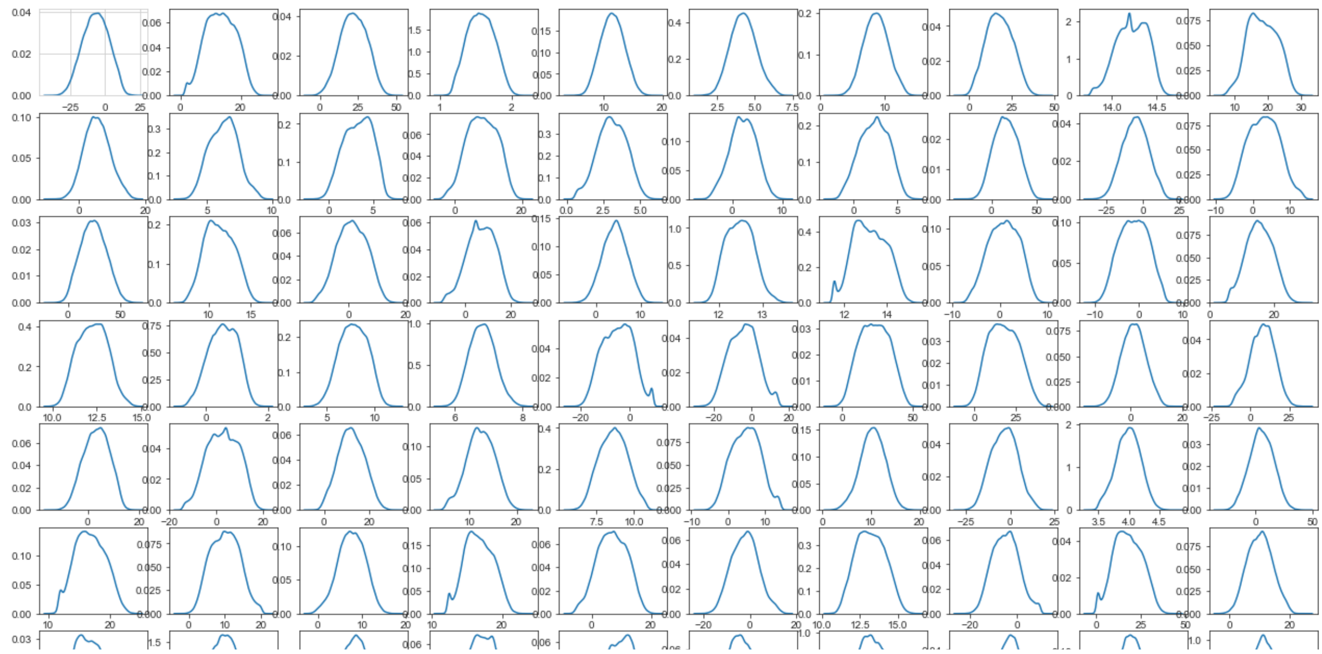


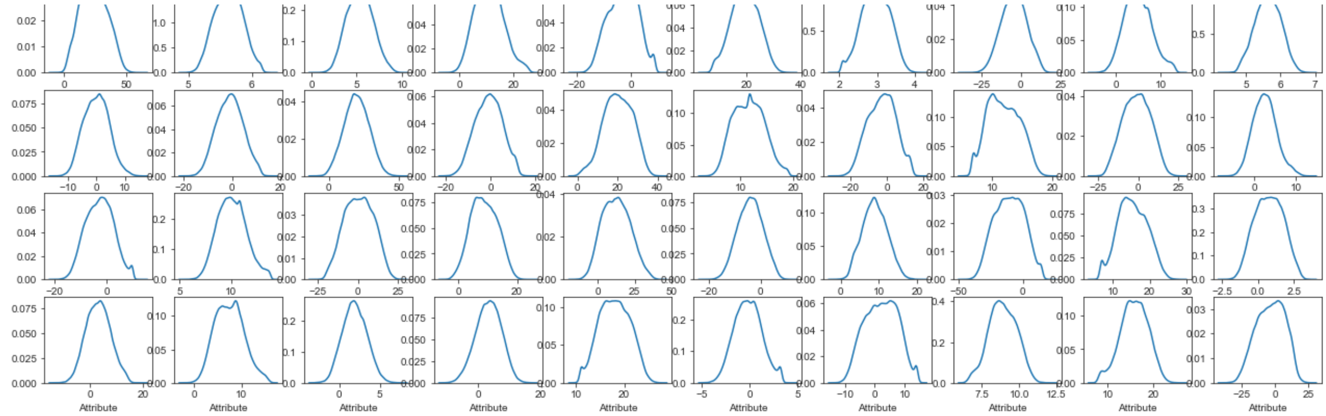
Wall time: 16.2 s

In [20]:

```
%%time
#test attributes from 101 to 202
test_attributes=test_df.columns.values[101:202]
#plot the distribution of test attributes
plot_test_attribute_distribution(test_attributes)
```

<Figure size 432x288 with 0 Axes>





Wall time: 16.4 s

In [22]:

```
print("""
In both test and train data sets, there are considerable number of features that significantly hav
e different distribution
and at the same time considerable number of features have similar distribution
""")
```

In both test and train data sets, there are considerable number of features that significantly hav
e different distribution
and at the same time considerable number of features have similar distribution

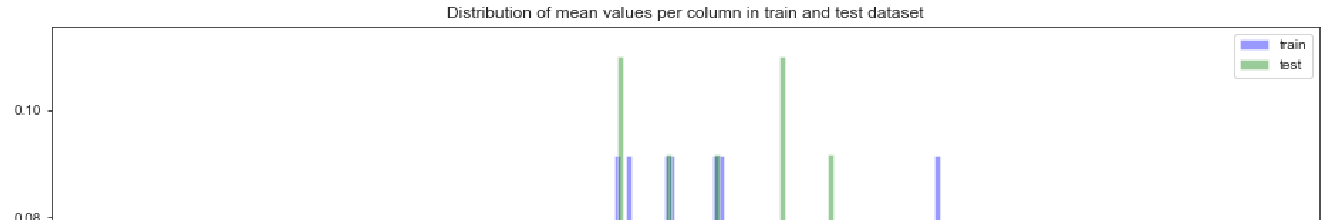
In [23]:

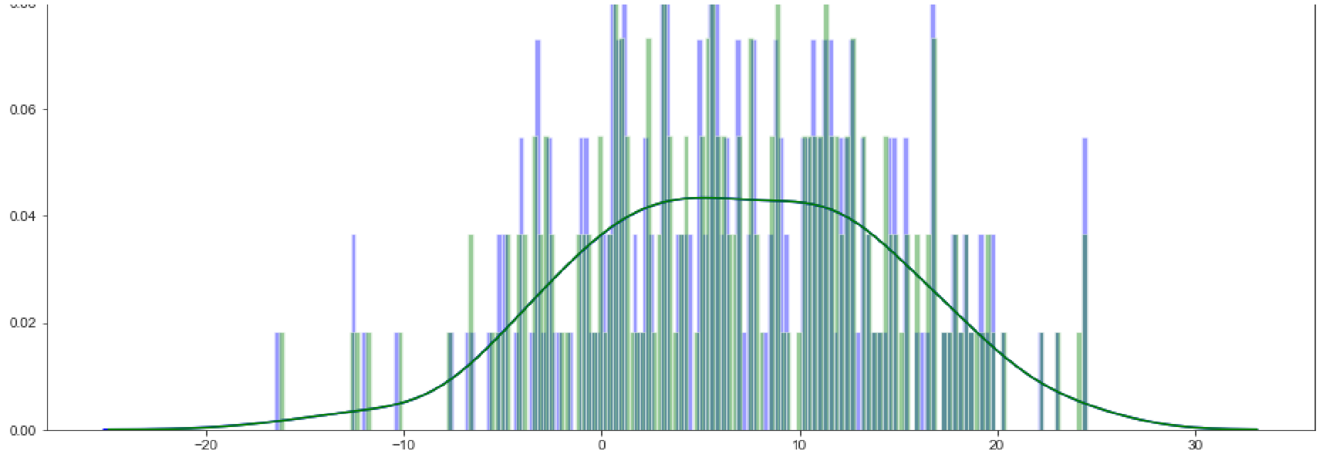
```
print("Let us look distribution of mean values per rows and columns in train and test dataset")
```

Let us look distribution of mean values per rows and columns in train and test dataset

