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.

Data: https://www.kaggle.com/c/santander-customer-transaction-prediction/overview/evaluation

Reference Link: https://www.youtube.com/watch?v=LEWpRlaEJO8

Business problem

Based on the data given for every user we have to predict wheater the user will do the transaction in future or not.

ML problem question

- This one is a classical Binary classification machine learing problem.
- There are 2 classes in the target 0 and 1.
- 0 means not do transaction and 1 means will do transaction.

Evaluation Metrix

The Evaluation metrix is the AUC and we have to make sure to get AUC as high as possible.

Mounting the Drive

In [0]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:

```
cd drive/My\ Drive/self_case_study_1
```

```
[Errno 2] No such file or directory: 'drive/My Drive/self_case_study_1'
/content/drive/My Drive/self_case_study_1
```

Loading the data from the kaggle

In [0]:

```
!wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36" --header="Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3" --header="Accept-Language: en-GB,en-US;q=0.9,en;q=0.8" --header="Referer: https://www.kaggle.com/" "https://storage.googleapis.com/kaggle-competitions-data/kaggle/10385/298493/train.csv.zip?GoogleAccessId=web-data@kaggle-161607.iam.gserviceaccount.com&Expires=1575271319&Signature=MR8AlCGs9Whsd9zMt3Tm%2BEIWiLFt2ehTuwOOqU8qRb6aqSqRs0b3Vs%2FqKFzrLqbB%2BJYOYxB3sG2BWI0lOwFa5x7bvqAwsVbGd32E6W9JT%2B3uRiful
```

```
Dgx%2F%2FmON00MT8Uzx0d%2BADCWBPhbm%2FFCpeyrJLMcvMyC5YcHfAs77UN9wolOi3G9tVfs1UQO9KA7gRLXjVtPqqfDJD
9fR6Cipcyhv4OCiNJlr%2FN835eyyLgii961FhYSrazlKZvK1xNW7KAZGB8OorFk3a8mdeXvUoGeYpdfzLEgp2jHr7QDrbWdomBbNBrI
87kP1o3rwWpkx3%2BHTpxwei6hg2qa7y18z12zPg%3D%3D" -O "train.csv.zip" -c
```

```
--2019-11-29 07:32:38--  https://storage.googleapis.com/kaggle-competitions-data/kaggle/10385/298493/train.csv.zip?GoogleAccessI
d=web-data@kaggle-161607.iam.gserviceaccount.com&Expires=1575271319&Signature=MR8AlCGs9Whsd9zMt3Tm%2BEIWiLFt2e
hTuwOOgU8qRb6aqSgRs0b3Vs%2FgKFzrLqbB%2BJYQYxB3sG2BWI0lOwFa5x7bvgAwsVbGd32E6W9JT%2B3uRjfuIDgx%2F%2F
mON00MT8Uzx0d%2BADCWBPhbm%2FFCpeyrJLMcvMyC5YcHfAs77UN9wolOi3G9tVfs1UQO9KA7gRLXjVtPqqfDJD9fR6Cipcyhv
4OCiNJlr%2FN835eyyLgii961FhYSrazlKZvK1xNW7KAZGB8OorFk3a8mdeXvUoGeYpdfzLEgp2jHr7QDrbWdomBbNBrI87kP1o3rwW
pkx3%2BHTpxwei6hg2qa7y18z12zPg%3D%3D
Resolving storage.googleapis.com (storage.googleapis.com)... 173.194.203.128, 2607:f8b0:400e:c07::80
Connecting to storage.googleapis.com (storage.googleapis.com)|173.194.203.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 128240759 (122M) [application/zip]
Saving to: ‘train.csv.zip’

train.csv.zip       100%[===================>] 122.30M  90.3MB/s    in 1.4s

2019-11-29 07:32:39 (90.3 MB/s) - ‘train.csv.zip’ saved [128240759/128240759]
```

Extracting the data

In [0]:

```
!7z e train.csv.zip
```

```
7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,2 CPUs Intel(R) Xeon(R) CPU @ 2.20GHz (406F0),ASM,A
ES-NI)

Scanning the drive for archives:
  0M Scan         1 file, 128240759 bytes (123 MiB)

Extracting archive: train.csv.zip
--
Path = train.csv.zip
Type = zip
Physical Size = 128240759

  0%      4% - train.csv                  9% - train.csv                 14% - train.csv                18% - train.csv                 23% - train.csv
 28% - train.csv                 34% - train.csv                 39% - train.csv                 43% - train.csv                 47% - train.csv                 52
% - train.csv                 56% - train.csv                 61% - train.csv                 66% - train.csv                 71% - train.csv                 76% -
train.csv                 81% - train.csv                 85% - train.csv                 90% - train.csv                 94% - train.csv                 99% - trai
n.csv                 Everything is Ok

Size:       302133017
Compressed: 128240759
```

In [0]:

```
!7z e test.csv.zip
```

```
7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,2 CPUs Intel(R) Xeon(R) CPU @ 2.30GHz (306F0),ASM,A
ES-NI)

Scanning the drive for archives:
  0M Scan         1 file, 127915900 bytes (122 MiB)

Extracting archive: test.csv.zip
--
Path = test.csv.zip
Type = zip
Physical Size = 127915900

  0%      3% - test.csv                  5% - test.csv                  9% - test.csv                 12% - test.csv                 15% - test.csv                 19%
- test.csv                 21% - test.csv                 24% - test.csv                 27% - test.csv                 31% - test.csv                 36% - test.csv
 40% - test.csv                 44% - test.csv                 48% - test.csv                 51% - test.csv                 56% - test.csv                 60% - test
.csv                 64% - test.csv                 68% - test.csv                 72% - test.csv                 76% - test.csv                 80% - test.csv
 85% - test.csv                 89% - test.csv                 93% - test.csv                 97% - test.csv                 Everything is Ok
```

Size: 301526706
Compressed: 127915900

Importing the libraries

In [0]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc,roc_auc_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RandomizedSearchCV
import lightgbm as lgb
import timeit
import time
```

In [58]:

```
! pip install imblearn
```

```
Requirement already satisfied: imblearn in /usr/local/lib/python3.6/dist-packages (0.0)
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.6/dist-packages (from imblearn) (0.4.3)
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.6/dist-packages (from imbalanced-learn->imblearn) (0.21.3)
Requirement already satisfied: scipy>=0.13.3 in /usr/local/lib/python3.6/dist-packages (from imbalanced-learn->imblearn) (1.3.3)
Requirement already satisfied: numpy>=1.8.2 in /usr/local/lib/python3.6/dist-packages (from imbalanced-learn->imblearn) (1.17.4)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (from scikit-learn>=0.20->imbalanced-learn->imbl
earn) (0.14.0)
```

Reading the data

In [0]:

```
data = pd.read_csv('train.csv')#train data
data_test = pd.read_csv('test.csv')#test data
```

In [0]:

```
data.head()
```

Out[0]:

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12	var_
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	-4.9200	5.7470	2.9252	3.1821	14.0137	0.57
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	3.1468	8.0851	-0.4032	8.0585	14.0239	8.41
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	-4.9193	5.9525	-0.3249	-11.2648	14.1929	7.31
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	-5.8609	8.2450	2.3061	2.8102	13.8463	11.97
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	6.2654	7.6784	-9.4458	-12.1419	13.8481	7.88

5 rows × 202 columns

In [0]:

```
data_test.head()
```

Out[0]:

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12	var_13	va
0	test_0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	2.1337	8.8100	-2.0248	-4.3554	13.9696	0.3458	7.5
1	test_1	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	-4.4131	5.9739	-1.3809	-0.3310	14.1129	2.5667	5.4
2	test_2	5.4827	-10.3581	10.1407	7.0479	10.2628	9.8052	4.8950	20.2537	1.5233	8.3442	-4.7057	-3.0422	13.6751	3.8183	10.8
3	test_3	8.5374	-1.3222	12.0220	6.5749	8.8458	3.1744	4.9397	20.5660	3.3755	7.4578	0.0095	-5.0659	14.0526	13.5010	8.7
4	test_4	11.7058	-0.1327	14.1295	7.7506	9.1035	-8.5848	6.8595	10.6048	2.9890	7.1437	5.1025	-3.2827	14.1013	8.9672	4.7

5 rows × 201 columns

In [0]:

```
data_1 = data.pop('target')
data['target'] = data_1
```

In [0]:

```
data.head()
```

Out[0]:

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12	var_13	var_
0	train_0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	-4.9200	5.7470	2.9252	3.1821	14.0137	0.5745	8.79
1	train_1	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	3.1468	8.0851	-0.4032	8.0585	14.0239	8.4135	5.43
2	train_2	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	-4.9193	5.9525	-0.3249	-11.2648	14.1929	7.3124	7.52
3	train_3	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	-5.8609	8.2450	2.3061	2.8102	13.8463	11.9704	6.45
4	train_4	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	6.2654	7.6784	-9.4458	-12.1419	13.8481	7.8895	7.78

5 rows × 202 columns

In [0]:

```
data.describe()
```

Out[0]:

	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	2000
mean	10.679914	-1.627622	10.715192	6.796529	11.078333	-5.065317	5.408949	16.545850	
std	3.040051	4.050044	2.640894	2.043319	1.623150	7.863267	0.866607	3.418076	
min	0.408400	-15.043400	2.117100	-0.040200	5.074800	-32.562600	2.347300	5.349700	
25%	8.453850	-4.740025	8.722475	5.254075	9.883175	-11.200350	4.767700	13.943800	
50%	10.524750	-1.608050	10.580000	6.825000	11.108250	-4.833150	5.385100	16.456800	
75%	12.758200	1.358625	12.516700	8.324100	12.261125	0.924800	6.003000	19.102900	
max	20.315000	10.376800	19.353000	13.188300	16.671400	17.251600	8.447700	27.691800	

8 rows × 201 columns

Exploratory data analysis

In [0]:

```
# data.to_csv('data_train.csv', index= False)
data_train = pd.read_csv('data_train.csv')
```

value count for both the target values

In [0]:

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

Out[0]:

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

Check for the null value count

In [0]:

```
data_train.isnull().sum()
```

Out[0]:

```
ID_code    0
var_0      0
var_1      0
var_2      0
var_3      0
          ..
var_196    0
var_197    0
var_198    0
var_199    0
target     0
Length: 202, dtype: int64
```

We are checking the data point distribution for both the target values 0 or 1

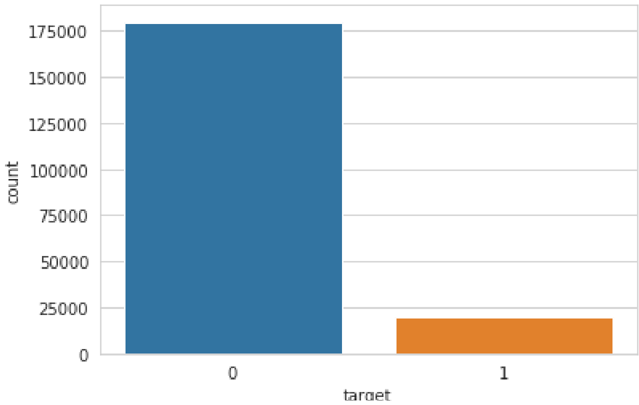
In [0]:

```
# First check for the distributio of 0 and 1
ax = sns.countplot('target',data=data_train)
print("percentage of data belongs to 0 :", data_train['target'].value_counts()[0]*100/200000,"%")
print("percentage of data belongs to 1 :", data_train['target'].value_counts()[1]*100/200000,"%")
ax.plot()
```

```
percentage of data belongs to 0 : 89.951 %
percentage of data belongs to 1 : 10.049 %
```

Out[0]:

```
[]
```



Observation

- The data is highly imbalanced by looking at the plot
- Around 90% of data belongs to the target class 0 and only 10% of data belongs target class 1

Checking th distriution of all the feature towards the target value

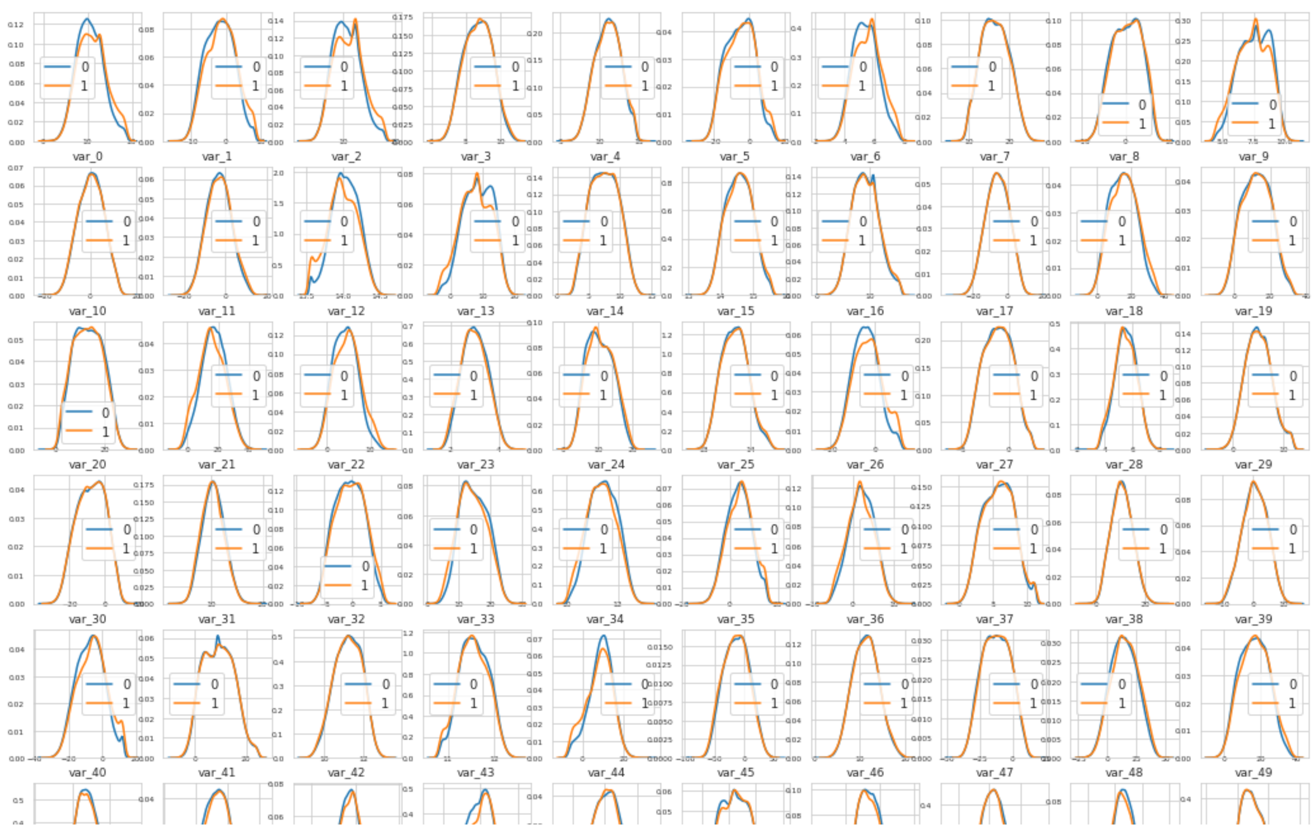
- Since we have 200 features and they all were anomyzed we have to check all the feature distribution towards the target value.
- For this we creat a function that gives us the distribution of each and every feature towards the target values.

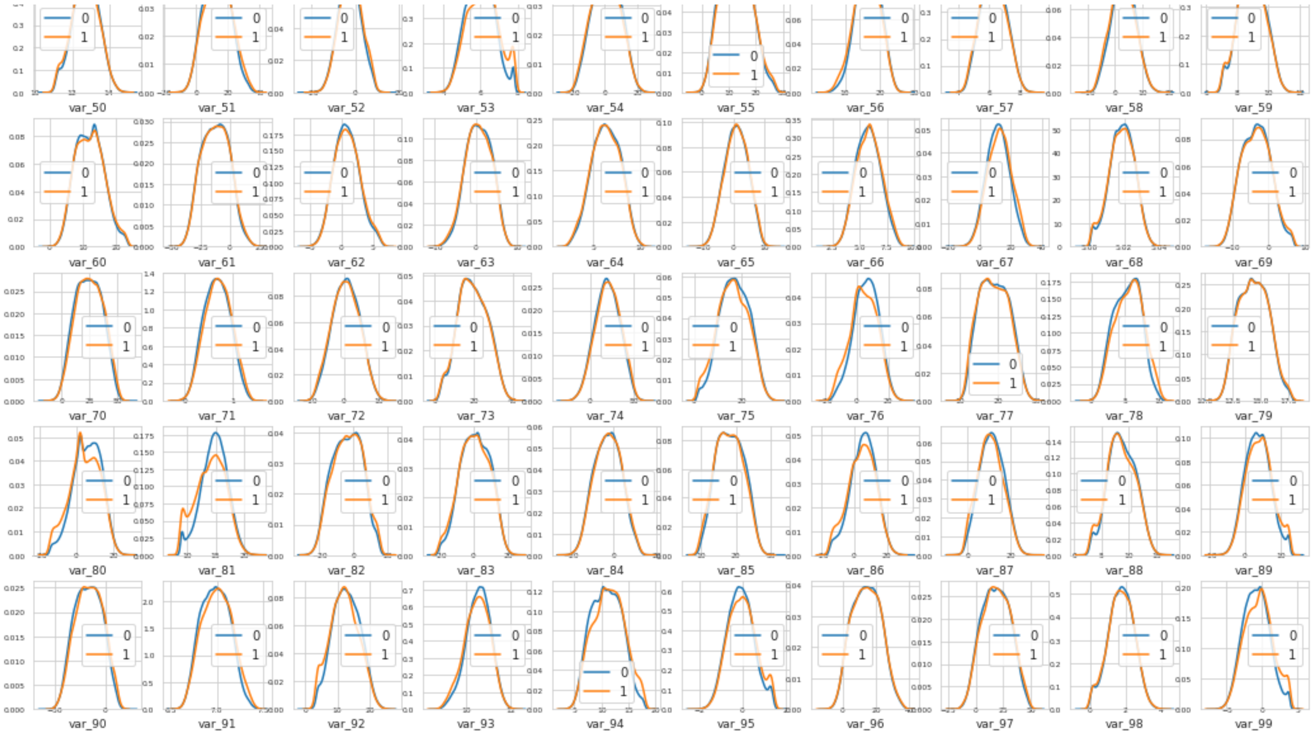
In [0]:

```
def feature_distribution(data_1,data_2,target_0,target_1,features_list):
    # Here we are setting the style of the plot and grid in it
    sns.set_style('whitegrid')
    plt.figure() # Here we are initializing the plt figure object
    # Here we are creating the subplot and initialzing it size and row col size
    fig, ax = plt.subplots(10,10,figsize=(18,22))
    for  plot_count, feature in enumerate(features_list):
        #plotting the plots here for every plot feature
        plt.subplot(10,10,plot_count+1)
        #plotting the pdf plot for every feature towards the target value
        sns.distplot(data_1[feature], hist=False,label=target_0)
        sns.distplot(data_2[feature], hist=False,label=target_1)
        plt.xlabel(feature, fontsize=9)# Here we are setting the x axis label
        locs, labels = plt.xticks()
        # Here we are setting the ticks for x and y axis
        plt.tick_params(axis='x', which='major', labelsize=6, pad=-6)
        plt.tick_params(axis='y', which='major', labelsize=6)
    plt.show();

## Dstribution for the first 100 features
target_0_data = data.loc[data_train['target'] == 0]
target_1_data = data.loc[data_train['target'] == 1]
features = data.columns.values[1:101]
feature_distribution(target_0_data, target_1_data, '0', '1', features)
```

<Figure size 432x288 with 0 Axes>

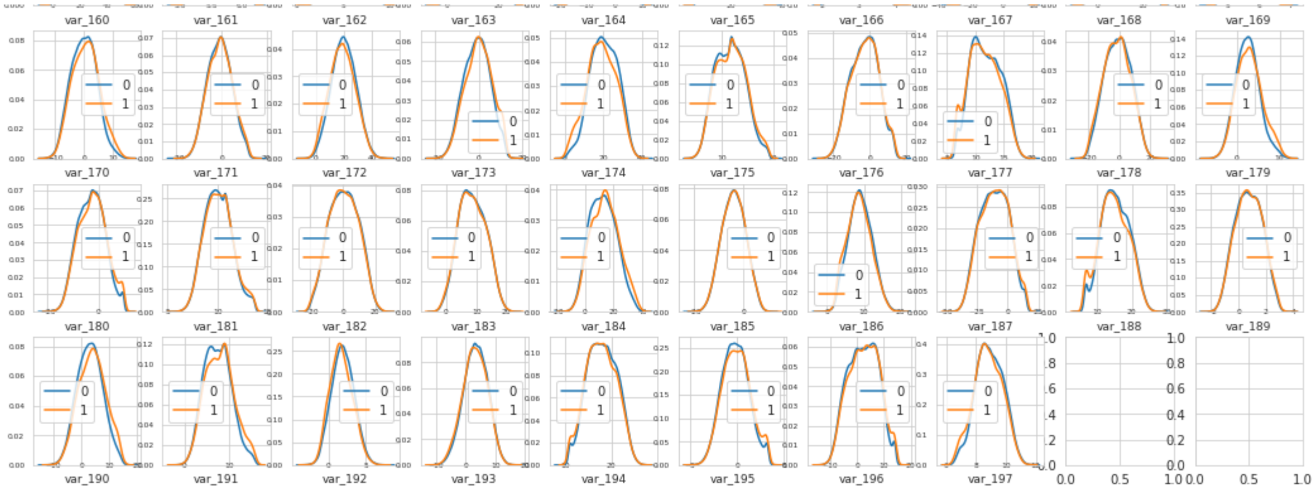




In [0]:

```
## Dstribution for the rest 100 features
features = data_train.columns.values[101:199]
feature_distribution(target_0_data, target_1_data, '0', '1', features)
```

<Figure size 432x288 with 0 Axes>



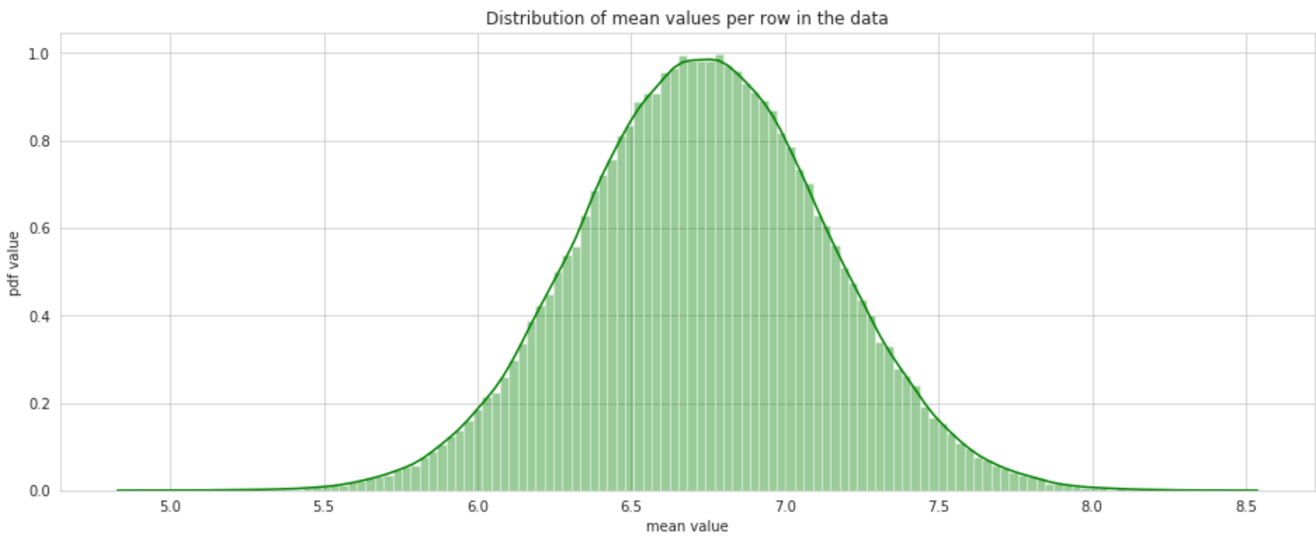
Observations

- By looking at the distribution of each feature towards the target values I found that most of the features have different distribution for the target values.
- We can also day that there are some features that are quite close to normal distribution not completely but a little
- Hence I can say that there are some kind o precessing is done on the data

Lets check the distribution of mean and std of the data

In [0]:

```
plt.figure(figsize=(16,6))
sns.set_style('whitegrid')
features = data_train.columns.values[1:202]
plt.title("Distribution of mean values per row in the data")
plt.xlabel('mean value')
plt.ylabel('pdf value')
sns.distplot(data_train[features].mean(axis=1),color="green", kde=True,bins=120)
plt.show()
```



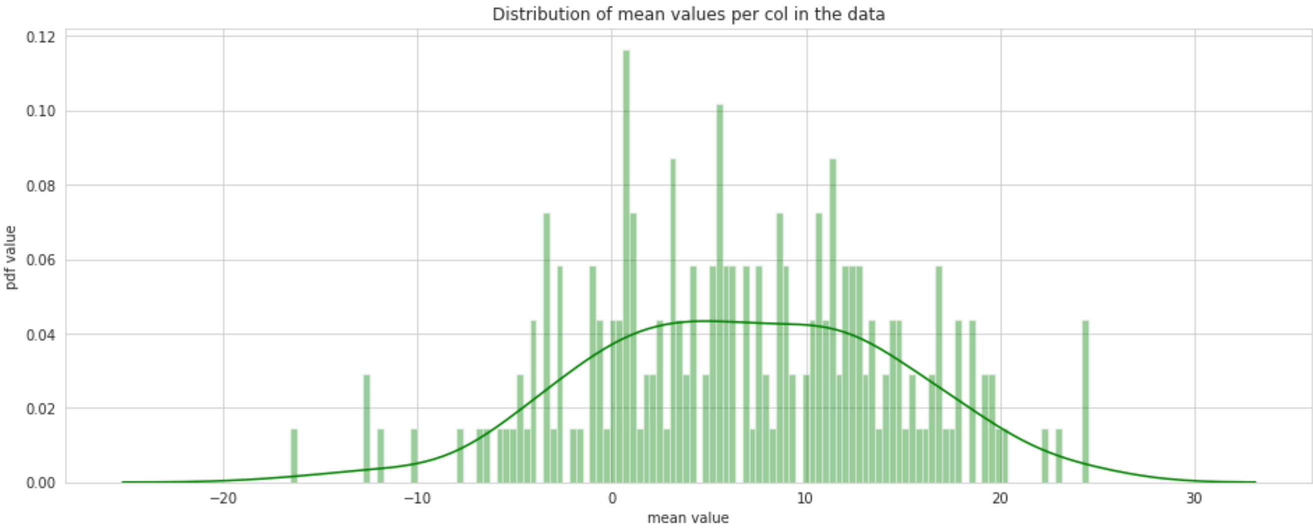
Observation

- The above graph shows the distribution of the means of each feature along the row and it seems to follow the kind of gausian.
- The graph looks kind of gausian with the mean value of 6.7342.
- From the above graph we can say that there are around 80% of feature whose mean lies between 6.5 and 7.0

In [0]:

```
plt.figure(figsize=(16,6))
```

```
plt.figure(figsize=(16,6))
sns.set_style('whitegrid')
features = data_train.columns.values[1:202]
plt.title("Distribution of mean values per col in the data")
plt.xlabel('mean value')
plt.ylabel('pdf value')
sns.distplot(data_train[features].mean(axis=0),color="green", kde=True,bins=120)
plt.show()
```

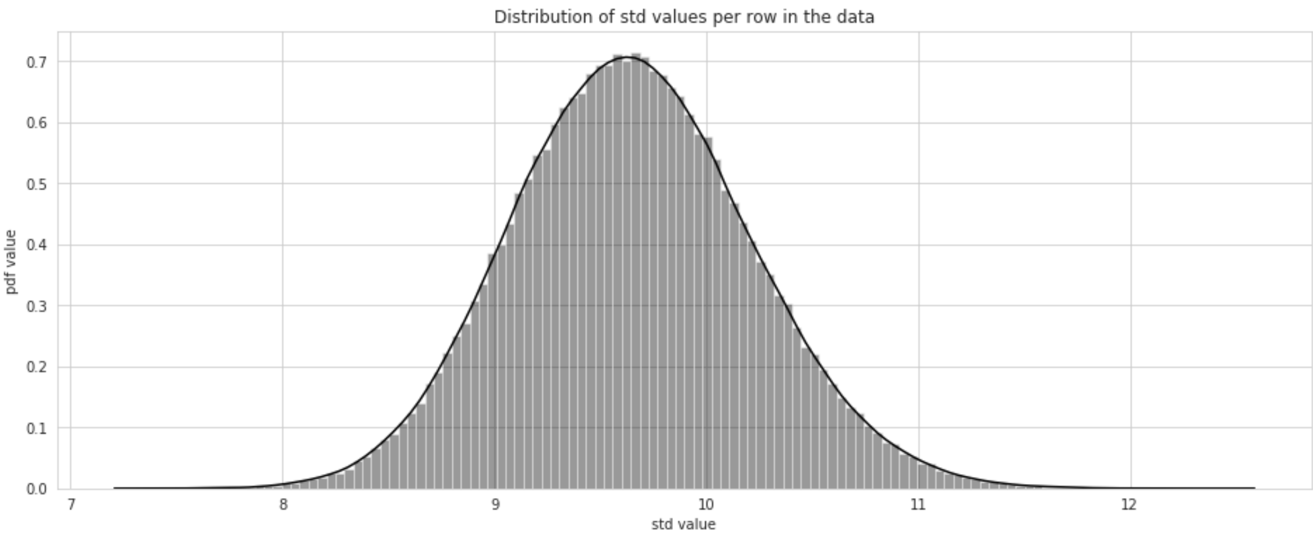


Observation

- The above graph is of the mean distriution of each feature columnwise.
- The columns wise mean distribution is not gaussian
- The mejority of columns having the mean value between -10 and 20

In [0]:

```
plt.figure(figsize=(16,6))
features = data_train.columns.values[1:202]
plt.title("Distribution of std values per row in the data")
plt.xlabel('std value')
plt.ylabel('pdf value')
sns.distplot(data_train[features].std(axis=1),color="black", kde=True,bins=120)
plt.show()
```



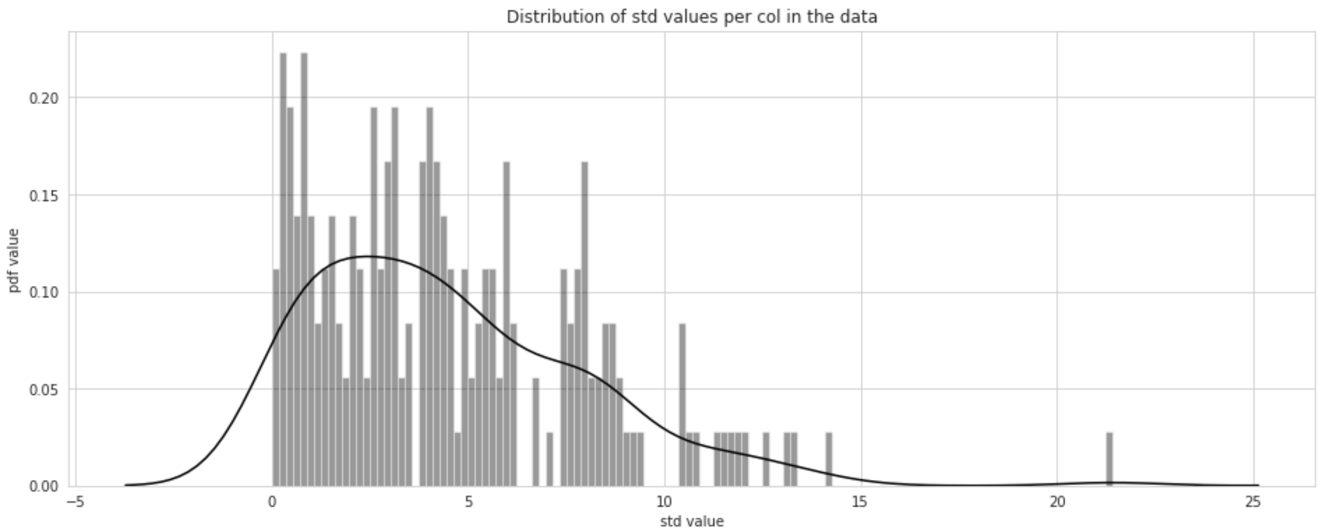
Observation

- we can see that the standard deviation distribution of each feature along the row also kind of follow the gussian distribution not exactly but as per the shape of the curve.

- Around 60% of features having the standard deviation around in the range of 9.3 - 10.

In [0]:

```
plt.figure(figsize=(16,6))
features = data_train.columns.values[1:202]
plt.title("Distribution of std values per col in the data")
plt.xlabel('std value')
plt.ylabel('pdf value')
sns.distplot(data_train[features].std(axis=0),color="black", kde=True,bins=120)
plt.show()
```



Observation

- As the graph we can say the distribution of the standard deviation along of features along the column is came from some other distribution.
- There are large number of feature having the deviation in the range of 0 and 6
- with minimum standard deviation of 0 and maximum of around 21 or something.

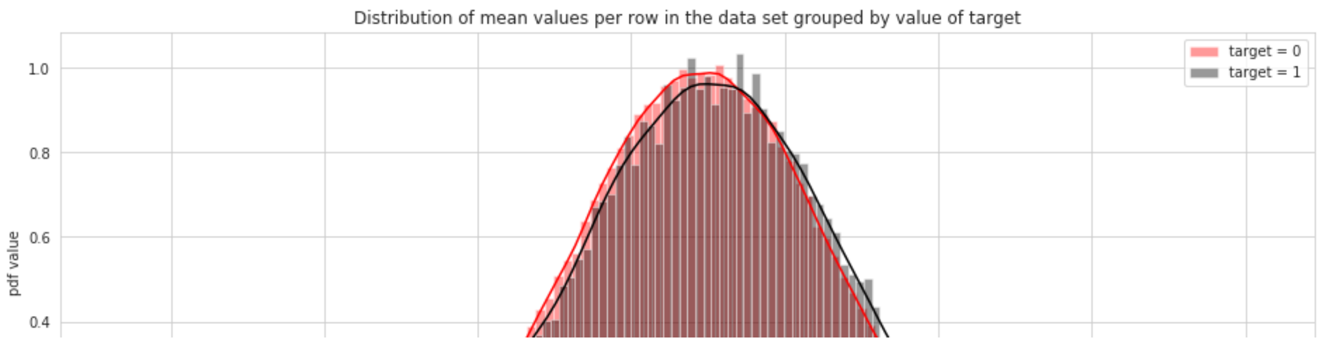
Distribution of the mean value dataset, grouped by value of target.

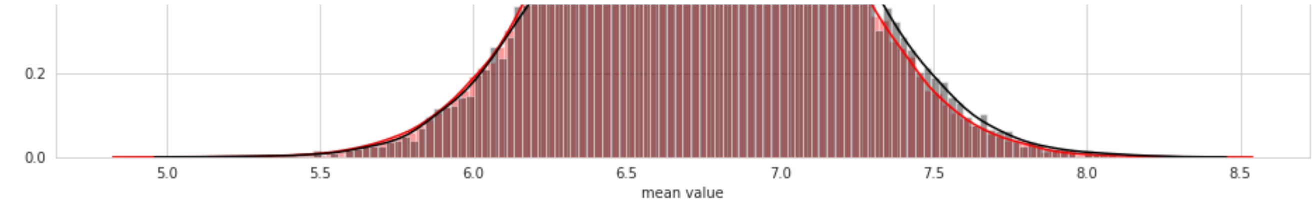
In [0]:

```
target_0_data = data_train.loc[data_train['target'] == 0]
target_1_data = data_train.loc[data_train['target'] == 1]
```

In [0]:

```
plt.figure(figsize=(16,6))
plt.title("Distribution of mean values per row in the data set grouped by value of target")
plt.xlabel('mean value')
plt.ylabel('pdf value')
sns.distplot(target_0_data[features].mean(axis=1),color="red", kde=True,bins=120, label='target = 0')
sns.distplot(target_1_data[features].mean(axis=1),color="black", kde=True,bins=120, label='target = 1')
plt.legend(); plt.show()
```



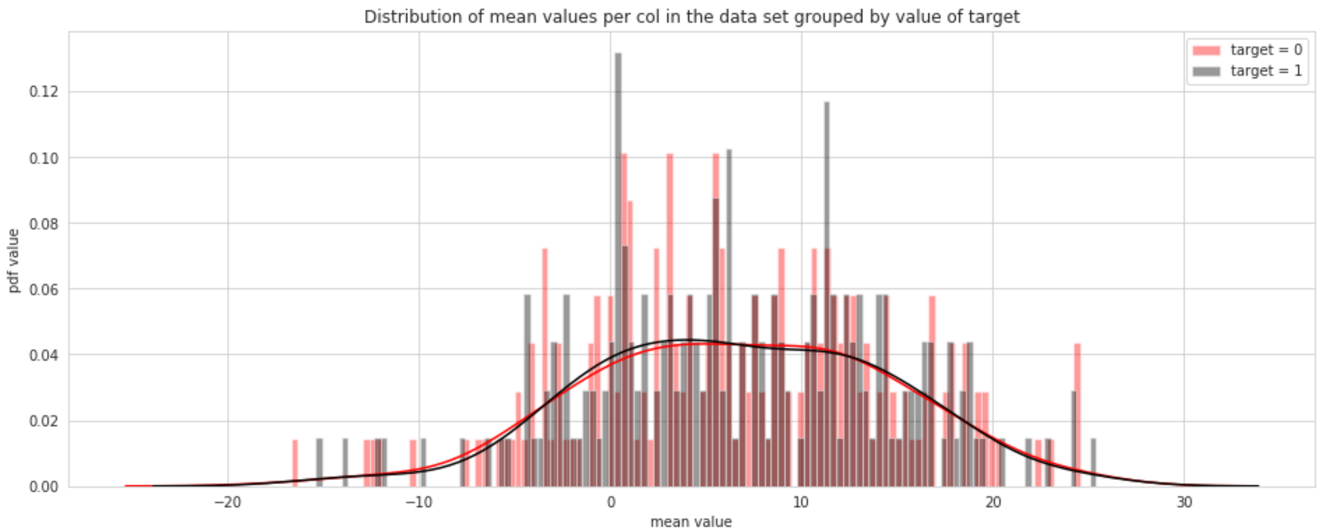


Observation

- The above graph show the mean distribution of every features towards the target
- The distribution of every feature towards each class lookes kind of similar.
- Hence the features will do welll on identifying the target class.

In [0]:

```
plt.figure(figsize=(16,6))
plt.title("Distribution of mean values per col in the data set grouped by value of target")
plt.xlabel('mean value')
plt.ylabel('pdf value')
sns.distplot(target_0_data[features].mean(axis=0),color="red", kde=True,bins=120, label='target = 0')
sns.distplot(target_1_data[features].mean(axis=0),color="black", kde=True,bins=120, label='target = 1')
plt.legend(); plt.show()
```



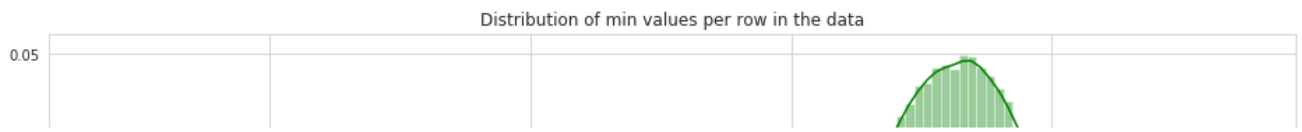
Observation

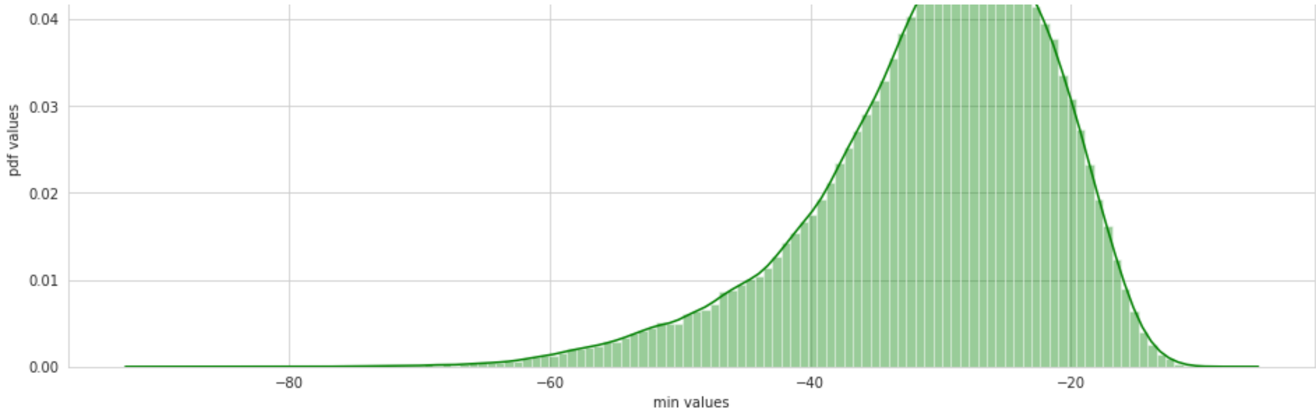
- Looking the above graph shows both the distirbution quite similary
- All feature will do we in identifying the target class
- The mejority of features means lies in the range of -10 to 20

Distribution of min and max value in data as row and col both wise

In [0]:

```
plt.figure(figsize=(16,6))
features = data_train.columns.values[1:202]
plt.title("Distribution of min values per row in the data")
plt.xlabel('min values')
plt.ylabel('pdf values')
sns.distplot(data_train[features].min(axis=1),color="green", kde=True,bins=120)
plt.show()
```



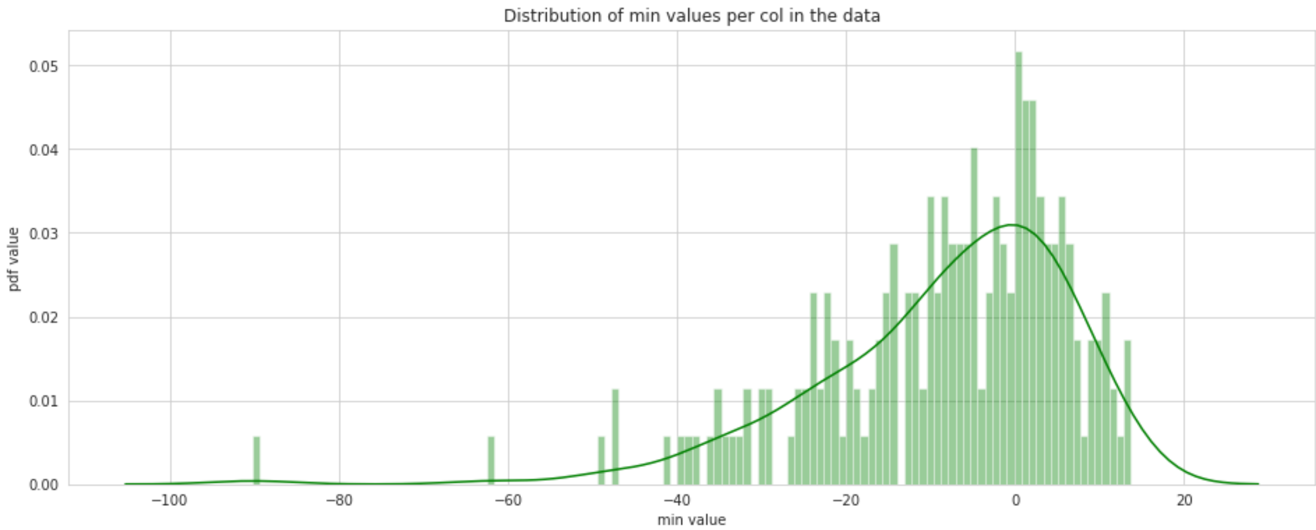


Observation

- The above graph shows the distribution of the min values of each features.
- The plot looks like skewed on the right side.
- Mejority of features having the min values in the range of -40 to -20.

In [0]:

```
plt.figure(figsize=(16,6))
features = data_train.columns.values[1:202]
plt.title("Distribution of min values per col in the data")
plt.xlabel('min value')
plt.ylabel('pdf value')
sns.distplot(data_train[features].min(axis=0),color="green", kde=True,bins=120)
plt.show()
```

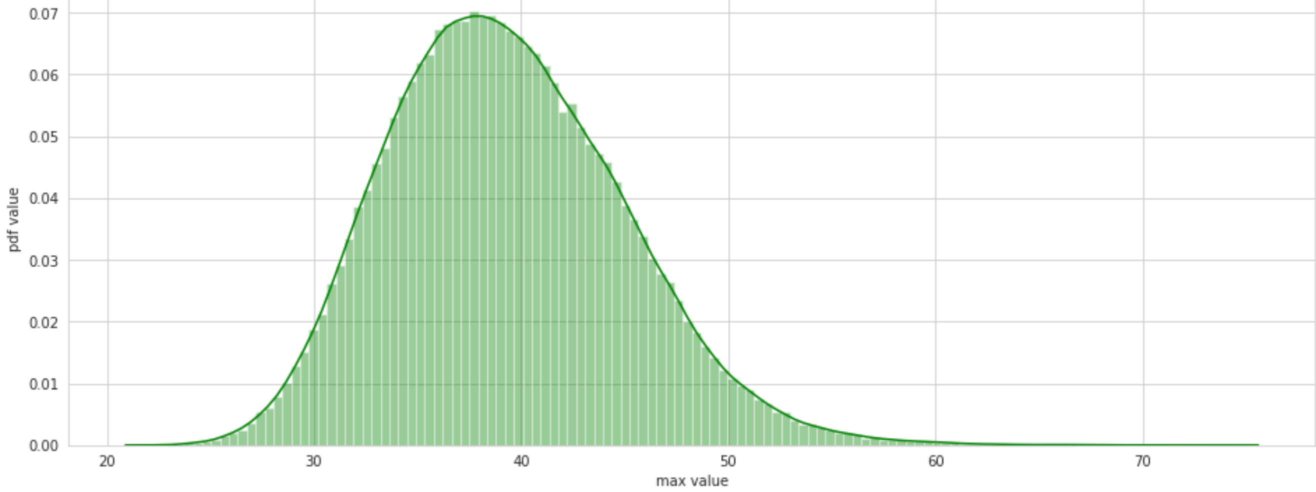


Observation

- The above graph is the column wise min value distribution of each feature.
- we have observed the lower value i.e -80 as the longer queue is at the lower side .

In [0]:

```
plt.figure(figsize=(16,6))
features = data_train.columns.values[1:202]
plt.title("Distribution of max values per row in the data")
plt.xlabel('max value')
plt.ylabel('pdf value')
sns.distplot(data_train[features].max(axis=1),color="green", kde=True,bins=120)
plt.show()
```

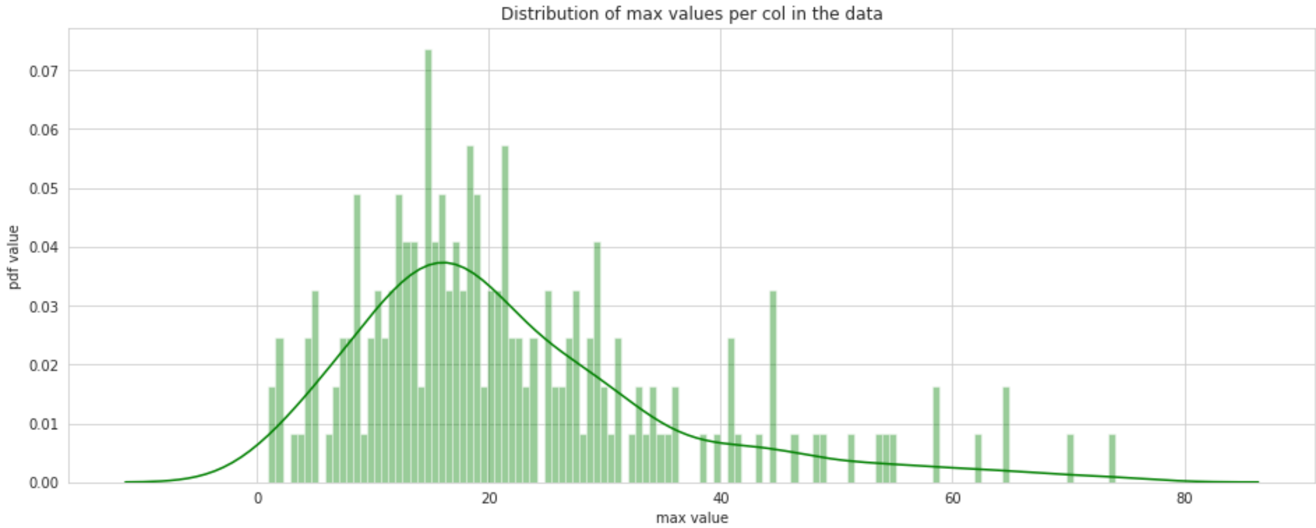


Observation

- The above distribution is the row wise distribution of the each feature max value.
- We can observe the max value of 70 on the right as the longer tail is on the right of the graph.
- The graph is skewed towards left as the longer tail is on the right side.

In [0]:

```
plt.figure(figsize=(16,6))
features = data_train.columns.values[1:202]
plt.title("Distribution of max values per col in the data")
plt.xlabel('max value')
plt.ylabel('pdf value')
sns.distplot(data_train[features].max(axis=0),color="green", kde=True,bins=120)
plt.show()
```



Observation

- The above distribution is the col wise distribution of the each feature max value.
- we can observe the max value of 80 as the longer tail of the graph is on the right side.

Now let's check the correation between the features

In [0]:

```
features = data_train.columns.values[1:200] # Here we are getting all th features
# And here we are calculating the correlation of every featre ad sorting it in ascending order
cor_data = data_train[features].corr().abs().unstack().sort_values(kind="quicksort").reset_index()
cor_data = cor_data[cor_data['level_0'] != cor_data['level_1']]
```

```
cor_data = cor_data[cor_data['level_0'] != cor_data['level_1']]
# cor_data.head(10)
```

some most correlated data

In [0]:

```
cor_data.tail(10)
```

Out[0]:

	level_0	level_1	0
39392	var_183	var_189	0.009359
39393	var_189	var_183	0.009359
39394	var_81	var_174	0.009490
39395	var_174	var_81	0.009490
39396	var_165	var_81	0.009714
39397	var_81	var_165	0.009714
39398	var_148	var_53	0.009788
39399	var_53	var_148	0.009788
39400	var_139	var_26	0.009844
39401	var_26	var_139	0.009844

some least correlated data

In [0]:

```
cor_data.head(10)
```

Out[0]:

	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_109	var_126	1.313947e-07
5	var_126	var_109	1.313947e-07
6	var_144	var_27	1.772502e-07
7	var_27	var_144	1.772502e-07
8	var_100	var_177	3.116544e-07
9	var_177	var_100	3.116544e-07

Training a Baseline model

In [0]:

```
data_train.head(5)
```

Out[0]:

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12	var_13	var_
0	train_0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	-4.9200	5.7470	2.9252	3.1821	14.0137	0.5745	8.7
1	train_1	11.5006	[illegible]	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	3.1468	8.0851	[illegible]	8.0585	14.0239	8.4135	5.4

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12	var_13	var_
2	train_2	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	-4.9193	5.9525	-0.3249	-11.2648	14.1929	7.3124	7.52
3	train_3	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	-5.8609	8.2450	2.3061	2.8102	13.8463	11.9704	6.45
4	train_4	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	6.2654	7.6784	-9.4458	-12.1419	13.8481	7.8895	7.78

5 rows × 202 columns

In [0]:

```
# creating the training and target data
y = data_train['target']
x = data_train.drop(['ID_code','target'],axis=1)
```

In [0]:

```
# Splitting it into train and cv
X_train, X_cv, y_train, y_cv = train_test_split(x,y ,test_size=0.33,random_state=42)
```

In [0]:

```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_cv.shape[0])
```

```
Number of data points in train data: 134000
Number of data points in test data: 66000
```

In [0]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

Hyperparameter tunning

In [0]:

```
parameters = {'num_leaves':[6,8,12,16],'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
         'n_estimators':[100,200,500,1000,2000],'max_depth':[3,5,10]} # setting the parameter to be tune
lgb_clf = lgb.LGBMClassifier()# Initializing the Lightgbm model
# tunning the parameter using the randomsearch
lgb_clf_rs = RandomizedSearchCV(lgb_clf, parameters, verbose=10, n_jobs=-1)
lgb_clf_rs.fit(X_train,y_train)# Training the model
print(lgb_clf_rs.best_params_) # Printing the best parameter
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   29.1s
/usr/local/lib/python3.6/dist-packages/joblib/externals/loky/process_executor.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:  3.6min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  5.3min
```

```
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  8.0min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 11.0min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 16.1min finished
```

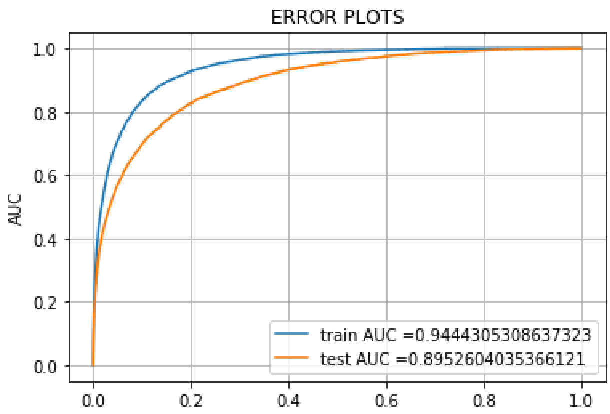
```
{'num_leaves': 8, 'n_estimators': 2000, 'max_depth': 5, 'learning_rate': 0.05}
```

In [0]:

```
start = timeit.default_timer()
# %%time
lgb_model = lgb.LGBMClassifier(boosting_type= 'gbdt', objective='binary',feature_fraction=0.05,
                               class_weight='balanced',num_leaves=8,n_estimators=2000,max_depth=5,
                               learning_rate=0.05,metric='auc',bagging_fraction=0.4, n_jobs=-1)
lgb_model.fit(X_train, y_train)

predict_y_train = batch_predict(lgb_model, X_train)
predict_y_cv = batch_predict(lgb_model, X_cv)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, predict_y_train)
test_fpr, test_tpr, te_thresholds = roc_curve(y_cv, predict_y_cv)
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()
stop = timeit.default_timer()
print('Time in mins: ',(stop - start)/60)
```




```
Time in mins:  0.8437324482166635
```

Observation

we can see that the baseline model which is trained on the raw features is giving a good auc result of 89.52%, Hence we can use it as our base model and try to improve it further with some feature engineering and get it 90% or above.

some simple feature engineering

simple features like sum, min, max, mean, std, skew, kurt and median as below row wise.

In [0]:

```
%%time
# From the EDA we can see that we can use the sum,min,max,mean,sd,skew,kurt,med as the features for both
# train and test
# https://www.youtube.com/watch?v=LEWpRlaEJO8
idx = features = data.columns.values[2:202]
for dataFrame in [data_test, data_train]:
  dataFrame['sum'] = dataFrame[idx].sum(axis=1)
  dataFrame['min'] = dataFrame[idx].min(axis=1)
```

```
    dataFrame['max'] = dataFrame[idx].max(axis=1)
    dataFrame['mean'] = dataFrame[idx].mean(axis=1)
    dataFrame['std'] = dataFrame[idx].std(axis=1)
    dataFrame['skew'] = dataFrame[idx].skew(axis=1)
    dataFrame['kurt'] = dataFrame[idx].kurtosis(axis=1)
    dataFrame['med'] = dataFrame[idx].median(axis=1)
```

CPU times: user 10.9 s, sys: 739 ms, total: 11.6 s
Wall time: 11.6 s

In [0]:

```
data_train.head(5)
```

Out[0]:

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12	var_13	var_
0	train_0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	-4.9200	5.7470	2.9252	3.1821	14.0137	0.5745	8.79
1	train_1	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	3.1468	8.0851	-0.4032	8.0585	14.0239	8.4135	5.43
2	train_2	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	-4.9193	5.9525	-0.3249	-11.2648	14.1929	7.3124	7.52
3	train_3	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	-5.8609	8.2450	2.3061	2.8102	13.8463	11.9704	6.45
4	train_4	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	6.2654	7.6784	-9.4458	-12.1419	13.8481	7.8895	7.78

5 rows × 210 columns

In [0]:

```
data_test.head(5)
```

Out[0]:

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12	var_13	va
0	test_0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	2.1337	8.8100	-2.0248	-4.3554	13.9696	0.3458	7.5
1	test_1	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	-4.4131	5.9739	-1.3809	-0.3310	14.1129	2.5667	5.4
2	test_2	5.4827	-10.3581	10.1407	7.0479	10.2628	9.8052	4.8950	20.2537	1.5233	8.3442	-4.7057	-3.0422	13.6751	3.8183	10.8
3	test_3	8.5374	-1.3222	12.0220	6.5749	8.8458	3.1744	4.9397	20.5660	3.3755	7.4578	0.0095	-5.0659	14.0526	13.5010	8.7
4	test_4	11.7058	-0.1327	14.1295	7.7506	9.1035	-8.5848	6.8595	10.6048	2.9890	7.1437	5.1025	-3.2827	14.1013	8.9672	4.7

5 rows × 209 columns

Let's take the round off feature's as well

In [0]:

```
%%time
# https://www.geeksforgeeks.org/numpy-round_-python/
features_value = [col for col in data_train.columns if col not in ['ID_code','target']]
# In this we ar rounding of the value of each columns and creating a new freature of the same
for feature in features_value:
    data_train['round_2'+ feature] = np.round(data_train[feature],2)
    data_test['round_2'+ feature] = np.round(data_test[feature],2)
    data_train['round_1'+ feature] = np.round(data_train[feature],1)
    data_test['round_1'+ feature] = np.round(data_test[feature],1)
```

CPU times: user 4.61 s, sys: 911 ms, total: 5.52 s
Wall time: 5.52 s

In [0]:

```
df = data_train.pop('target')
data_train['target'] = df
data_train.head(5)
```

Out[0]:

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12	var_13	var_
0	train_0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	-4.9200	5.7470	2.9252	3.1821	14.0137	0.5745	8.79
1	train_1	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	3.1468	8.0851	-0.4032	8.0585	14.0239	8.4135	5.43
2	train_2	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	-4.9193	5.9525	-0.3249	-11.2648	14.1929	7.3124	7.52
3	train_3	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	-5.8609	8.2450	2.3061	2.8102	13.8463	11.9704	6.45
4	train_4	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	6.2654	7.6784	-9.4458	-12.1419	13.8481	7.8895	7.78

5 rows × 626 columns

In [0]:

```
data_test.head(5)
```

Out[0]:

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12	var_13	va
0	test_0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	2.1337	8.8100	-2.0248	-4.3554	13.9696	0.3458	7.5
1	test_1	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	-4.4131	5.9739	-1.3809	-0.3310	14.1129	2.5667	5.4
2	test_2	5.4827	-10.3581	10.1407	7.0479	10.2628	9.8052	4.8950	20.2537	1.5233	8.3442	-4.7057	-3.0422	13.6751	3.8183	10.8
3	test_3	8.5374	-1.3222	12.0220	6.5749	8.8458	3.1744	4.9397	20.5660	3.3755	7.4578	0.0095	-5.0659	14.0526	13.5010	8.7
4	test_4	11.7058	-0.1327	14.1295	7.7506	9.1035	-8.5848	6.8595	10.6048	2.9890	7.1437	5.1025	-3.2827	14.1013	8.9672	4.7

5 rows × 625 columns

In [0]:

```
print("total number of features after feature engineering",len(data_train.columns))
```

total number of features after feature engineering 626

In [0]:

```
print("total number of features after feature engineering",len(data_test.columns))
```

total number of features after feature engineering 625

In [0]:

```
# data_train.to_csv('data_train_final.csv', index = False)
data_train_final_ = pd.read_csv('data_train_final.csv')
```

In [0]:

```
# data_test.to_csv('data_test_final.csv', index = False)
data_test_final_ = pd.read_csv('data_test_final.csv')
```

In [0]:

```
df_test = data_test_final_.pop('ID_code')
```

In [100]:

```
data_test_final_
```

Out[100]:

	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12	var_13	var_1
0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	2.1337	8.8100	-2.0248	-4.3554	13.9696	0.3458	7.540
1	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	-4.4131	5.9739	-1.3809	-0.3310	14.1129	2.5667	5.498
2	5.4827	-10.3581	10.1407	7.0479	10.2628	9.8052	4.8950	20.2537	1.5233	8.3442	-4.7057	-3.0422	13.6751	3.8183	10.853
3	8.5374	-1.3222	12.0220	6.5749	8.8458	3.1744	4.9397	20.5660	3.3755	7.4578	0.0095	-5.0659	14.0526	13.5010	8.766
4	11.7058	-0.1327	14.1295	7.7506	9.1035	-8.5848	6.8595	10.6048	2.9890	7.1437	5.1025	-3.2827	14.1013	8.9672	4.727
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
199995	13.1678	1.0136	10.4333	6.7997	8.5974	-4.1641	4.8579	14.7625	-2.7239	6.9937	2.6802	6.1565	14.3201	17.4594	5.371
199996	9.7171	-9.1462	7.3443	9.1421	12.8936	3.0191	5.6888	18.8862	5.0915	6.3545	3.2618	-2.0445	13.8246	6.6547	5.030
199997	11.6360	2.2769	11.2074	7.7649	12.6796	11.3224	5.3883	18.3794	1.6603	5.7341	9.8596	-0.3412	14.0675	13.9975	6.257
199998	13.5745	-0.5134	13.6584	7.4855	11.2241	-11.3037	4.1959	16.8280	5.3208	8.9032	5.5000	-13.1346	14.3051	4.2644	11.125
199999	10.4664	1.8070	10.2277	6.0654	10.0258	1.0789	4.8879	14.4892	-0.5902	7.8362	8.4796	-5.8960	13.8333	2.4590	7.888

200000 rows × 624 columns

Let's train lightgbm model on this dataset

In [0]:

```
y = data_train_final_['target']
x = data_train_final_.drop(['ID_code','target'],axis=1)
```

In [0]:

```
parameter = {
    '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 [108]:

```
# https://www.kaggle.com/adrianlievano/light-gbm-with-stratified-kfold
# Getting all the features name except the ID_code , target
features = [col for col in data_train_final_.columns if col not in ['ID_code', 'target']]
# Inititalizing the K-Fold object
K_folds = StratifiedKFold(n_splits=10, shuffle=False, random_state=44000)
# this creates the empty numpy array of length of x in which we store the prediction of every validation data
val_pred = np.zeros(len(x))
# In this we keep the predicted output of the test data
predictions_test = np.zeros(len(data_test_final_))
#In this loop we are doing the training and prediction for each folds and we are getting the train and valid data
# using the trn_idx and val_idx
for n_fold, (trn_idx, val_idx) in enumerate(K_folds.split(x.values, y.values)):
    print("Fold {}".format(n_fold))
    # Getting the train and validation data from the x data
    train_data = lgb.Dataset(x.iloc[trn_idx][features], label=y.iloc[trn_idx])
    valid_data = lgb.Dataset(x.iloc[val_idx][features], label=y.iloc[val_idx])
    # Here we are training lightgbm model on train and valid dataset
    num_round = 1000000
    classifier = lgb.train(parameter, train_data, num_round,
                           valid_sets = [train_data, valid_data],
                           verbose_eval=1000, early_stopping_rounds = 3000)
    # Here we are doing the prediction on the valid data
    val_pred[val_idx] = classifier.predict(x.iloc[val_idx][features], num_iteration=classifier.best_iteration)
    # And here we are doing the prediction on the test data
    predictions_test += classifier.predict(data_test_final_[features],
                                           num_iteration=classifier.best_iteration) / K_folds.n_splits
print("CV score: {:<8.5f}".format(roc_auc_score(y, val_pred)))
```

```
Fold 0
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.884606 valid_1's auc: 0.864238
[2000] training's auc: 0.909543 valid_1's auc: 0.884301
[3000] training's auc: 0.921172 valid_1's auc: 0.892468
[4000] training's auc: 0.928569 valid_1's auc: 0.896445
[5000] training's auc: 0.934233 valid_1's auc: 0.898244
[6000] training's auc: 0.93909 valid_1's auc: 0.89918
[7000] training's auc: 0.943654 valid_1's auc: 0.89976
[8000] training's auc: 0.947872 valid_1's auc: 0.89997
[9000] training's auc: 0.951984 valid_1's auc: 0.900189
[10000] training's auc: 0.955868 valid_1's auc: 0.900104
[11000] training's auc: 0.959515 valid_1's auc: 0.900119
Early stopping, best iteration is:
[8864] training's auc: 0.951432 valid_1's auc: 0.900263
Fold 1
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.885043 valid_1's auc: 0.865176
[2000] training's auc: 0.909497 valid_1's auc: 0.884609
[3000] training's auc: 0.921101 valid_1's auc: 0.892007
[4000] training's auc: 0.928448 valid_1's auc: 0.895779
[5000] training's auc: 0.934179 valid_1's auc: 0.897228
[6000] training's auc: 0.939098 valid_1's auc: 0.898258
[7000] training's auc: 0.943683 valid_1's auc: 0.898647
[8000] training's auc: 0.947879 valid_1's auc: 0.898747
[9000] training's auc: 0.951974 valid_1's auc: 0.89876
[10000] training's auc: 0.955782 valid_1's auc: 0.898835
[11000] training's auc: 0.959523 valid_1's auc: 0.898605
Early stopping, best iteration is:
[8706] training's auc: 0.950803 valid_1's auc: 0.898912
Fold 2
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.885983 valid_1's auc: 0.861552
[2000] training's auc: 0.910108 valid_1's auc: 0.880106
[3000] training's auc: 0.921748 valid_1's auc: 0.8875
[4000] training's auc: 0.929068 valid_1's auc: 0.891016
[5000] training's auc: 0.934709 valid_1's auc: 0.892928
[6000] training's auc: 0.939557 valid_1's auc: 0.89385
[7000] training's auc: 0.944015 valid_1's auc: 0.894434
[8000] training's auc: 0.948275 valid_1's auc: 0.89449
[9000] training's auc: 0.952324 valid_1's auc: 0.89449
[10000] training's auc: 0.956192 valid_1's auc: 0.894362
Early stopping, best iteration is:
[7436] training's auc: 0.945858 valid_1's auc: 0.894632
Fold 3
Training until validation scores don't improve for 3000 rounds.
```

```
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.885443 valid_1's auc: 0.867247
[2000] training's auc: 0.90954 valid_1's auc: 0.886153
[3000] training's auc: 0.921309 valid_1's auc: 0.892953
[4000] training's auc: 0.928761 valid_1's auc: 0.895989
[5000] training's auc: 0.934347 valid_1's auc: 0.896856
[6000] training's auc: 0.939225 valid_1's auc: 0.89763
[7000] training's auc: 0.943746 valid_1's auc: 0.89756
[8000] training's auc: 0.947931 valid_1's auc: 0.897708
[9000] training's auc: 0.951995 valid_1's auc: 0.897849
[10000] training's auc: 0.955848 valid_1's auc: 0.897771
[11000] training's auc: 0.959463 valid_1's auc: 0.897726
[12000] training's auc: 0.962911 valid_1's auc: 0.897596
Early stopping, best iteration is:
[9010] training's auc: 0.952037 valid_1's auc: 0.897867
Fold 4
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.884807 valid_1's auc: 0.866529
[2000] training's auc: 0.909424 valid_1's auc: 0.886117
[3000] training's auc: 0.921107 valid_1's auc: 0.893545
[4000] training's auc: 0.928536 valid_1's auc: 0.896759
[5000] training's auc: 0.934213 valid_1's auc: 0.897806
[6000] training's auc: 0.939185 valid_1's auc: 0.898462
[7000] training's auc: 0.943667 valid_1's auc: 0.898662
[8000] training's auc: 0.947877 valid_1's auc: 0.898594
[9000] training's auc: 0.951866 valid_1's auc: 0.898649
[10000] training's auc: 0.955672 valid_1's auc: 0.898559
Early stopping, best iteration is:
[7290] training's auc: 0.944869 valid_1's auc: 0.89883
Fold 5
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.884561 valid_1's auc: 0.870618
[2000] training's auc: 0.909006 valid_1's auc: 0.890413
[3000] training's auc: 0.920664 valid_1's auc: 0.89779
[4000] training's auc: 0.928177 valid_1's auc: 0.900929
[5000] training's auc: 0.93387 valid_1's auc: 0.902325
[6000] training's auc: 0.938875 valid_1's auc: 0.903087
[7000] training's auc: 0.943425 valid_1's auc: 0.90311
[8000] training's auc: 0.947708 valid_1's auc: 0.903108
[9000] training's auc: 0.951768 valid_1's auc: 0.903242
[10000] training's auc: 0.955608 valid_1's auc: 0.903108
[11000] training's auc: 0.959266 valid_1's auc: 0.902936
Early stopping, best iteration is:
[8748] training's auc: 0.950784 valid_1's auc: 0.903312
Fold 6
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.884649 valid_1's auc: 0.869436
[2000] training's auc: 0.909382 valid_1's auc: 0.888267
[3000] training's auc: 0.92114 valid_1's auc: 0.894835
[4000] training's auc: 0.928623 valid_1's auc: 0.898115
[5000] training's auc: 0.934278 valid_1's auc: 0.899504
[6000] training's auc: 0.93928 valid_1's auc: 0.900205
[7000] training's auc: 0.943733 valid_1's auc: 0.900487
[8000] training's auc: 0.948019 valid_1's auc: 0.900548
[9000] training's auc: 0.952087 valid_1's auc: 0.90049
[10000] training's auc: 0.955889 valid_1's auc: 0.900483
Early stopping, best iteration is:
[7756] training's auc: 0.947008 valid_1's auc: 0.900679
Fold 7
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.884911 valid_1's auc: 0.866136
[2000] training's auc: 0.909461 valid_1's auc: 0.885681
[3000] training's auc: 0.921156 valid_1's auc: 0.892834
[4000] training's auc: 0.928467 valid_1's auc: 0.896455
[5000] training's auc: 0.934128 valid_1's auc: 0.898287
[6000] training's auc: 0.939183 valid_1's auc: 0.899155
[7000] training's auc: 0.943664 valid_1's auc: 0.899398
[8000] training's auc: 0.94792 valid_1's auc: 0.899446
[9000] training's auc: 0.952002 valid_1's auc: 0.899523
[10000] training's auc: 0.955831 valid_1's auc: 0.899552
[11000] training's auc: 0.95948 valid_1's auc: 0.899317
[12000] training's auc: 0.962967 valid_1's auc: 0.899352
Early stopping, best iteration is:
[9645] training's auc: 0.954545 valid_1's auc: 0.899626
Fold 8
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.88476 valid_1's auc: 0.87396
[2000] training's auc: 0.908852 valid_1's auc: 0.892868
```

```
[3000] training's auc: 0.920811 valid_1's auc: 0.900231
[4000] training's auc: 0.928157 valid_1's auc: 0.902959
[5000] training's auc: 0.933848 valid_1's auc: 0.9041
[6000] training's auc: 0.938791 valid_1's auc: 0.904795
[7000] training's auc: 0.943392 valid_1's auc: 0.904765
[8000] training's auc: 0.947665 valid_1's auc: 0.904797
[9000] training's auc: 0.951701 valid_1's auc: 0.904872
[10000] training's auc: 0.955555 valid_1's auc: 0.90476
[11000] training's auc: 0.959255 valid_1's auc: 0.904859
Early stopping, best iteration is:
[8451] training's auc: 0.949493 valid_1's auc: 0.904973
Fold 9
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.884948 valid_1's auc: 0.869464
[2000] training's auc: 0.909302 valid_1's auc: 0.887962
[3000] training's auc: 0.921136 valid_1's auc: 0.89503
[4000] training's auc: 0.928514 valid_1's auc: 0.898296
[5000] training's auc: 0.934177 valid_1's auc: 0.899851
[6000] training's auc: 0.939089 valid_1's auc: 0.90058
[7000] training's auc: 0.943661 valid_1's auc: 0.901047
[8000] training's auc: 0.947907 valid_1's auc: 0.901073
[9000] training's auc: 0.951996 valid_1's auc: 0.901121
[10000] training's auc: 0.95587 valid_1's auc: 0.901159
[11000] training's auc: 0.959491 valid_1's auc: 0.901077
[12000] training's auc: 0.962948 valid_1's auc: 0.900954
[13000] training's auc: 0.966231 valid_1's auc: 0.900753
Early stopping, best iteration is:
[10394] training's auc: 0.957302 valid_1's auc: 0.901253
CV score: 0.90000
```

Creating the submission dataframe

In [0]:

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

In [0]:

```
sub_data = pd.read_csv("submission.csv")
```

In [0]:

```
sub_data
```

Out[0]:

	ID_code	target
0	test_0	0.088934
1	test_1	0.227910
2	test_2	0.170440
3	test_3	0.204776
4	test_4	0.042111
...	...	...
199995	test_199995	0.036870
199996	test_199996	0.007906
199997	test_199997	0.004250
199998	test_199998	0.091177
199999	test_199999	0.063739

200000 rows × 2 columns