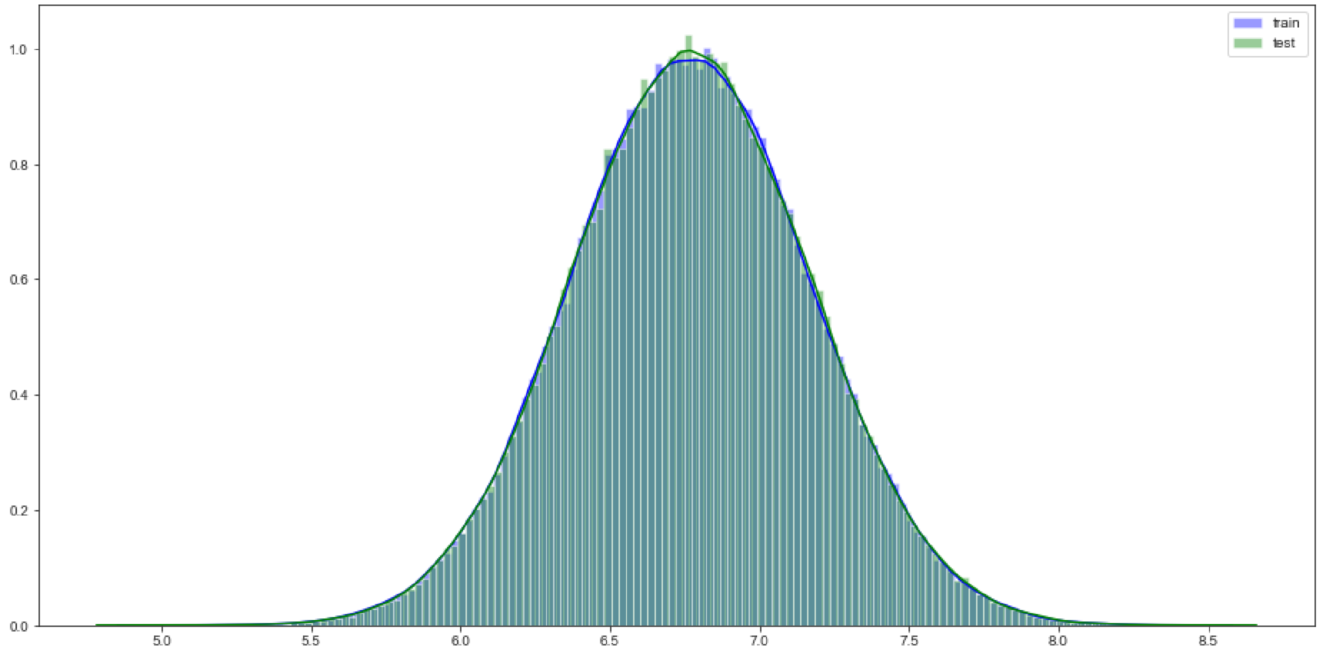
In [24]:

```
#Distribution of mean values per column in train and test dataset
plt.figure(figsize=(16,8))
#train attributes
train_attributes=train_df.columns.values[2:202]
#test attributes
test_attributes=test_df.columns.values[1:201]
#Distribution plot for mean values per column in train attributes
sns.distplot(train_df[train_attributes].mean(axis=0),color='blue',kde=True,bins=150,label='train')
#Distribution plot for mean values per column in test attributes
sns.distplot(test_df[test_attributes].mean(axis=0),color='green',kde=True,bins=150,label='test')
plt.title('Distribution of mean values per column in train and test dataset')
plt.legend()
plt.show()

#Distribution of mean values per row in train and test dataset
plt.figure(figsize=(16,8))
#Distribution plot for mean values per row in train attributes
sns.distplot(train_df[train_attributes].mean(axis=1),color='blue',kde=True,bins=150,label='train')
#Distribution plot for mean values per row in test attributes
sns.distplot(test_df[test_attributes].mean(axis=1),color='green',kde=True, bins=150, label='test')
plt.title('Distribution of mean values per row in train and test dataset')
plt.legend()
plt.show()
```





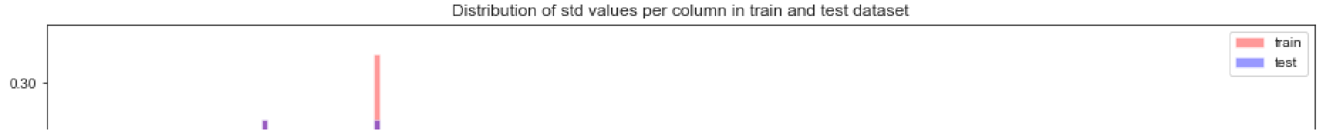
Distribution of mean values per row in train and test dataset

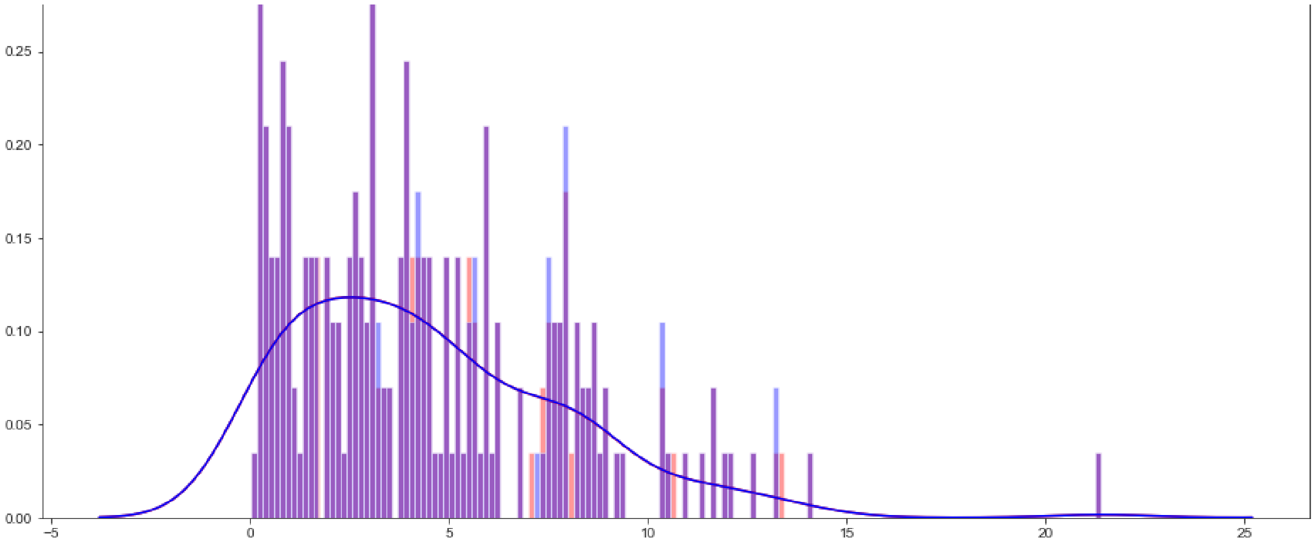


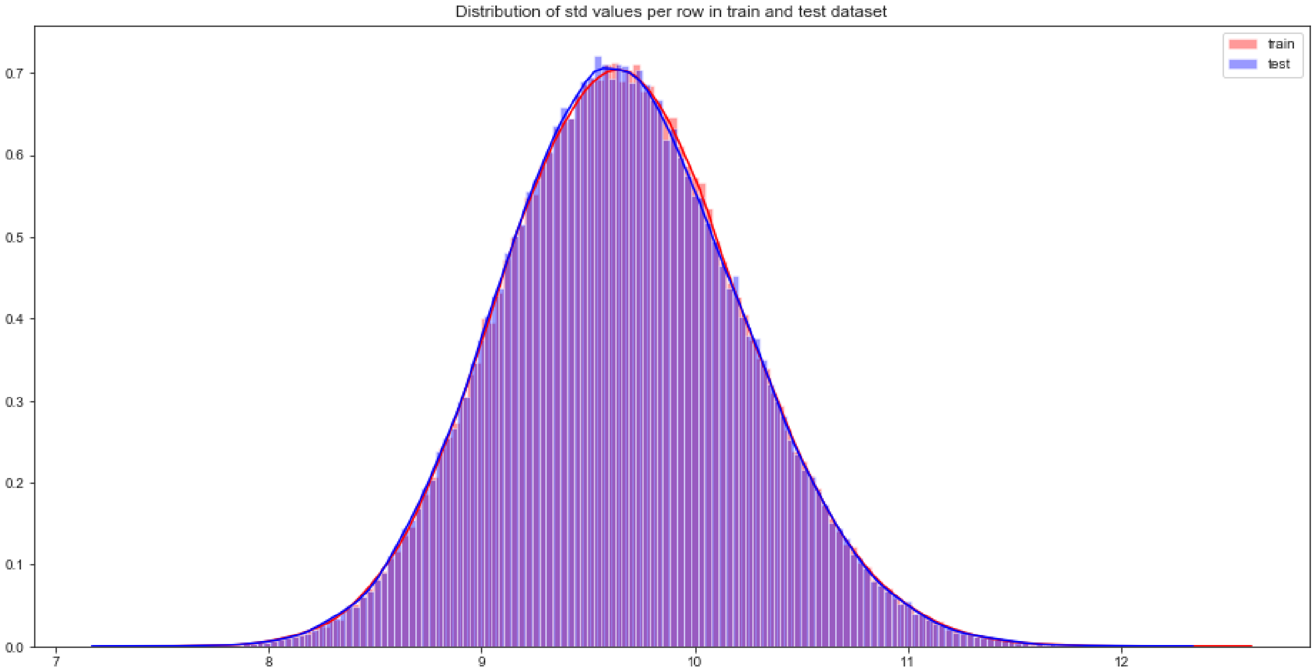
In [25]:

```
#Distribution of std values per column in train and test dataset
plt.figure(figsize=(16,8))
#train attributes
train_attributes=train_df.columns.values[2:202]
#test attributes
test_attributes=test_df.columns.values[1:201]
#Distribution plot for std values per column in train attributes
sns.distplot(train_df[train_attributes].std(axis=0),color='red',kde=True,bins=150,label='train')
#Distribution plot for std values per column in test attributes
sns.distplot(test_df[test_attributes].std(axis=0),color='blue',kde=True,bins=150,label='test')
plt.title('Distribution of std values per column in train and test dataset')
plt.legend()
plt.show()

#Distribution of std values per row in train and test dataset
plt.figure(figsize=(16,8))
#Distribution plot for std values per row in train attributes
sns.distplot(train_df[train_attributes].std(axis=1),color='red',kde=True,bins=150,label='train')
#Distribution plot for std values per row in test attributes
sns.distplot(test_df[test_attributes].std(axis=1),color='blue',kde=True, bins=150, label='test')
plt.title('Distribution of std values per row in train and test dataset')
plt.legend()
plt.show()
```

Distribution of std values per column in train and test dataset







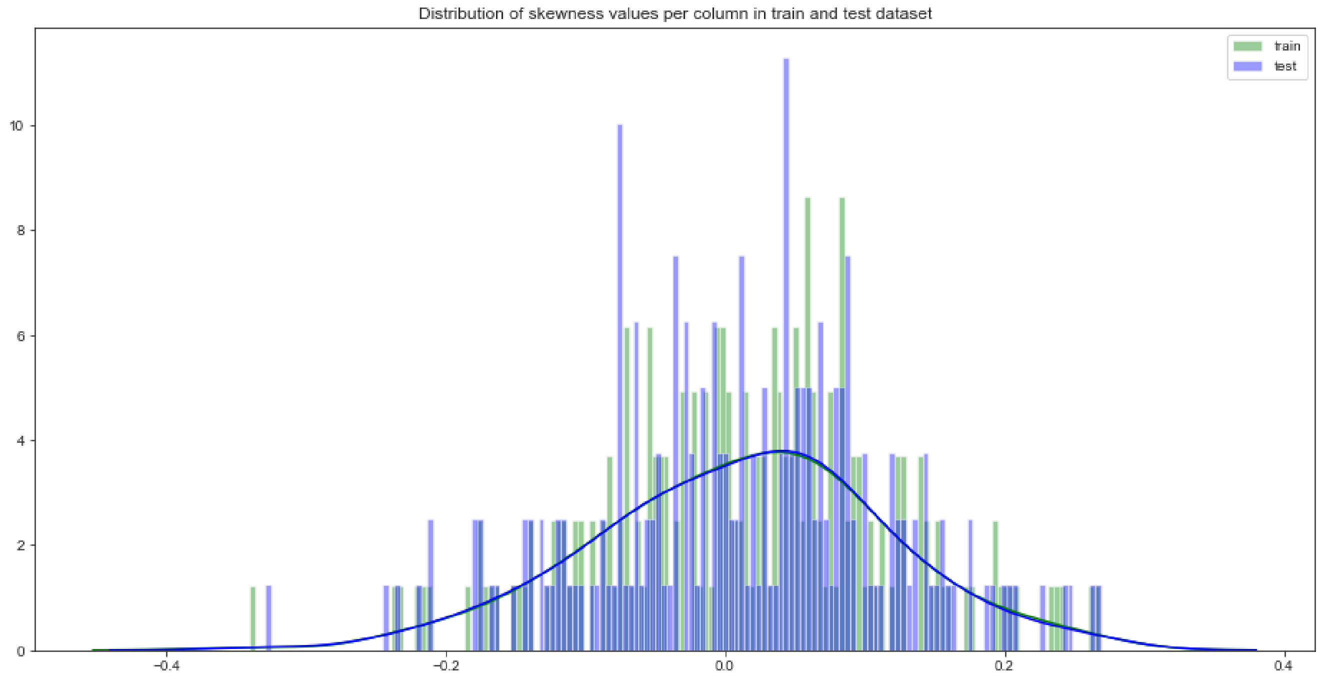
In [26]:

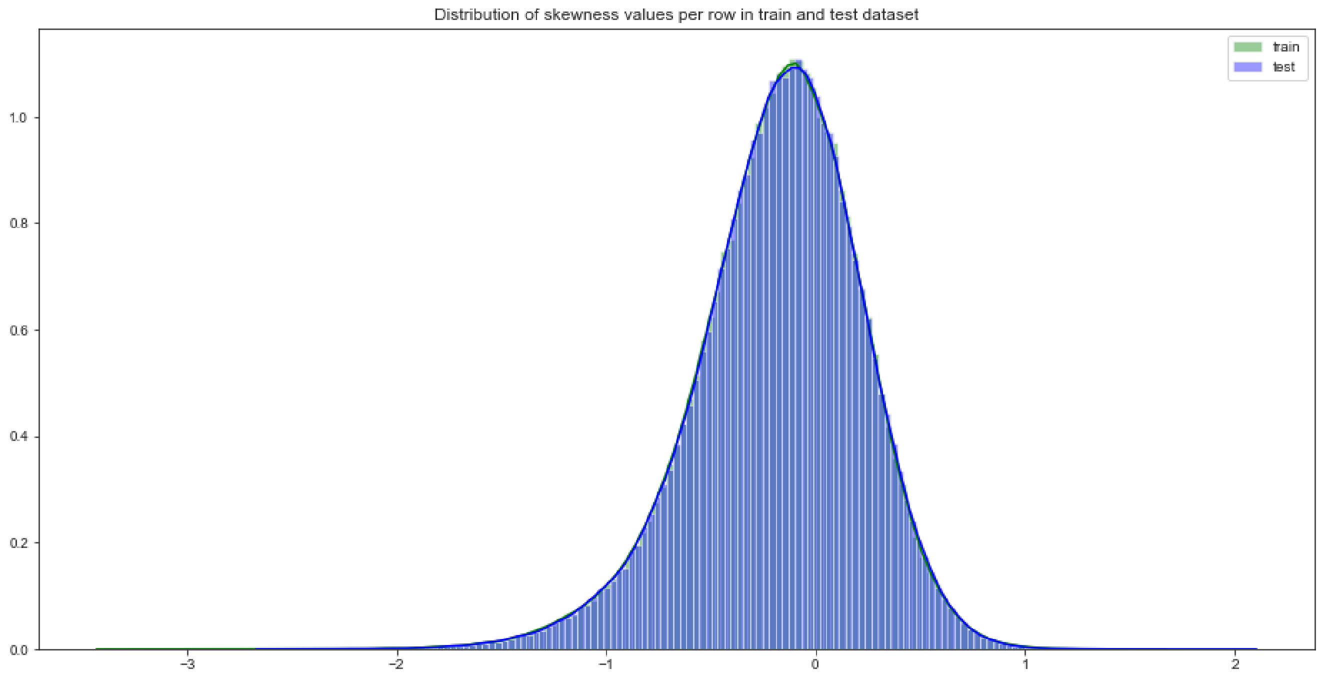
```
#Let us look distribution of skewness per rows and columns in train and test dataset
```

In [27]:

```
%%time
#Distribution of skew values per column in train and test dataset
plt.figure(figsize=(16,8))
#train attributes
train_attributes=train_df.columns.values[2:202]
#test attributes
test_attributes=test_df.columns.values[1:201]
#Distribution plot for skew values per column in train attributes
sns.distplot(train_df[train_attributes].skew(axis=0),color='green',kde=True,bins=150,label='train')
#Distribution plot for skew values per column in test attributes
sns.distplot(test_df[test_attributes].skew(axis=0),color='blue',kde=True,bins=150,label='test')
plt.title('Distribution of skewness values per column in train and test dataset')
plt.legend()
plt.show()

#Distribution of skew values per row in train and test dataset
plt.figure(figsize=(16,8))
#Distribution plot for skew values per row in train attributes
sns.distplot(train_df[train_attributes].skew(axis=1),color='green',kde=True,bins=150,label='train')
#Distribution plot for skew values per row in test attributes
sns.distplot(test_df[test_attributes].skew(axis=1),color='blue',kde=True, bins=150, label='test')
```

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






```
Wall time: 12 s
```

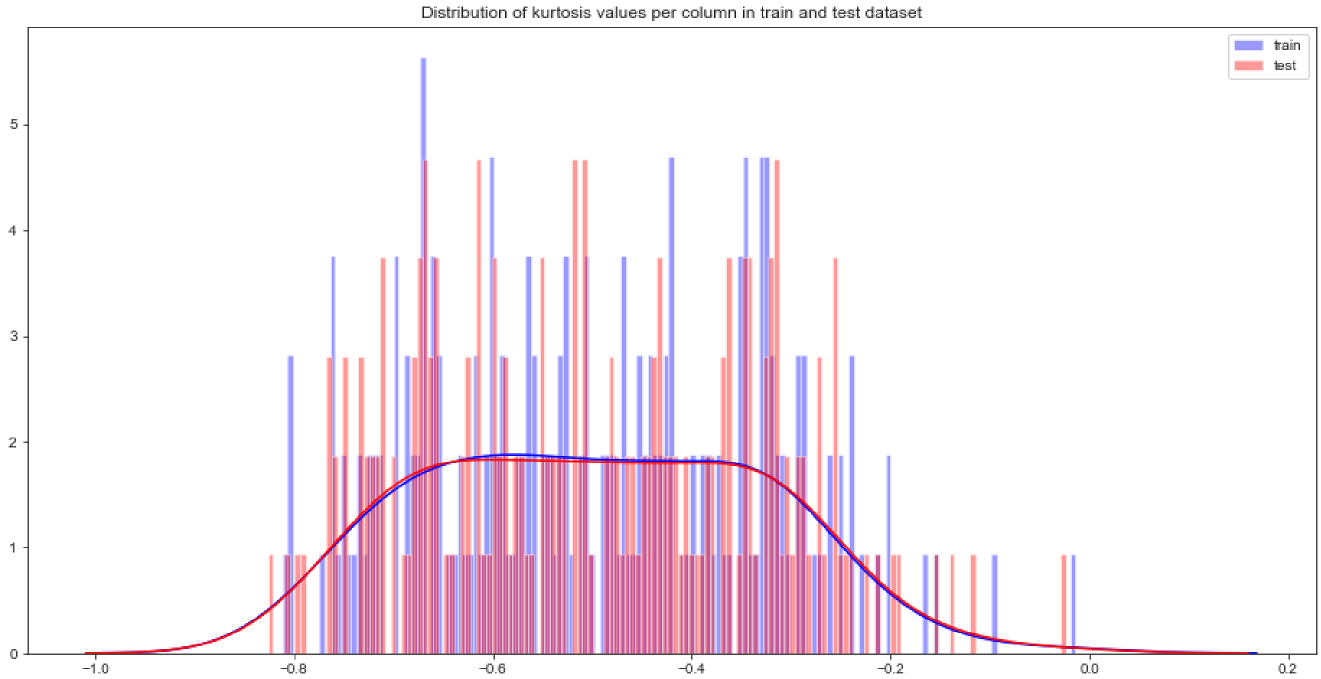
In [28]:

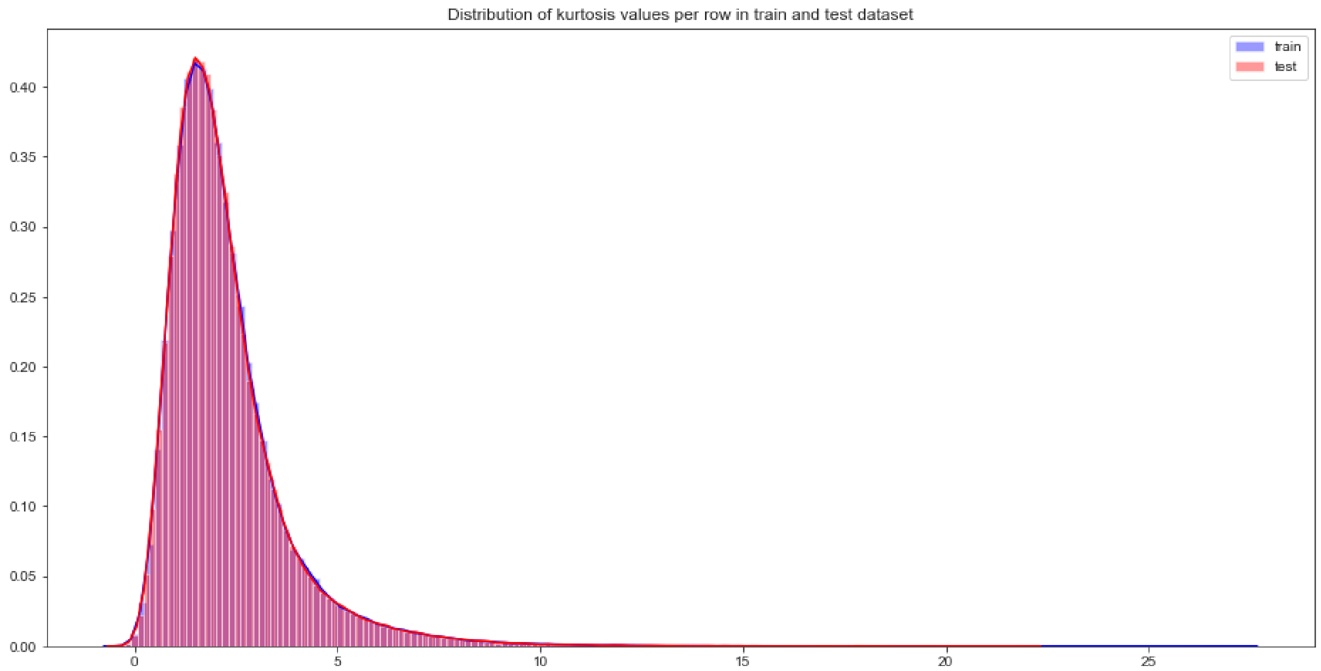
```
#Let us look distribution of kurtosis values per rows and columns in train and test dataset
```

In [29]:

```
%%time
#Distribution of kurtosis values per column in train and test dataset
plt.figure(figsize=(16,8))
#train attributes
train_attributes=train_df.columns.values[2:202]
#test attributes
test_attributes=test_df.columns.values[1:201]
#Distribution plot for kurtosis values per column in train attributes
sns.distplot(train_df[train_attributes].kurtosis(axis=0),color='blue',kde=True,bins=150,label='trai
```

```
n')
#Distribution plot for kurtosis values per column in test attributes
sns.distplot(test_df[test_attributes].kurtosis(axis=0),color='red',kde=True,bins=150,label='test')
plt.title('Distribution of kurtosis values per column in train and test dataset')
plt.legend()
plt.show()

#Distribution of kutosis values per row in train and test dataset
plt.figure(figsize=(16,8))
#Distribution plot for kurtosis values per row in train attributes
sns.distplot(train_df[train_attributes].kurtosis(axis=1),color='blue',kde=True,bins=150,label='trai
n')
#Distribution plot for kurtosis values per row in test attributes
sns.distplot(test_df[test_attributes].kurtosis(axis=1),color='red',kde=True, bins=150, label='test'
)
plt.title('Distribution of kurtosis values per row in train and test dataset')
plt.legend()
plt.show()
```






```
Wall time: 8.4 s
```

In [30]:

```
#Missing value analysis
```

In [31]:

```
%%time
#Finding the missing values in train and test data
train_missing=train_df.isnull().sum().sum()
test_missing=test_df.isnull().sum().sum()
print('Missing values in train data :',train_missing)
print('Missing values in test data :',test_missing)
```

```
Missing values in train data : 0
Missing values in test data : 0
Wall time: 1.29 s
```

In [33]:

```
%%time
#Correlations in train attributes
train_attributes=train_df.columns.values[2:202]
train_correlations=train_df[train_attributes].corr().abs().unstack().sort_values(kind='quicksort')
.reset_index()
train_correlations=train_correlations[train_correlations['level_0']!=train_correlations['level_1']
]
print(train_correlations.head(10))
print(train_correlations.tail(10))
```

```
   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
       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
Wall time: 18.4 s
```

In [35]:

```
%%time
#Correlations in test attributes
test_attributes=test_df.columns.values[1:201]
test_correlations=test_df[test_attributes].corr().abs().unstack().sort_values(kind='quicksort').re
set_index()
test_correlations=test_correlations[test_correlations['level_0']!=test_correlations['level_1']]
print(test_correlations.head(10))
print(test_correlations.tail(10))
```

```
   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
      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
Wall time: 18.3 s
```

In [ ]:

```
print("""


Correlation plot for train and test data

We can observed from correlation distribution plot that the correlation between the train and test attributes is very
very small, it means that features are independent each other.

""")
```

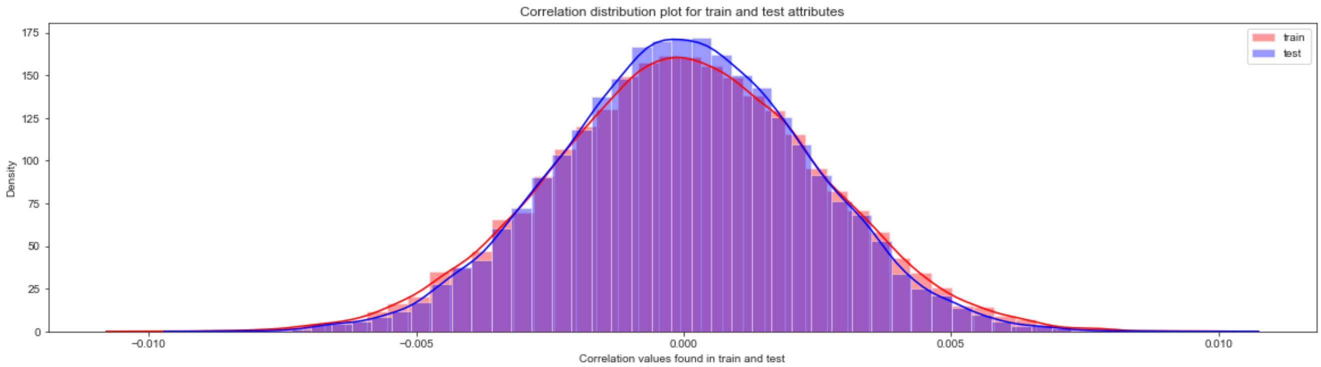
In [36]:

```
%%time
#Correlations in train data
train_correlations=train_df[train_attributes].corr()
train_correlations=train_correlations.values.flatten()
train_correlations=train_correlations[train_correlations!=1]
#Correlations in test data
test_correlations=test_df[test_attributes].corr()
test_correlations=test_correlations.values.flatten()
test_correlations=test_correlations[test_correlations!=1]

plt.figure(figsize=(20,5))
#Distribution plot for correlations in train data
sns.distplot(train_correlations, color="Red", label="train")
#Distribution plot for correlations in test data
sns.distplot(test_correlations, color="Blue", label="test")
plt.xlabel("Correlation values found in train and test")
plt.ylabel("Density")
plt.title("Correlation distribution plot for train and test attributes")
plt.legend()
```

```
Wall time: 39 s
```

Out[36]:

```
<matplotlib.legend.Legend at 0x25149244c50>
```



In [37]:

```
#training and testing data
X=train_df.drop(columns=['ID_code','target'],axis=1)
test=test_df.drop(columns=['ID_code'],axis=1)
y=train_df['target']
```

In [38]:

```
#Split the training data
X_train,X_valid,y_train,y_valid=train_test_split(X,y,random_state=42)

print('Shape of X_train :',X_train.shape)
print('Shape of X_valid :',X_valid.shape)
print('Shape of y_train :',y_train.shape)
print('Shape of y_valid :',y_valid.shape)
```

```
Shape of X_train : (150000, 200)
Shape of X_valid : (50000, 200)
Shape of y_train : (150000,)
Shape of y_valid : (50000,)
```

In [39]:

```
%%time
#Random forest classifier
rf_model=RandomForestClassifier(n_estimators=10,random_state=42)
#fitting the model
rf_model.fit(X_train,y_train)
```

```
Wall time: 1min 16s
```

Out[39]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=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)
```

In [40]:

```
%%time
#Permutation importance
from eli5.sklearn import PermutationImportance
perm_imp=PermutationImportance(rf_model,random_state=42)
#fitting the model
perm_imp.fit(X_valid,y_valid)
```

```
Wall time: 5min 2s
```

Out[40]:

```
PermutationImportance(cv='prefit',
                      estimator=RandomForestClassifier(bootstrap=True,
                                                       class_weight=None,
                                                       criterion='gini',
                                                       max_depth=None,
                                                       max_features='auto',
                                                       max_leaf_nodes=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)
```

```
n_iter=5, random_state=42, refit=True, scoring=None)
```

In [41]:

```
%%time
#Important features
eli5.show_weights(perm_imp,feature_names=X_valid.columns.tolist(),top=200)
```

```
Wall time: 103 ms
```

Out[41]:

Weight	Feature
0.0004 ± 0.0002	var_81
0.0003 ± 0.0002	var_146
0.0003 ± 0.0002	var_109
0.0003 ± 0.0002	var_12
0.0002 ± 0.0001	var_110
0.0002 ± 0.0000	var_173
0.0002 ± 0.0001	var_174
0.0002 ± 0.0002	var_0
0.0002 ± 0.0002	var_26
0.0001 ± 0.0001	var_166
0.0001 ± 0.0001	var_169
0.0001 ± 0.0001	var_22
0.0001 ± 0.0001	var_99
0.0001 ± 0.0001	var_53
0.0001 ± 0.0001	var_8
0.0001 ± 0.0001	var_1
0.0001 ± 0.0000	var_37
0.0001 ± 0.0003	var_133
0.0001 ± 0.0000	var_152
0.0001 ± 0.0001	var_175
0.0001 ± 0.0001	var_88
0.0001 ± 0.0001	var_66
0.0001 ± 0.0001	var_184
0.0001 ± 0.0000	var_95
0.0001 ± 0.0001	var_176
0.0001 ± 0.0001	var_74
0.0001 ± 0.0001	var_68
0.0001 ± 0.0001	var_34
0.0001 ± 0.0001	var_162
0.0001 ± 0.0001	var_36
0.0001 ± 0.0002	var_148
0.0001 ± 0.0001	var_71
0.0001 ± 0.0001	var_64
0.0001 ± 0.0000	var_112
0.0001 ± 0.0002	var_165
0.0001 ± 0.0001	var_128
0.0001 ± 0.0001	var_21
0.0001 ± 0.0001	var_108
0.0001 ± 0.0001	var_163
0.0001 ± 0.0001	var_177
0.0001 ± 0.0001	var_160
0.0001 ± 0.0002	var_2
0.0001 ± 0.0001	var_91
0.0001 ± 0.0001	var_167
0.0001 ± 0.0001	var_192
0.0001 ± 0.0001	var_32
0.0001 ± 0.0001	var_44
0.0001 ± 0.0002	var_76
0.0001 ± 0.0001	var_33
0.0001 ± 0.0001	var_195
0.0001 ± 0.0000	var_140
0.0001 ± 0.0001	var_78
0.0000 ± 0.0001	var_46
0.0000 ± 0.0001	var_63
0.0000 ± 0.0001	var_80
0.0000 ± 0.0000	var_84
0.0000 ± 0.0001	var_104
0.0000 ± 0.0000	var_145
0.0000 ± 0.0001	var_153
0.0000 ± 0.0001	var_117
0.0000 ± 0.0000	var_93
0.0000 ± 0.0001	var_183
0.0000 ± 0.0000	var_185
0.0000 ± 0.0001	var_56
0.0000 ± 0.0001	var_120
0.0000 ± 0.0001	var_41
0.0000 ± 0.0001	var_77
0.0000 ± 0.0001	var_190
0.0000 ± 0.0001	var_47
0.0000 ± 0.0002	var_90
0.0000 ± 0.0001	var_111
0.0000 ± 0.0000	var_85
0.0000 ± 0.0001	var_38
0.0000 ± 0.0001	var_86
0.0000 ± 0.0000	var_197
0.0000 ± 0.0001	var_69

Weight	Feature
0.0000 ± 0.0001	var_121
0.0000 ± 0.0001	var_49
0.0000 ± 0.0000	var_114
0.0000 ± 0.0000	var_16
0.0000 ± 0.0001	var_60
0.0000 ± 0.0001	var_54
0.0000 ± 0.0001	var_181
0.0000 ± 0.0001	var_82
0.0000 ± 0.0001	var_149
0.0000 ± 0.0001	var_62
0.0000 ± 0.0001	var_98
0.0000 ± 0.0001	var_199
0.0000 ± 0.0002	var_6
0.0000 ± 0.0002	var_67
0.0000 ± 0.0001	var_178
0.0000 ± 0.0001	var_182
0.0000 ± 0.0001	var_50
0.0000 ± 0.0001	var_141
0.0000 ± 0.0001	var_52
0.0000 ± 0.0001	var_96
0.0000 ± 0.0001	var_147
0.0000 ± 0.0001	var_97
0.0000 ± 0.0001	var_123
0.0000 ± 0.0000	var_75
0.0000 ± 0.0001	var_118
0.0000 ± 0.0000	var_100
0.0000 ± 0.0000	var_57
0.0000 ± 0.0001	var_194
0.0000 ± 0.0001	var_27
0.0000 ± 0.0001	var_58
0.0000 ± 0.0000	var_116
0.0000 ± 0.0001	var_29
0.0000 ± 0.0001	var_168
0.0000 ± 0.0000	var_48
0.0000 ± 0.0001	var_24
0.0000 ± 0.0001	var_170
0.0000 ± 0.0001	var_188
0.0000 ± 0.0001	var_28
-0.0000 ± 0.0001	var_115
-0.0000 ± 0.0001	var_42
-0.0000 ± 0.0001	var_20
-0.0000 ± 0.0001	var_127
-0.0000 ± 0.0001	var_107
-0.0000 ± 0.0000	var_150
-0.0000 ± 0.0000	var_79
-0.0000 ± 0.0001	var_159
-0.0000 ± 0.0000	var_137
-0.0000 ± 0.0001	var_11
-0.0000 ± 0.0001	var_65
-0.0000 ± 0.0000	var_73
-0.0000 ± 0.0001	var_103
-0.0000 ± 0.0001	var_31
-0.0000 ± 0.0000	var_30
-0.0000 ± 0.0001	var_17
-0.0000 ± 0.0001	var_138
-0.0000 ± 0.0001	var_9
-0.0000 ± 0.0001	var_87
-0.0000 ± 0.0002	var_191
-0.0000 ± 0.0001	var_119
-0.0000 ± 0.0001	var_10
-0.0000 ± 0.0000	var_45
-0.0000 ± 0.0001	var_59
-0.0000 ± 0.0001	var_113
-0.0000 ± 0.0001	var_72
-0.0000 ± 0.0001	var_70
-0.0000 ± 0.0000	var_189
-0.0000 ± 0.0000	var_164
-0.0000 ± 0.0001	var_136
-0.0000 ± 0.0000	var_25
-0.0000 ± 0.0001	var_23
-0.0000 ± 0.0001	var_142
-0.0000 ± 0.0000	var_143
-0.0000 ± 0.0001	var_124
-0.0000 ± 0.0000	var_131
-0.0000 ± 0.0001	var_126
-0.0000 ± 0.0001	var_151
-0.0000 ± 0.0001	var_155
-0.0000 ± 0.0001	var_158
-0.0000 ± 0.0001	var_187
-0.0000 ± 0.0001	var_61
-0.0000 ± 0.0001	var_171
-0.0000 ± 0.0001	var_7
-0.0000 ± 0.0001	var_161
-0.0000 ± 0.0001	var_18
-0.0000 ± 0.0001	var_102
-0.0000 ± 0.0001	var_15
-0.0000 ± 0.0001	var_39
-0.0000 ± 0.0003	var_139
-0.0000 ± 0.0001	var_129
-0.0000 ± 0.0001	var_180
-0.0000 ± 0.0001	var_19
-0.0000 ± 0.0001	var_55
-0.0000 ± 0.0001	var_157
-0.0000 ± 0.0000	var_101

Weight	Feature
-0.0000 ± 0.0001	var_135
-0.0000 ± 0.0001	var_130
-0.0000 ± 0.0001	var_106
-0.0000 ± 0.0001	var_13
-0.0000 ± 0.0001	var_105
-0.0000 ± 0.0001	var_51
-0.0000 ± 0.0001	var_83
-0.0000 ± 0.0001	var_40
-0.0000 ± 0.0000	var_132
-0.0000 ± 0.0001	var_122
-0.0001 ± 0.0001	var_43
-0.0001 ± 0.0001	var_35
-0.0001 ± 0.0001	var_14
-0.0001 ± 0.0001	var_186
-0.0001 ± 0.0000	var_144
-0.0001 ± 0.0001	var_5
-0.0001 ± 0.0001	var_193
-0.0001 ± 0.0001	var_196
-0.0001 ± 0.0000	var_3
-0.0001 ± 0.0001	var_92
-0.0001 ± 0.0001	var_89
-0.0001 ± 0.0001	var_94
-0.0001 ± 0.0001	var_154
-0.0001 ± 0.0002	var_179
-0.0001 ± 0.0001	var_156
-0.0001 ± 0.0001	var_134
-0.0001 ± 0.0001	var_4
-0.0001 ± 0.0001	var_125
-0.0001 ± 0.0001	var_172
-0.0002 ± 0.0003	var_198

In [42]:

```
print("""
Take aways:

Importance of the features decreases as we move down the top of the column.
As we can see the features shown in green indicate that they have a positive impact on our predict
ion
As we can see the features shown in white indicate that they have no effect on our prediction
As we can see the features shown in red indicate that they have a negative impact on our predictio
n
The most important feature is 'Var_81'
""")
```

```
Take aways:

Importance of the features decreases as we move down the top of the column.
As we can see the features shown in green indicate that they have a positive impact on our predict
ion
As we can see the features shown in white indicate that they have no effect on our prediction
As we can see the features shown in red indicate that they have a negative impact on our predictio
n
The most important feature is 'Var_81'
```

In [43]:

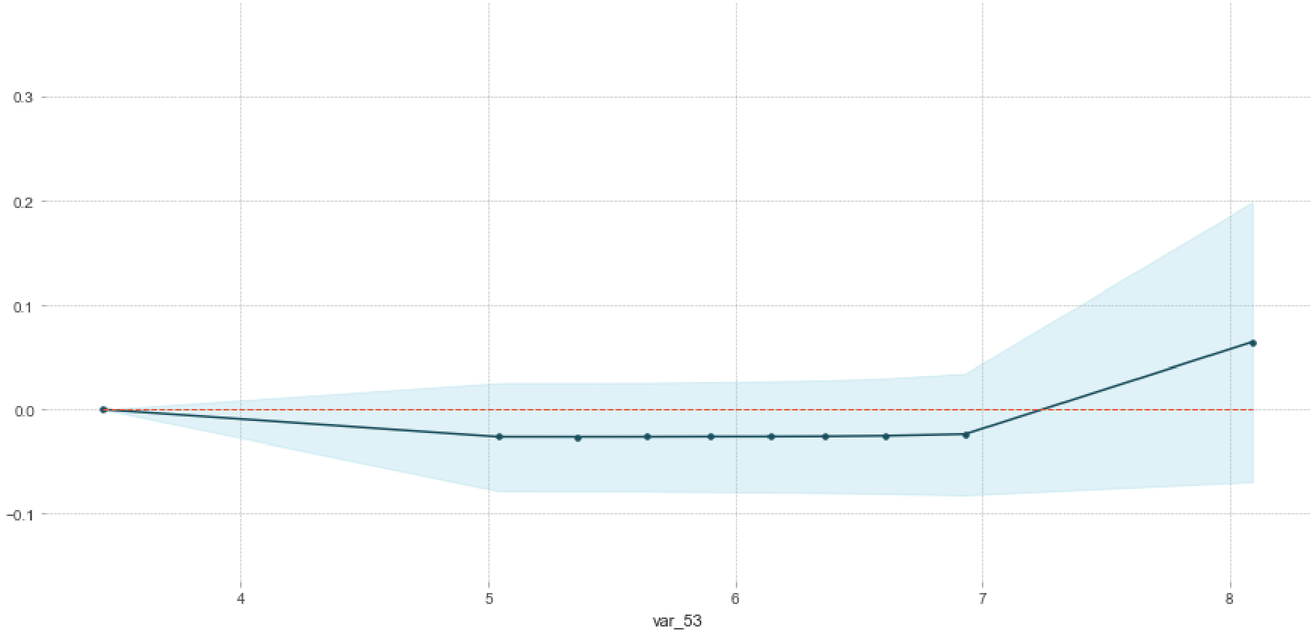
```
#Partial dependence plot
```

In [44]:

```
%%time
#Create the data we will plot 'var_53'
features=[v for v in X_valid.columns if v not in ['ID_code','target']]
pdp_data=pdp.pdp_isolate(rf_model,dataset=X_valid,model_features=features,feature='var_53')
#plot feature "var_53"
pdp.pdp_plot(pdp_data,'var_53')
plt.show()
```

PDP for feature "var_53"

Number of unique grid points: 10




```
Wall time: 5.21 s
```

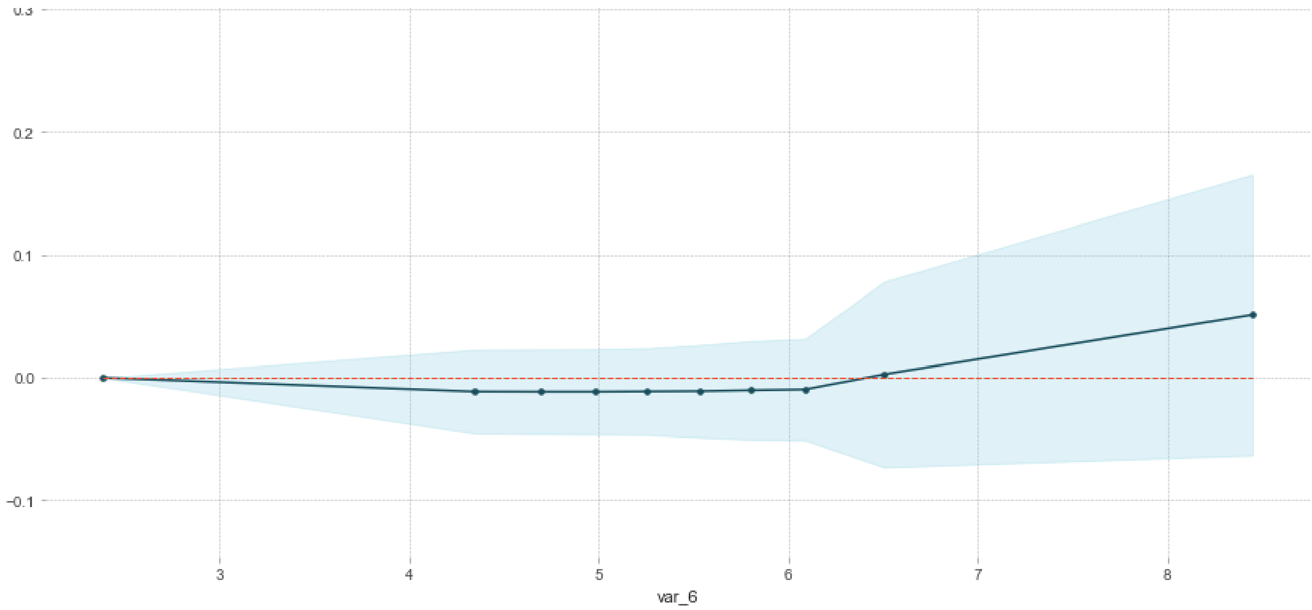
In [45]:

```
print("""

Take aways:

The y_axis does not show the predictor value instead how the value changing with the change in giv
en predictor variable.
The blue shaded area indicates the level of confidence of 'var_53'.
On y-axis having a positive value means for that particular value of predictor variable it is less
likely to predict the
correct class and having a positive value means it has positive impact on predicting the correct c
lass.

""")
```

```
Take aways:

The y_axis does not show the predictor value instead how the value changing with the change in giv
en predictor variable.
The blue shaded area indicates the level of confidence of 'var_53'.
On y-axis having a positive value means for that particular value of predictor variable it is less
likely to predict the
correct class and having a positive value means it has positive impact on predicting the correct c
lass.
```

In [46]:

```
%%time
#Create the data we will plot
pdp_data=pdp.pdp_isolate(rf_model,dataset=X_valid,model_features=features,feature='var_6')
#plot feature "var_6"
pdp.pdp_plot(pdp_data,'var_6')
plt.show()
```




```
Wall time: 5.4 s
```

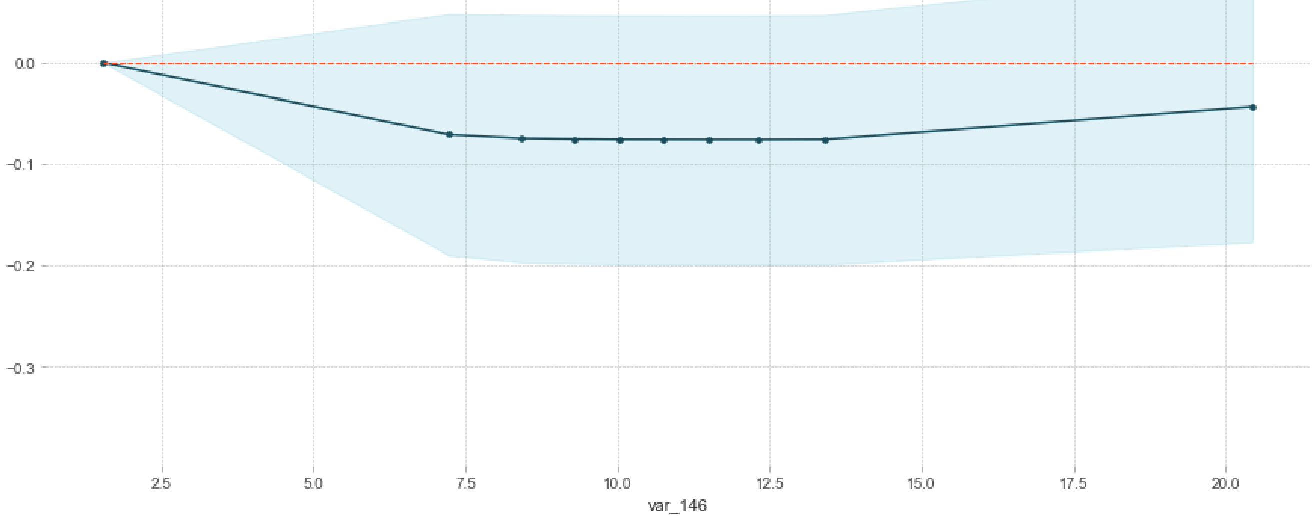
In [47]:

```
print("""
Take aways:

The y_axis does not show the predictor value instead how the value changing with the change in giv
en predictor variable.
The blue shaded area indicates the level of confidence of 'var_6'.
On y-axis having a positive value means for that particular value of predictor variable it is less
likely to predict the
correct class and having a positive value means it has positive impact on predicting the correct c
lass.

""")
```

```
Take aways:

The y_axis does not show the predictor value instead how the value changing with the change in giv
en predictor variable.
The blue shaded area indicates the level of confidence of 'var_6'.
On y-axis having a positive value means for that particular value of predictor variable it is less
likely to predict the
correct class and having a positive value means it has positive impact on predicting the correct c
lass.
```

In [48]:

```
%%time
#Create the data we will plot
pdp_data=pdp.pdp_isolate(rf_model,dataset=X_valid,model_features=features,feature='var_146')
#plot feature "var_146"
pdp.pdp_plot(pdp_data,'var_146')
plt.show()
```




```
Wall time: 5.17 s
```

In [49]:

```
print("""
Take aways:

The y_axis does not show the predictor value instead how the value changing with the change in giv
en predictor variable.
The blue shaded area indicates the level of confidence of 'var_146'.
On y-axis having a positive value means for that particular value of predictor variable it is less
likely to predict the
correct class and having a positive value means it has positive impact on predicting the correct c
lass.
""")
```

```
Take aways:

The y_axis does not show the predictor value instead how the value changing with the change in giv
en predictor variable.
The blue shaded area indicates the level of confidence of 'var_146'.
On y-axis having a positive value means for that particular value of predictor variable it is less
likely to predict the
correct class and having a positive value means it has positive impact on predicting the correct c
lass.
```

In [50]:

```
#Split the train data using StratefiedKFold cross validator
#Training data
X=train_df.drop(['ID_code','target'],axis=1)
Y=train_df['target']
#StratifiedKFold cross validator
cv=StratifiedKFold(n_splits=5,random_state=42,shuffle=True)
for train_index,valid_index in cv.split(X,Y):
    X_train, X_valid=X.iloc[train_index], X.iloc[valid_index]
    y_train, y_valid=Y.iloc[train_index], Y.iloc[valid_index]

print('Shape of X_train :',X_train.shape)
print('Shape of X_valid :',X_valid.shape)
print('Shape of y_train :',y_train.shape)
print('Shape of y_valid :',y_valid.shape)
```

```
Shape of X_train : (160001, 200)
Shape of X_valid : (39999, 200)
Shape of y_train : (160001,)
Shape of y_valid : (39999,)
```

In [51]:

```
%%time
```

```
#Logistic regression model
lr_model=LogisticRegression(random_state=42)
#fitting the lr model
lr_model.fit(X_train,y_train)
```

```
Wall time: 3min 18s
```

Out[51]:

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

In [52]:

```
#Accuracy of the model
lr_score=lr_model.score(X_train,y_train)
print('Accuracy of the lr_model :',lr_score)
```

```
Accuracy of the lr_model : 0.9149255317154268
```

In [53]:

```
%%time
#Cross validation prediction
cv_predict=cross_val_predict(lr_model,X_valid,y_valid,cv=5)
#Cross validation score
cv_score=cross_val_score(lr_model,X_valid,y_valid,cv=5)
print('cross_val_score :',np.average(cv_score))
```

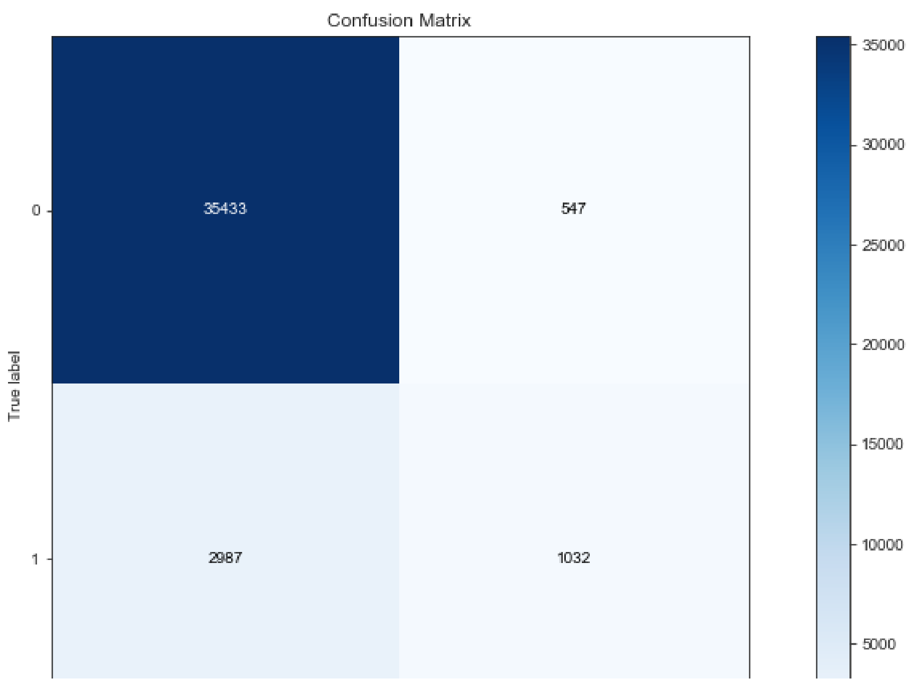
```
cross_val_score : 0.911647852856607
Wall time: 5min 51s
```

In [54]:

```
#Confusion matrix
cm=confusion_matrix(y_valid,cv_predict)
#Plot the confusion matrix
plot_confusion_matrix(y_valid,cv_predict,normalize=False,figsize=(15,8))
```

Out[54]:

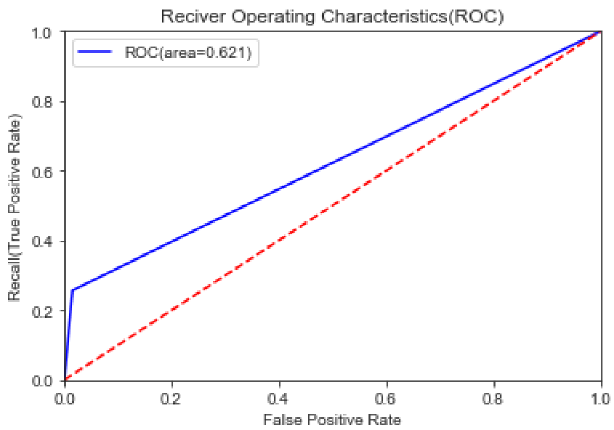
```
<matplotlib.axes._subplots.AxesSubplot at 0x2510d856c18>
```



In [55]:

```
#ROC_AUC score
roc_score=roc_auc_score(y_valid,cv_predict)
print('ROC score :',roc_score)

#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(y_valid,cv_predict)
roc_auc=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc)
```

```
ROC score : 0.6207887015553276
```




```
AUC: 0.6207887015553276
```

In [56]:

```
print("When we compare the roc_auc_score and model accuracy , model is not performing well on imba
lanced data.")
```

```
When we compare the roc_auc_score and model accuracy , model is not performing well on imbalanced
data.
```

In [57]:

```
#Classification report
scores=classification_report(y_valid,cv_predict)
print(scores)
```

```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95     35980
           1       0.65      0.26      0.37      4019

    accuracy                           0.91     39999
   macro avg       0.79      0.62      0.66     39999
weighted avg       0.90      0.91      0.89     39999
```

In [58]:

```
print("""
We can observed that f1 score is high for number of customers those who will not make a transactio
n than the who will make
a transaction. So, we are going to change the algorithm.

""")
```

```
We can observed that f1 score is high for number of customers those who will not make a
transaction than the who will make
a transaction. So, we are going to change the algorithm.
```

In [59]:

```
%%time
#Predicting the model
X_test=test_df.drop(['ID_code'],axis=1)
lr_pred=lr_model.predict(X_test)
print(lr_pred)
```

```
[0 0 0 ... 0 0 0]
Wall time: 303 ms
```

In [60]:

```
print("""
Synthetic Minority Oversampling Technique(SMOTE)

SMOTE uses a nearest neighbors algorithm to generate new and synthetic data to used for training t
he model.
""")
```

```
Synthetic Minority Oversampling Technique(SMOTE)

SMOTE uses a nearest neighbors algorithm to generate new and synthetic data to used for training t
he model.
```

In [61]:

```
%%time
from imblearn.over_sampling import SMOTE
#Synthetic Minority Oversampling Technique
sm = SMOTE(random_state=42, ratio=1.0)
#Generating synthetic data points
X_smote,y_smote=sm.fit_sample(X_train,y_train)
X_smote_v,y_smote_v=sm.fit_sample(X_valid,y_valid)
```

```
Wall time: 2min 27s
```

In [62]:

```
%%time
#Logistic regression model for SMOTE
smote=LogisticRegression(random_state=42)
#fitting the smote model
smote.fit(X_smote,y_smote)
```

```
Wall time: 11min 4s
```

Out[62]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
```

```
                       random_state=42, solver='warn', tol=0.0001, verbose=0,
                       warm_start=False)
```

In [63]:

```
#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 : 0.7986027153597087
```

In [64]:

```
%%time
#Cross validation prediction
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_val_score :',np.average(cv_score))
```

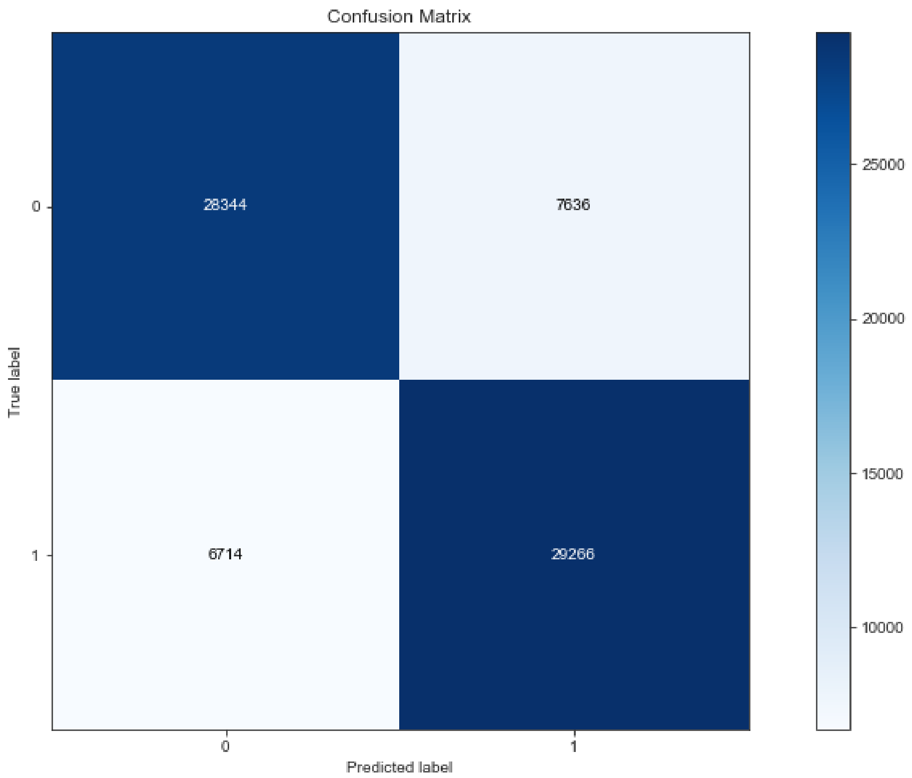
```
cross_val_score : 0.8005836575875487
Wall time: 16min 10s
```

In [65]:

```
#Confusion matrix
cm=confusion_matrix(y_smote_v,cv_pred)
#Plot the confusion matrix
plot_confusion_matrix(y_smote_v,cv_pred,normalize=False,figsize=(15,8))
```

Out[65]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2514a0f5470>
```

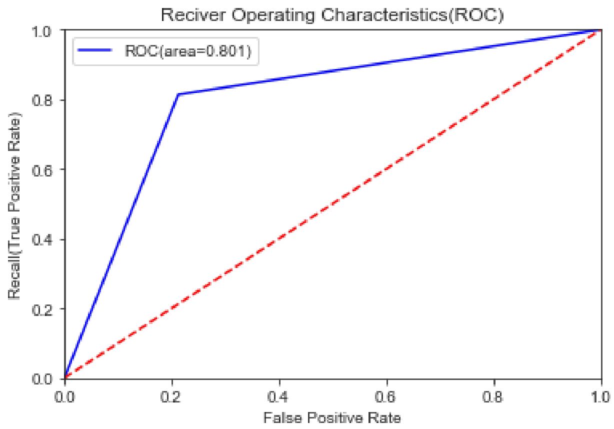


In [66]:

```
#ROC_AUC score
roc_score=roc_auc_score(y_smote_v,cv_pred)
print('ROC score :',roc_score)

#ROC_AUC curve
```

```
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(y_smote_v,cv_pred)
roc_auc=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc)
```

```
ROC score : 0.8005836575875487
```




```
AUC: 0.8005836575875487
```

In [67]:

```
#Classification report
scores=classification_report(y_smote_v,cv_pred)
print(scores)
```

```
              precision    recall  f1-score   support

           0       0.81      0.79      0.80     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
```

In [68]:

```
%%time
#Predicting the model
X_test=test_df.drop(['ID_code'],axis=1)
smote_pred=smote.predict(X_test)
print(smote_pred)
```

```
[1 1 0 ... 0 0 1]
Wall time: 1.19 s
```

In [69]:

```
print("""
We can observed that smote model is performing well on imbalance data compare to logistic regressi
on.
""")
```

```
We can observed that smote model is performing well on imbalance data compare to logistic
```

regression.

In [70]:

```
print("""
LightGBM:

LightGBM is a gradient boosting framework that uses tree based learning algorithms. We are going t
o use LightGBM model.
""")
```

```
LightGBM:

LightGBM is a gradient boosting framework that uses tree based learning algorithms. We are going t
o use LightGBM model.
```

In [71]:

```
#Training the model
#training data
lgb_train=lgb.Dataset(X_train,label=y_train)
#validation data
lgb_valid=lgb.Dataset(X_valid,label=y_valid)
```

In [72]:

```
#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 [73]:

```
num_rounds=10000
lgbm= lgb.train(params,lgb_train,num_rounds,valid_sets=[lgb_train,lgb_valid],verbose_eval=1000,ear
ly_stopping_rounds = 5000)
lgbm
```

```
Training until validation scores don't improve for 5000 rounds
[1000] training's auc: 0.938996 valid_1's auc: 0.885963
[2000] training's auc: 0.958629 valid_1's auc: 0.890769
[3000] training's auc: 0.972001 valid_1's auc: 0.89195
[4000] training's auc: 0.981625 valid_1's auc: 0.892447
[5000] training's auc: 0.988357 valid_1's auc: 0.892444
[6000] training's auc: 0.992858 valid_1's auc: 0.892633
[7000] training's auc: 0.995834 valid_1's auc: 0.892332
[8000] training's auc: 0.997652 valid_1's auc: 0.89205
[9000] training's auc: 0.99874 valid_1's auc: 0.891803
[10000] training's auc: 0.999366 valid_1's auc: 0.891481
Did not meet early stopping. Best iteration is:
[10000] training's auc: 0.999366 valid_1's auc: 0.891481
```

Out[73]:

<lightgbm.basic.Booster at 0x2510e63cba8>

In [ ]:

```

```

In [74]:

```
X_test=test_df.drop(['ID_code'],axis=1)
#predict the model
#probability predictions
lgbm_predict_prob=lgbm.predict(X_test,random_state=42,num_iteration=lgbm.best_iteration)
#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)
```

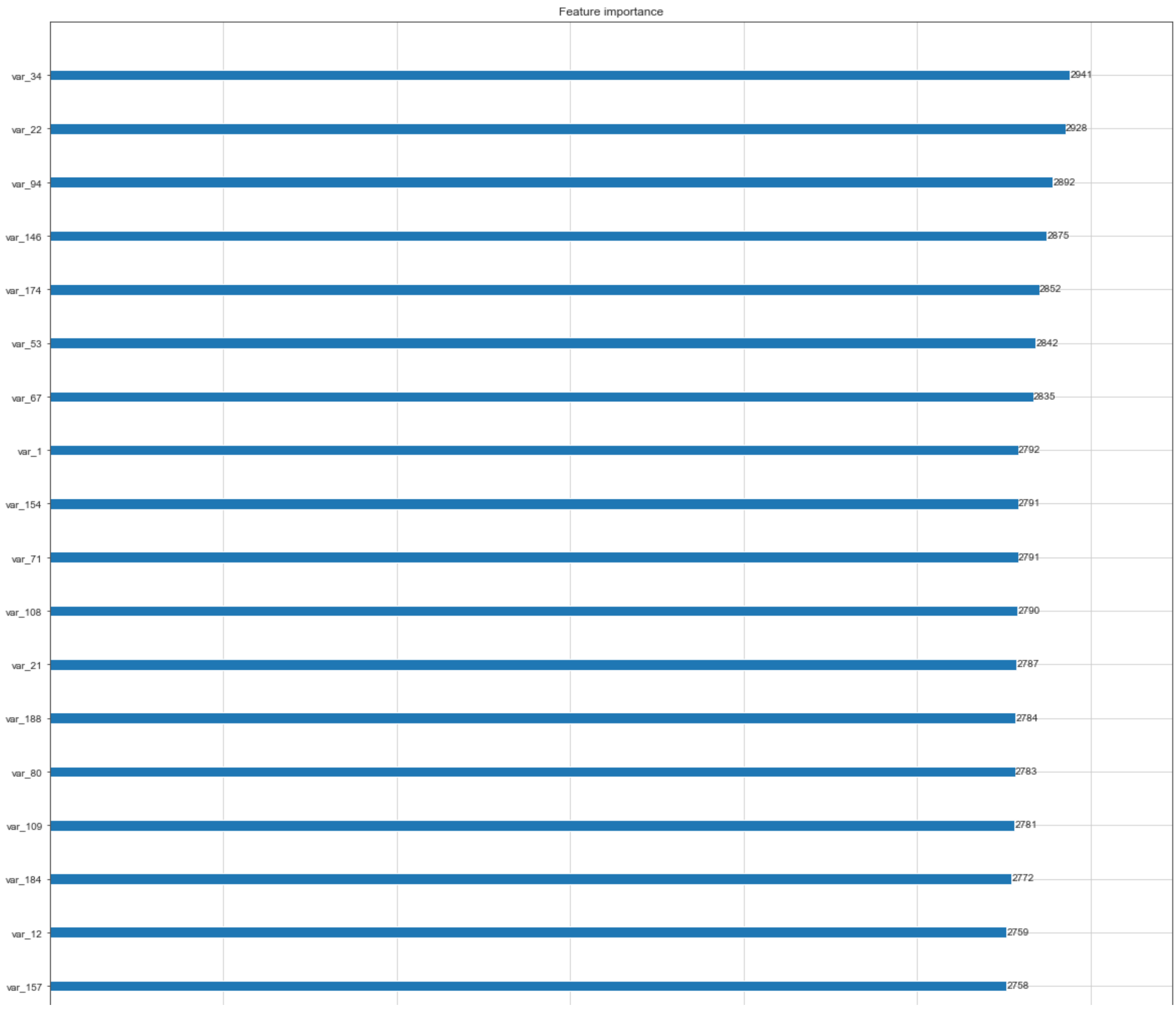
```
[0.32983774 0.35499097 0.33798803 ... 0.01180197 0.25520441 0.20054849]
[0 0 0 ... 0 0 0]
```

In [75]:

```
#plot the important features
lgb.plot_importance(lgbm,max_num_features=50,importance_type="split",figsize=(20,50))
```

Out[75]:

<matplotlib.axes._subplots.AxesSubplot at 0x2514a2d9da0>



var_127
var_44
var_137
var_2
var_99
var_177
var_155
var_51
var_131
var_173
var_199
var_75
var_179
var_148
var_18
var_52
var_6
var_123
var_9
var_119
var_110
var_13
var_170
var_40
var_49
var_95
var_191
var_180
var_78
var_107
var_196
var_130
2751
2747
2744
2739
2727
2725
2711
2710
2706
2702
2696
2685
2680
2673
2671
2667
2667
2666
2658
2657
2657
2654
2652
2649
2648
2645
2642
2642
2641
2633
2630
2628
Features
0
500
1000
1500
2000
2500
3000
Feature importance

In [76]:

```
print("""
Conclusion :

We tried model with logistic regression,smote and lightgbm. But lightgbm model is performing well
on imbalanced data
compared to other models based on scores of roc_auc_score.
""")
```

```
Conclusion :

We tried model with logistic regression,smote and lightgbm. But lightgbm model is performing well
on imbalanced data
compared to other models based on scores of roc_auc_score.
```

In [77]:

```
#final submission
sub_df=pd.DataFrame({'ID_code':test_df['ID_code'].values})
sub_df['lgbm_predict_prob']=lgbm_predict_prob
sub_df['lgbm_predict']=lgbm_predict
sub_df.to_csv('submission.csv',index=False)
sub_df.head()
```

Out[77]:

	ID_code	lgbm_predict_prob	lgbm_predict
0	test_0	0.329838	0
1	test_1	0.354991	0
2	test_2	0.337988	0
3	test_3	0.436098	0
4	test_4	0.102491	0

In [ ]:

In [ ]:

In [ ]:

In [ ]: