Scania truck APS failure prediction

Introduction

Heavy duty vehicle is essential part of our transportation system. In heavy duty vehicle we use Air Pressure System in various parts like break and gear system. Basically we use the APS to control the vehicle, for that it become very important to do regular maintenance otherwise it can lead the accident and high cost maintenance. To deal with this problem Scania truck wants to minimize the cost of maintenance using Machine Learning algorithm. Dataset consist of data collected from heavy Scania Truck in everyday usage. Our task is to predict whether a given failure is occurred due to specific component of the APS or not. This may help in avoiding failure during the operation and thereby reducing maintenance cost.

ML Formulation

This is a binary classification problem. There are two classes positive and negative. Positive class tells us that failure occurred due to the ASP and negative class tells us that failure did not occurred due to ASP. We have to build a ML model which can take data from the various sensors and predict that failure was happened due to APS or not. It will going save the time and reduce the cost of maintenance

Business Constrain

Latency must be low and model should be able to predict the failure in ASP as soon as quickly. Cost of misclassification is very high specially False negative and lead the high cost of maintenance

Data overview

Data consists of two sets of file

i. Train.csv

ii. Test.csv

Training set which have 60000 data points and 171 features out of which 59000 belong to the negative class and 1000 belong to the positive class. Test set contain 16000 data points out of which 15625 belongs to the negative class and 375 data point belong to positive class. The attribute name of the data has been anonymized for proprietary reason. It consists of both single numerical counters and histogram consisting of bin with different conditions. Data is highly imbalance and there are lots of missing values. In dataset have total 171 features out of this feature, 70 features are histogram features

Performance Metric

In this problem we can use Macro-F1 score as our performance metric to calculate the cost_1 and cost_2. Basically in Macro F1 score we calculate the F1 score of each class separately and compute the average of it. Macro-F1 score best value is 1 and worst value is 0. With the help of cost_1 and cost_2 we can calculate the Total cost of maintenance.

Dataset Link

https://archive.ics.uci.edu/ml/datasets/APS+Failure+at+Scania+Trucks

https://www.kaggle.com/c/scania-truck-failures

Exploratory Data Analysis and Data Preprocessing

In [ ]:

```
# Mounting google drive on notebook

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

```
Mounted at /content/drive
```

Important Libraries

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#from fancyimpute import SoftImpute
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from collections import Counter
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import RFE
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.experimental import enable_iterative_imputer
from imblearn.over_sampling import SMOTE
from sklearn.impute import IterativeImputer
from sklearn.linear_model import Ridge
import seaborn as sns
import joblib
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve,auc
from sklearn.metrics import confusion_matrix
from sklearn.naive_bayes import GaussianNB
from scipy.stats import uniform,randint
from tqdm import tqdm
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.preprocessing import MinMaxScaler
from lightgbm import LGBMClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import precision_recall_curve
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import AdaBoostClassifier
#from prettytable import PrettyTable
import pickle


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

Loading Train Data

In [ ]:

```
# loading the train_csv data form drve
```

```
train_df = pd.read_csv('/content/drive/MyDrive/aps_failure_training_set.csv',header='infe
r',skiprows=20)
train_df.head()
```

Out[ ]:

	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	ag_003	ag_004	ag_005	ag_006
0	neg	76698	na	2130706438	280	0	0	0	0	0	0	37250	1432864	3664156
1	neg	33058	na	0	na	0	0	0	0	0	0	18254	653294	1720800
2	neg	41040	na	228	100	0	0	0	0	0	0	1648	370592	1883374
3	neg	12	0	70	66	0	10	0	0	0	318	2212	3232	1872
4	neg	60874	na	1368	458	0	0	0	0	0	0	43752	1966618	1800340

5 rows × 171 columns

In [ ]:

```
# derscribeing the data
train_df.describe()
```

Out[ ]:

	aa_000
count	6.000000e+04
mean	5.933650e+04
std	1.454301e+05
min	0.000000e+00
25%	8.340000e+02
50%	3.077600e+04
75%	4.866800e+04
max	2.746564e+06

In [ ]:

```
train_df.info() # informantion about the data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60000 entries, 0 to 59999
Columns: 171 entries, class to eg_000
dtypes: int64(1), object(170)
memory usage: 78.3+ MB
```

Observation: In train data there are total 60000 data point and 171 feature

Here we are going to replacing negetive class with 0 and positive class with 1

In [ ]:

```
remap = {'neg':0, 'pos': 1}  # here we are neg and pos with 0 and 1
train_df = train_df.replace(remap)
```

In [ ]:

```
train_df.head()
```

Out[ ]:

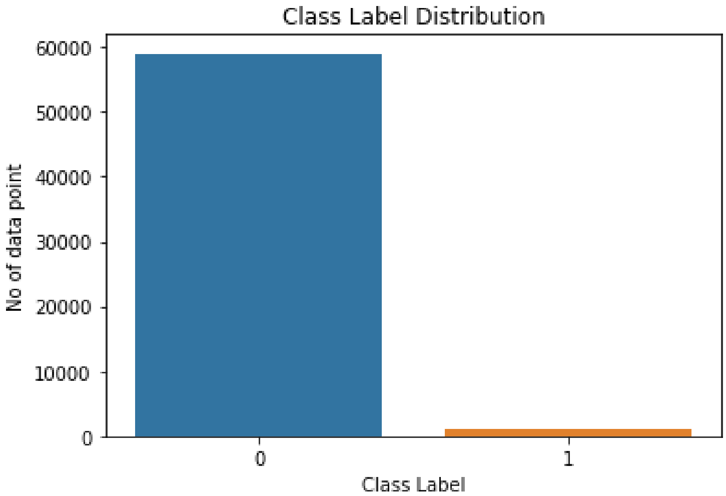
class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	ag_003	ag_004	ag_005	ag_006

	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	ag_003	ag_004	ag_005	ag_006
0	0	76698	na	2130706438	280	0	0	0	0	0	0	37250	1432864	3664156
1	0	33058	na	0	na	0	0	0	0	0	0	18254	653294	1720800
2	0	41040	na	228	100	0	0	0	0	0	0	1648	370592	1883374
3	0	12	0	70	66	0	10	0	0	0	318	2212	3232	1872
4	0	60874	na	1368	458	0	0	0	0	0	0	43752	1966618	1800340

5 rows × 171 columns

In [ ]:

```
# No of class label in data set

import seaborn as sns
x = train_df['class'].unique()
y = train_df['class'].value_counts()
sns.barplot(x, y)
plt.title('Class Label Distribution')
plt.xlabel('Class Label')
plt.ylabel('No of data point')
plt.show()
```



Observation: Data is highly Imbalance, 59000 point belong to negetive class and 1000 point belongs to positive class

In [ ]:

```
train_df = train_df.replace('na', np.NaN) # In train data replacing na value to np.NaN va
lues
```

In [ ]:

```
# which data have zero std we are going to remove those feature
def std_zero(x):
  x = x.astype(float)
  for i in x:
    if x[i].std() == 0:
      x = x.drop([i], axis = 1)
      print('feature with zero varience:', i)
x= train_df
std_zero(x)
```

```
feature with zero varience: cd_000
```

In [ ]:

```
# here we want to drop duplicate feature
train_df = train_df.T.drop_duplicates().T
```

In [ ]:

```
train_df.shape
```

Out[ ]:

```
(60000, 171)
```

Observation:

1) In train data there are only one feature which standard daviation is 0, this feature not going to add any value to the model for that we remove this row

2) train data have only one duplicate row so we remove this row.

In [ ]:

```
# Loading the test dataset
test_data = pd.read_csv('/content/drive/MyDrive/aps_failure_test_set.csv', header = 'infe
r',skiprows= 20)
```

In [ ]:

```
test_data.head(3)
```

Out[ ]:

	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	ag_003	ag_004	ag_005	ag_006	ag_00
0	neg	60	0	20	12	0	0	0	0	0	2682	4736	3862	1846	
1	neg	82	0	68	40	0	0	0	0	0	0	748	12594	3636	
2	neg	66002	2	212	112	0	0	0	0	0	199486	1358536	1952422	452706	2513

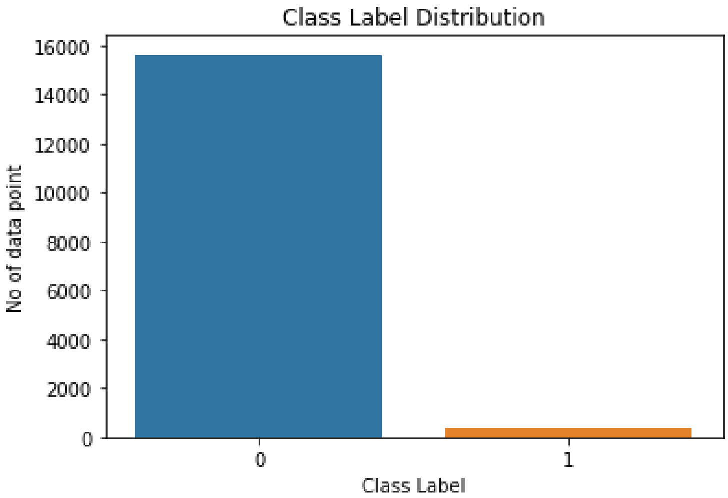
3 rows × 171 columns

In [ ]:

```
# replacing the neg with 0, pos with 1 and na to np.NaN
test_data = test_data.replace('na', np.NaN)
remap = {'neg':0, 'pos': 1}
test_data = test_data.replace(remap)
```

In [ ]:

```
# using seaborn we are going to plot the class label of dataset
x = test_data['class'].unique()
y = test_data['class'].value_counts()
sns.barplot(x, y)
plt.title('Class Label Distribution') # title of the plot
plt.xlabel('Class Label') # X label title
plt.ylabel('No of data point')# y label title
plt.show()
```



In [ ]:

```
# removing the feature which have sdt is zero
def std_zero(x):
  x = x.astype(float)
  for i in x:
    if x[i].std() == 0:
      x = x.drop([i], axis = 1)
      print('feature with zero varience:', i)
x= test_data
std_zero(x)
```

```
feature with zero varience: cd_000
```

In [ ]:

```
# removing the duplicate feature
test_data = test_data.drop_duplicates()
```

In [ ]:

```
test_data.shape
```

Out[ ]:

```
(16000, 171)
```

In [ ]:

```
test_data.head(3)
```

Out[ ]:

	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	ag_003	ag_004	ag_005	ag_006	ag_0
0	0	60	0	20	12	0	0	0	0	0	2682	4736	3862	1846	
1	0	82	0	68	40	0	0	0	0	0	0	748	12594	3636	
2	0	66002	2	212	112	0	0	0	0	0	199486	1358536	1952422	452706	251

3 rows × 171 columns

Observation: test data have 16000 data point and 171 feature, test data have same data distibution as train data

Calculating Missing value from features

In [ ]:

```
# reference = https://www.thiscodeworks.com/python-find-out-the-percentage-of-missing-val
ues-in-each-column-in-the-given-dataset-stack-overflow-python/607d4c3f6013b5001411542c

train_percent_missing = train_df.isnull().sum() * 100 / len(train_df)    # here we are cal
culating the missing values in feature
train_miss_df = pd.DataFrame({'column_name': train_df.columns,
                              'percent_missing': train_percent_missing}) # making the
dataframe using Pandas
train_miss_df.sort_values('percent_missing',ascending=False, inplace=True)   # here we are
sorting the feature in descending from on the basis of missing value
```

In [ ]:

```
train_miss_df.head(10)
```

Out[ ]:

	column_name	percent_missing

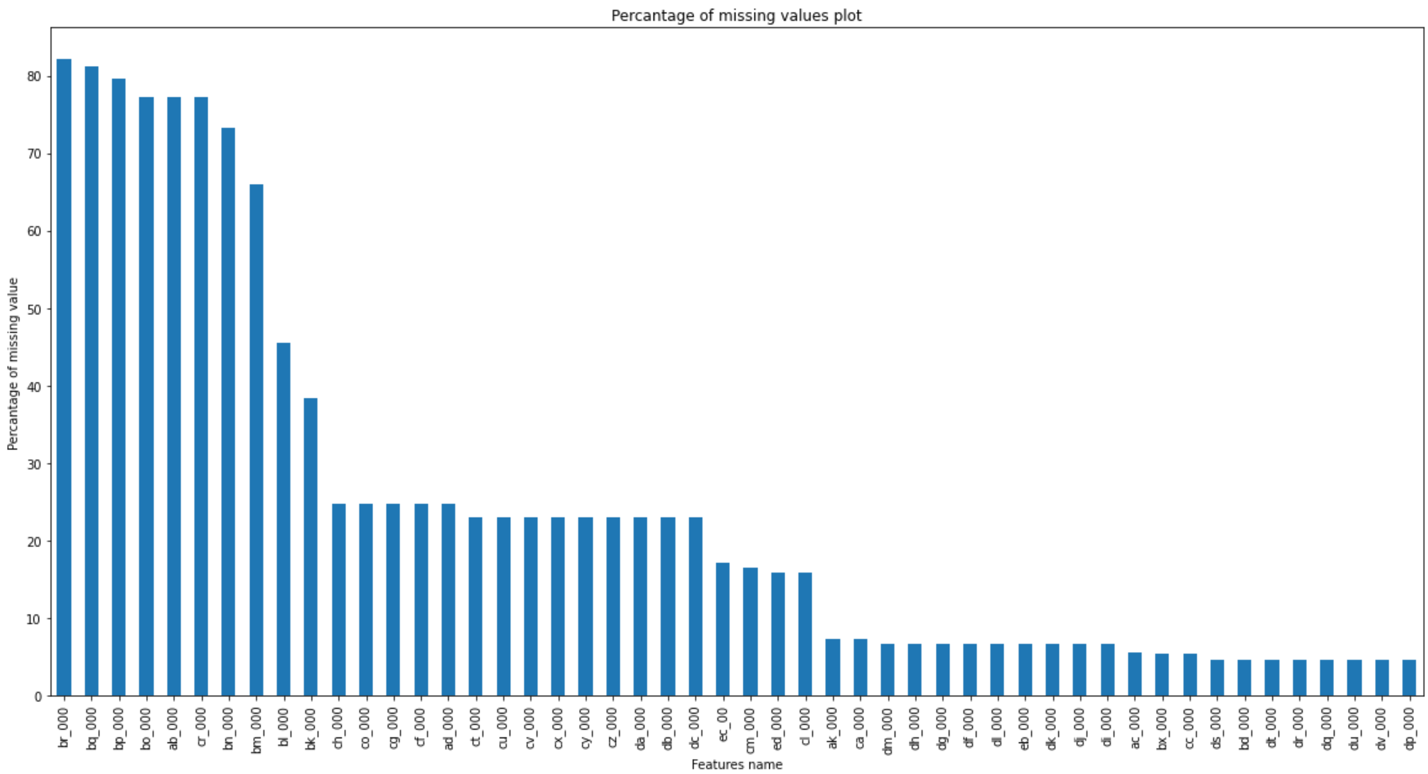
	column_name	percent_missing
br_000	br_000	82.106667
bq_000	bq_000	81.203333
bp_000	bp_000	79.566667
bo_000	bo_000	77.221667
ab_000	ab_000	77.215000
cr_000	cr_000	77.215000
bn_000	bn_000	73.348333
bm_000	bm_000	65.915000
bl_000	bl_000	45.461667
bk_000	bk_000	38.390000

In [ ]:

```
# ploting the plotbar using feature and missing value

ax= train_miss_df['percent_missing'][:50].plot.bar(figsize=(20,10)) # here we are only p
loting the top 50 feature which have maximum missing values
ax.set_xlabel('Features name')
ax.set_title('Percantage of missing values plot')
ax.set_ylabel('Percantage of missing value')
```

Out[ ]:

```
Text(0, 0.5, 'Percantage of missing value')
```



Here we are going to calculate the missing values in test data set

In [ ]:

```
# we are calculating the missing values

test_percent_missing = test_data.isnull().sum() * 100 / len(test_data) # here we are calc
ulating the missing percentage of values in feature
test_miss_df = pd.DataFrame({'column_name': test_data.columns,  # making the data freme
using the feature name and missing persentage
                             'percent_missing': test_percent_missing})
test_miss_df.sort_values('percent_missing',ascending=False, inplace=True)  # here we are
descending the feature according the missing values
```

In [ ]:

```
# printing the to 10 feature which have highest missing values
test_miss_df.head(10)
```

Out[ ]:

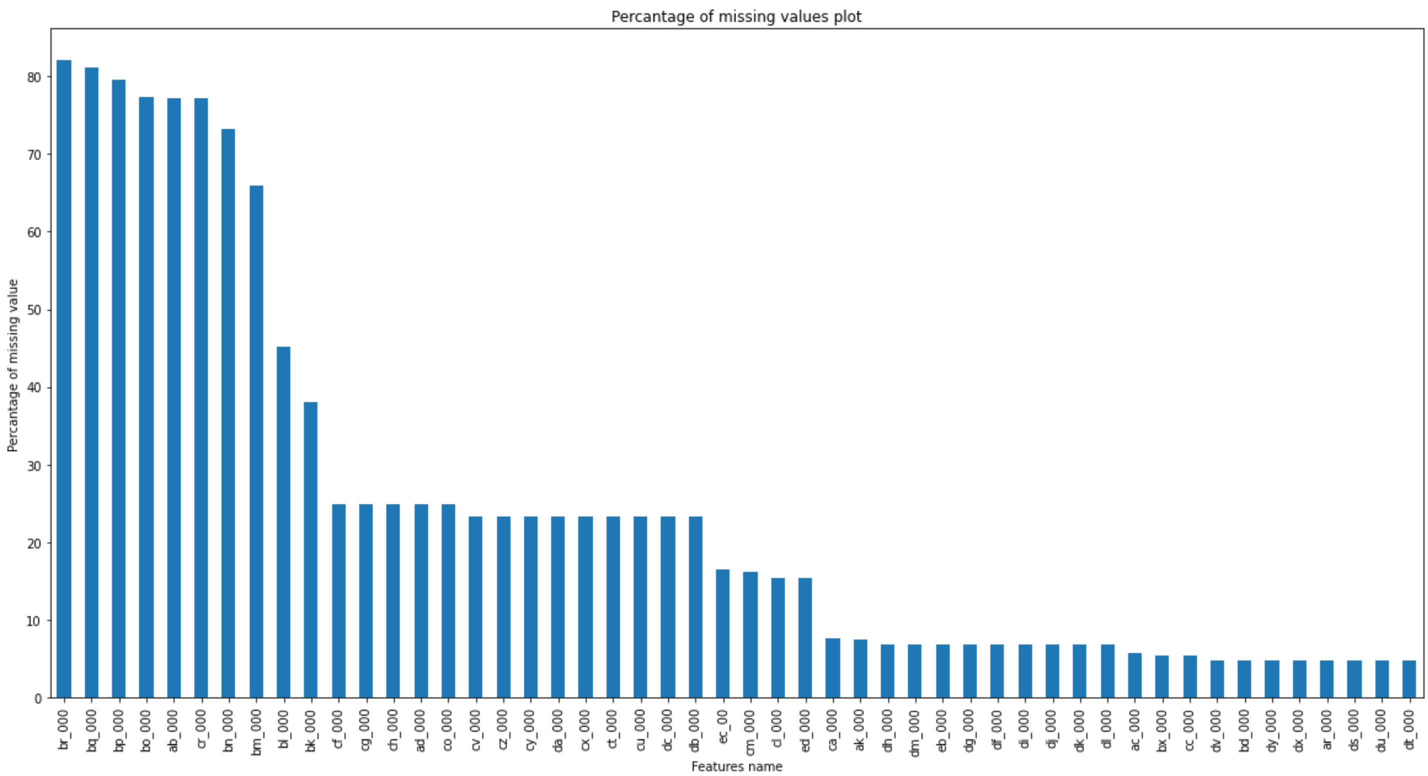
	column_name	percent_missing
br_000	br_000	82.05625
bq_000	bq_000	81.13125
bp_000	bp_000	79.50625
bo_000	bo_000	77.35000
ab_000	ab_000	77.26875
cr_000	cr_000	77.26875
bn_000	bn_000	73.20625
bm_000	bm_000	65.91250
bl_000	bl_000	45.16250
bk_000	bk_000	38.08750

In [ ]:

```
# using plotbar we are ploting the feature according to the missing values
ax= test_miss_df['percent_missing'][:50].plot.bar(figsize=(20,10))
ax.set_xlabel('Features name')   # feature name on X axis
ax.set_title('Percantage of missing values plot') # title
ax.set_ylabel('Percantage of missing value') # percentage of missing values
```

Out[ ]:

```
Text(0, 0.5, 'Percantage of missing value')
```



In [ ]:

```

```

Observation: For both train data and test data

1) 8 feature have more than 60 % of data missing

2) 16 feature have 20% to 60% value missing

3) Rest feature have less than 20% missing value

Handling Missing Data

1. **we removed those feature having missing value moer than 75 %.**
2. **we used median imputation for those feature which have less than 15 % missing values**
3. **we used KNN imputer to impute those feature which have less than 75% and more than 15% missing values.**

In [ ]:

```
# here we are elemineting those feature which have more the 75% missing values

elemineted_feature = train_percent_missing[train_percent_missing > 75].index # here we ar
e seperatig the those feature which have more than 75% missing value
train_df.drop(elemineted_feature, axis = 1, inplace = True) # here we are droping the tho
se feature
train_df.shape
```

Out[ ]:

```
(60000, 165)
```

there are total 6 feature which have more the 75% missing values. we droped those feature.

In [ ]:

```
# here we are seprating those feature which have less than 15% missing values

median_imp_feature = train_percent_missing[train_percent_missing < 15].index
median_imp_feature_df = train_df.filter(median_imp_feature) # filtering those feature whi
ch have less than 15% missing values
median_imp_feature_df.shape
```

Out[ ]:

```
(60000, 143)
```

we find that there are total 143 feature which have less than 15% missing values

In [ ]:

```
# here we are seprating and storing the those feature which have less than 75% and more t
han 15% missing values
model_imp_feature = train_percent_missing[(train_percent_missing < 75) & (train_percent_
missing >= 15)].index
model_imp_feature_df = train_df.filter(model_imp_feature)
model_imp_feature_df.shape
```

Out[ ]:

```
(60000, 22)
```

we find that there are total 22 feature which have less than 75% and more than 15% missing values

Here we are using median SimpleImputer for imputing the feature which have less than 15% missing values

In [ ]:

```
from sklearn.impute import SimpleImputer # importing simpleImputer
median_imputer = SimpleImputer(missing_values = np.NaN, strategy='median') # we use media
n strategy
```

```
median_imp_fea = median_imputer.fit_transform(median_imp_feature_df)
median_imp_df = pd.DataFrame(median_imp_fea, columns = median_imp_feature_df.columns) # making the dataframe
```

Here we are using KNN Imputer for imputing the those feature which have more than 15% and less than 75% missing values

In [ ]:

```
from sklearn.impute import KNNImputer # loading KNN Imputer
imputer = KNNImputer()
model_imp_fea = imputer.fit_transform(model_imp_feature_df)
model_imp_df = pd.DataFrame(model_imp_fea, columns = model_imp_feature_df.columns)
```

In [ ]:

```
# here we are making the data frame by concat the median imputed feature and KNN Imputed features
train_imp_df = pd.concat((model_imp_df, median_imp_df), axis = 1)
```

In [ ]:

```
print(train_imp_df.shape)
train_imp_df.head(2)
```

```
(60000, 165)
```

Out[ ]:

	ad_000	bk_000	bl_000	bm_000	bn_000	cf_000	cg_000	ch_000	cl_000	cm_000	co_000	ct_000	cu_000	cv_0(
0	280.0	330760.0	353400.0	299160.0	305200.0	2.0	96.0	0.0	6.0	1924.0	220.0	532.0	734.0	4122704
1	354.0	341420.0	359780.0	366560.0	389688.0	158.8	280.4	0.0	0.0	0.0	441.6	762.4	2077.2	3292817

2 rows × 165 columns

In [ ]:

```
train_imp_df.to_csv('/content/drive/MyDrive/train_imp_df')
```

test data preprocessing - we are doing the same process as we did in train dataset

In [ ]:

```
# here we are elemineting those feature which have more the 75% missing values

elemineted_feature = test_percent_missing[test_percent_missing > 75].index
test_data.drop(elemineted_feature, axis = 1, inplace = True)
test_data.shape
```

Out[ ]:

```
(16000, 165)
```

In [ ]:

```
# here we are seprating those feature which have less than 15% missing values

median_imp_feature = test_percent_missing[test_percent_missing < 15].index
median_imp_feature_df = test_data.filter(median_imp_feature)
median_imp_feature_df.shape
```

Out[ ]:

```
(16000, 143)
```

In [ ]:

```
# here we are seprating and storing the those feature which have less than 75% and more t
han 15% missing values
model_imp_feature = test_percent_missing[(test_percent_missing < 75) & (test_percent_mis
sing >= 15)].index
model_imp_feature_df = test_data.filter(model_imp_feature)
model_imp_feature_df.shape
```

Out[ ]:

```
(16000, 22)
```

Here we are using median SimpleImputer for imputing the feature which have less than 15% missing values

In [ ]:

```
median_imputer = SimpleImputer(missing_values = np.NaN, strategy='median')
median_imp_fea = median_imputer.fit_transform(median_imp_feature_df)
median_imp_df = pd.DataFrame(median_imp_fea, columns = median_imp_feature_df.columns)
```

In [ ]:

```
median_imp_df.shape
```

Out[ ]:

```
(16000, 143)
```

In [ ]:

```
median_imp_df.head(2)
```

Out[ ]:

	class	aa_000	ac_000	ae_000	af_000	ag_000	ag_001	ag_002	ag_003	ag_004	ag_005	ag_006	ag_007	ag_008	ag_009
0	0.0	60.0	20.0	0.0	0.0	0.0	0.0	0.0	2682.0	4736.0	3862.0	1846.0	0.0	0.0	0.0
1	0.0	82.0	68.0	0.0	0.0	0.0	0.0	0.0	0.0	748.0	12594.0	3636.0	0.0	0.0	0.0

2 rows × 143 columns

Here we are using KNN Imputer for imputing the those feature which have more than 15% and less than 75% missing values

In [ ]:

```
from sklearn.impute import KNNImputer
imputer = KNNImputer()
model_imp_test = imputer.fit_transform(model_imp_feature_df)
model_imp_df = pd.DataFrame(model_imp_test, columns = model_imp_feature_df.columns)
```

In [ ]:

```
model_imp_df.head(2)
```

Out[ ]:

	ad_000	bk_000	bl_000	bm_000	bn_000	cf_000	cg_000	ch_000	cl_000	cm_000	co_000	ct_000	cu_000	cv_000	cx_00(
0	12.0	0.0	0.0	0.0	0.0	0.0	6.0	0.0	6.0	30.0	8.0	22.0	42.0	5336.0	1276.(
1	40.0	0.0	0.0	0.0	0.0	2.0	4.0	0.0	0.0	42.0	14.0	80.0	206.0	7802.0	1466.(

In [ ]:

```
# here we are concating the model imputed feature and median imputed feature
test_imp_df = pd.concat((model_imp_df, median_imp_df), axis = 1)
test_imp_df.head()
```

Out[ ]:

	ad_000	bk_000	bl_000	bm_000	bn_000	cf_000	cg_000	ch_000	cl_000	cm_000	co_000	ct_000	cu_000	cv_0(
0	12.0	0.0	0.0	0.0	0.0	0.0	6.0	0.0	6.0	30.0	8.0	22.0	42.0	5336
1	40.0	0.0	0.0	0.0	0.0	2.0	4.0	0.0	0.0	42.0	14.0	80.0	206.0	7802
2	112.0	336240.0	194360.0	245240.0	255528.0	0.0	104.0	0.0	148.0	720.0	52.0	226.0	572.0	3593728
3	936.0	176000.0	208420.0	159380.0	169364.0	0.0	144.0	0.0	0.0	0.0	1278.0	1516.0	1398.0	2050280
4	140.0	160648.0	78068.0	82076.0	85128.0	0.0	8.0	0.0	0.0	0.0	2.0	230.0	178.0	93820

5 rows × 165 columns

In [ ]:

```
# saving the test csv file
test_imp_df.to_csv('/content/drive/MyDrive/test_imp_df')
```

Selecting Top 15 features from train data using Recursive Feature Elimination(RFE)

1) Recursive feature elimination (RFE) is a feature selection method that fits a model and removes the weakest features until the specified number of features is reached

2) here I use DecisionTreeClassifier model to select the top 15 feature

In [ ]:

```
y = train_imp_df['class']
x = train_imp_df.drop('class', axis = 1)
```

In [ ]:

```
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier

def get_top_feature(x, y, n):

  model = DecisionTreeClassifier(max_depth= 5)
  rfe = RFE(estimator= model, n_features_to_select= n)
  rfe.fit(x, y)
  top_15_feature = [k for i, k in enumerate(x.columns.tolist()) if rfe.support_[i]]
  return top_15_feature
list_of_top_15_feature = get_top_feature(x, y, 15)
print('List of top 15 feature \n:', list_of_top_15_feature)
```

```
List of top 15 feature
: ['ag_001', 'ag_002', 'ah_000', 'am_0', 'ay_002', 'ay_005', 'ay_006', 'ay_008', 'ay_009'
, 'az_004', 'bi_000', 'bj_000', 'cc_000', 'cn_004', 'cn_007']
```

Observation:

1) we can see that I have get top 15 features from the all feature.

2) In our datasets have total 170 features, out of 170 features 70 are histogram feature and 100 numerical features

3) out of top 15 feature there are 9 feature are histogram feature and 6 are numerical features

4) it means histograme features are most imprtant feature than numerical feature

here I filter the those top 15 feature which are selecting from train_imp.

In [ ]:

```
train_top_feature = train_imp_df.filter(['class','ag_001', 'ag_002', 'ah_000', 'am_0', '
as_000', 'ay_001', 'ay_005', 'ay_008', 'ay_009', 'az_004', 'bj_000', 'cc_000', 'cn_002',
'cn_007', 'ee_002'], axis =1)
```

In [ ]:

```
train_top_feature.shape
```

Out[ ]:

```
(60000, 16)
```

In [ ]:

```
train_top_feature.head()
```

Out[ ]:

	class	ag_001	ag_002	ah_000	am_0	as_000	ay_001	ay_005	ay_008	ay_009	az_004	bj_000	cc_000	cr
0	0.0	0.0	0.0	2551696.0	0.0	0.0	0.0	469014.0	755876.0	0.0	615248.0	799478.0	6167850.0	
1	0.0	0.0	0.0	1393352.0	0.0	0.0	0.0	71510.0	99560.0	0.0	1010074.0	392208.0	2942850.0	
2	0.0	0.0	0.0	1234132.0	0.0	0.0	0.0	0.0	1450312.0	0.0	1811606.0	139730.0	2560566.0	
3	0.0	0.0	0.0	2668.0	3894.0	0.0	0.0	0.0	5596.0	0.0	76.0	3090.0	7710.0	2
4	0.0	0.0	0.0	1974038.0	0.0	0.0	0.0	372236.0	584074.0	0.0	30194.0	399410.0	3946944.0	

Correlation matrix

In [ ]:

```
train_top_feature_without_softimpute = train_df.filter(['class','ag_001', 'ag_002', 'ah_0
00', 'am_0', 'as_000', 'ay_001', 'ay_005', 'ay_008', 'ay_009', 'az_004', 'bj_000', 'cc_00
0', 'cn_002', 'cn_007', 'ee_002'], axis =1)
```

In [ ]:

```
train_top_feature_without_softimpute.shape
```

Out[ ]:

```
(60000, 16)
```

In [ ]:

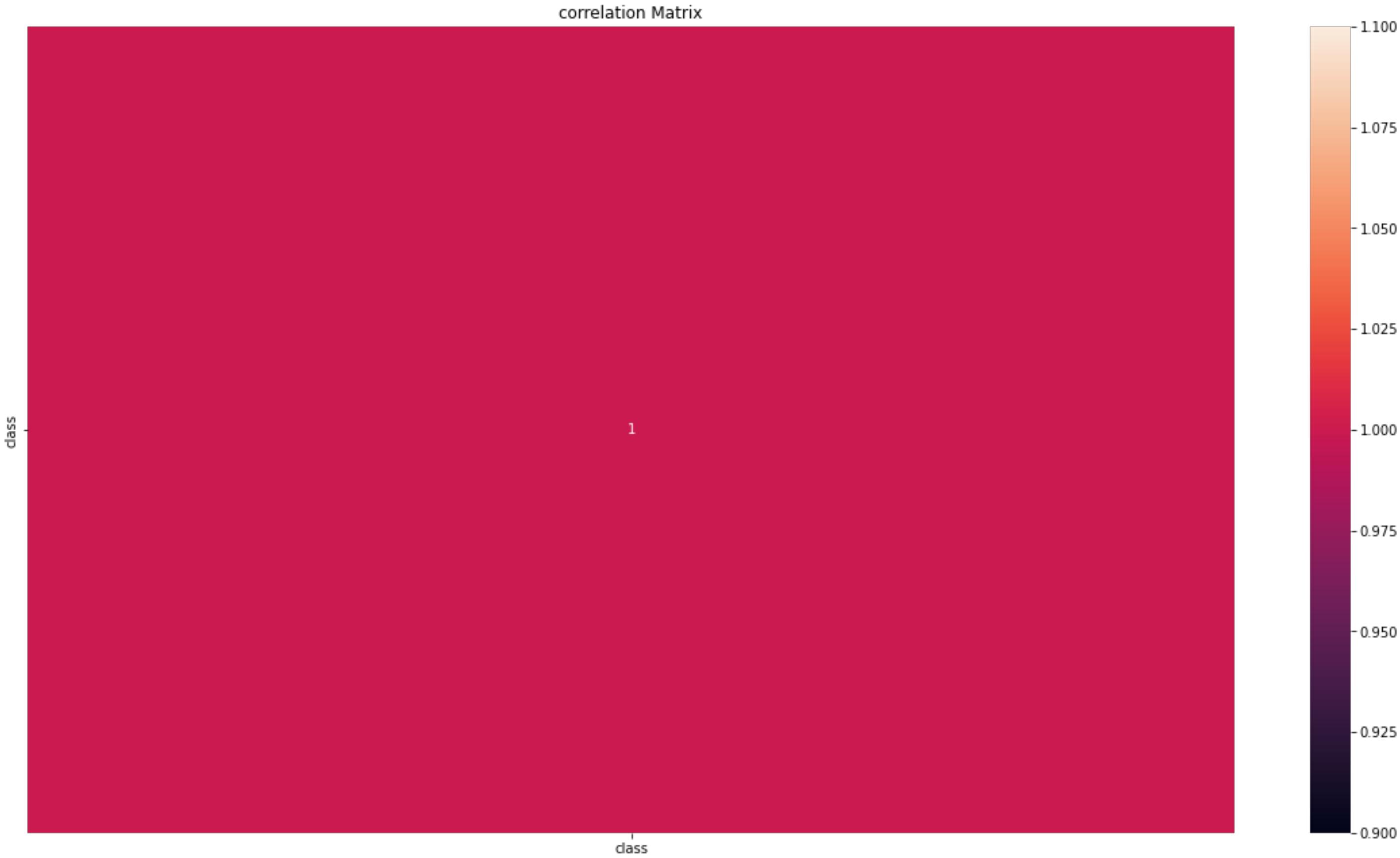
```
train_top_feature_without_softimpute.head(3)
```

Out[ ]:

	class	ag_001	ag_002	ah_000	am_0	as_000	ay_001	ay_005	ay_008	ay_009	az_004	bj_000	cc_000	cn_002	cn_(
0	0	0	0	2551696	0	0	0	469014	755876	0	615248	799478	6167850	0	988
1	0	0	0	1393352	0	0	0	71510	99560	0	1010074	392208	2942850	38	362
2	0	0	0	1234132	0	0	0	0	1450312	0	1811606	139730	2560566	0	102

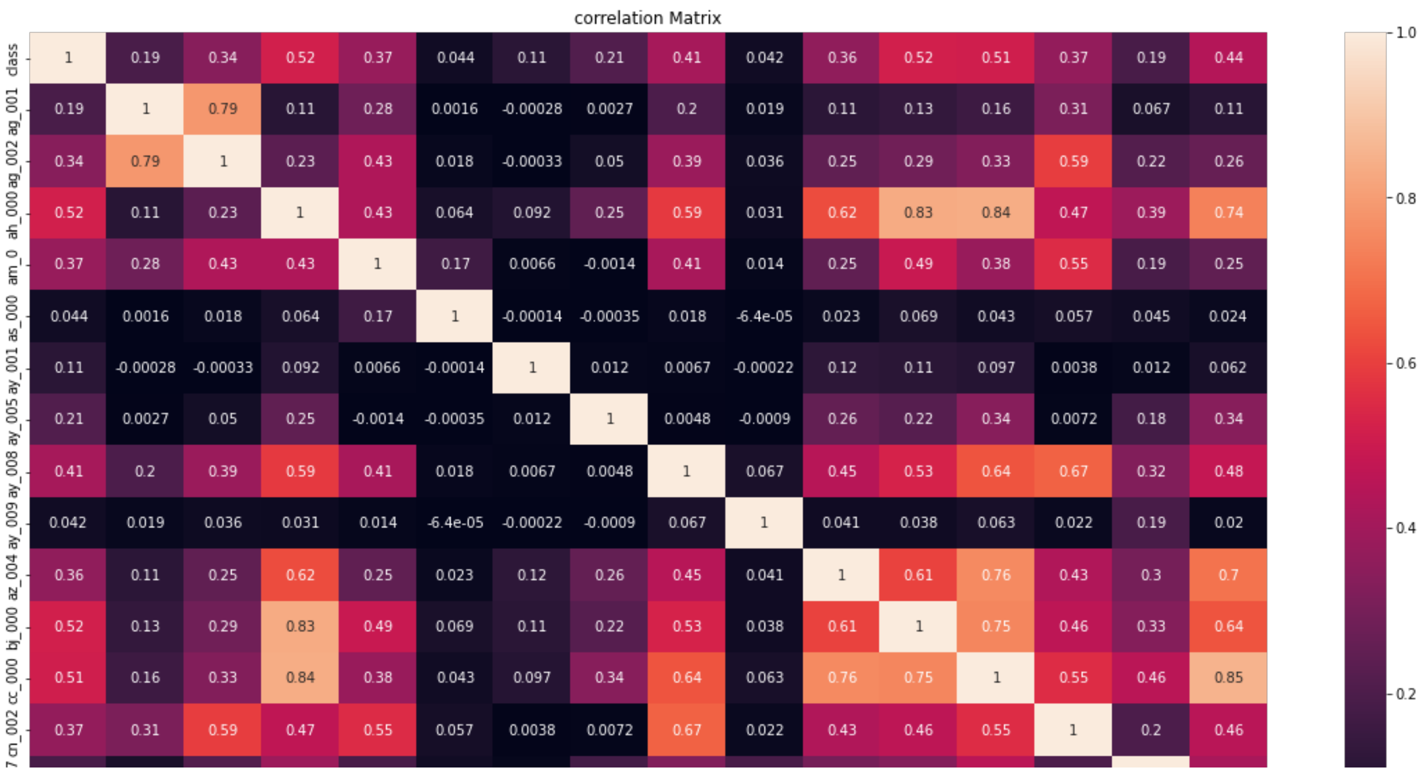
In [ ]:

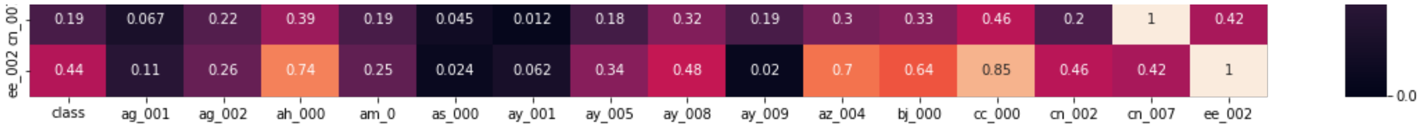
```python
import seaborn as sns
plt.figure(figsize=(20,11))
cor = train_top_feature_without_softimpute.corr()
sns.heatmap(cor, annot= True)
plt.title('correlation Matrix')
plt.show()
```



In [ ]:

```python
import seaborn as sns
plt.figure(figsize=(20,11))
cor = train_top_feature.corr()
sns.heatmap(cor, annot= True)
plt.title('correlation Matrix')
plt.show()
```





Observation before appling soft impute on train data

1. all values looks like equaly correlated to each other

Observation after apply softimpute on train data

1. we can see that features are not highly correlated to each other.
2. In this plot we can see that some feature are not higly correlated like **ay_001 to ag_001, ay_001 to ag_002, ay_001 to as_000, ay_009 to as_000, ay_001 to cn_002, ay_005 to cn_002**

we can see that after applying the softimpute on train data some feature becomes more correlate to each other and some features becomes less correlate to each other

In [ ]:

```

```

Bivariate Analysis of feature which are not higly correlated to each other

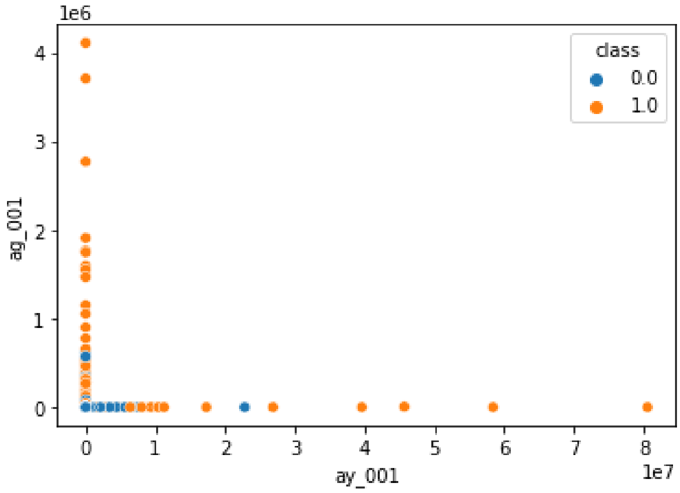
1. we are going to do Bivariate analysis of these feature like **ay_001 to ag_001, ay_001 to ag_002, ay_001 to as_000, ay_009 to as_000, ay_001 to cn_002, ay_005 to cn_002**

In [ ]:

```
# ploting scatterplot using two feature ay_001 and ag_001

sns.scatterplot(train_top_feature["ay_001"],train_top_feature["ag_001"],hue=train_top_fea
ture["class"])
```

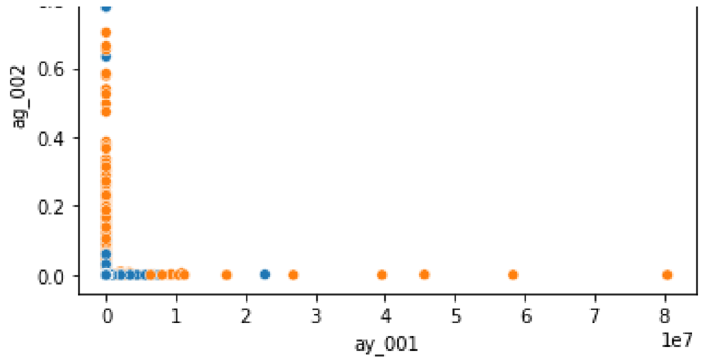
Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd528cbb250>
```



In [ ]:

```
#ploting scatterplot using two feature ay_001 and ag_002

sns.scatterplot(train_top_feature["ay_001"],train_top_feature["ag_002"],hue=train_top_fea
ture["class"])
```

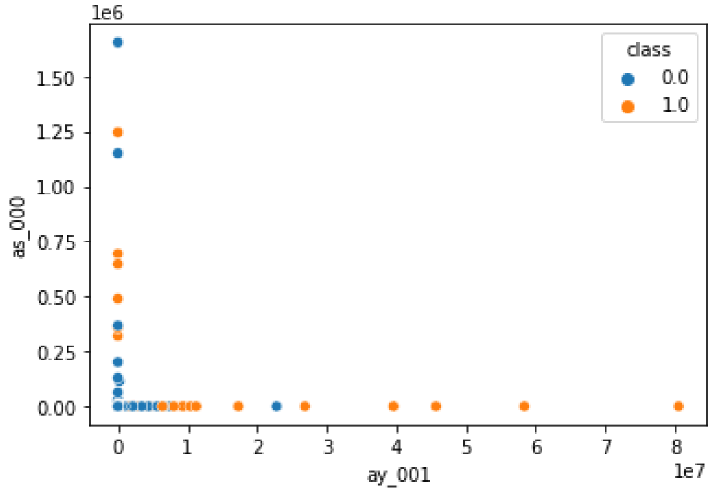


In [ ]:

```
#ploting scatterplot using two feature ay_001 and as_000

sns.scatterplot(train_top_feature["ay_001"],train_top_feature["as_000"],hue=train_top_fea
ture["class"])
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd526a97210>
```

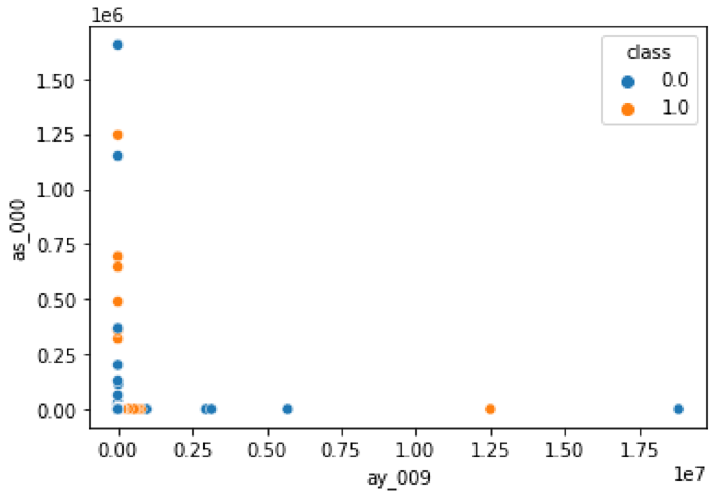


In [ ]:

```
#ploting scatterplot using two feature ay_009 and as_000

sns.scatterplot(train_top_feature["ay_009"],train_top_feature["as_000"],hue=train_top_fea
ture["class"])
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd526a42310>
```



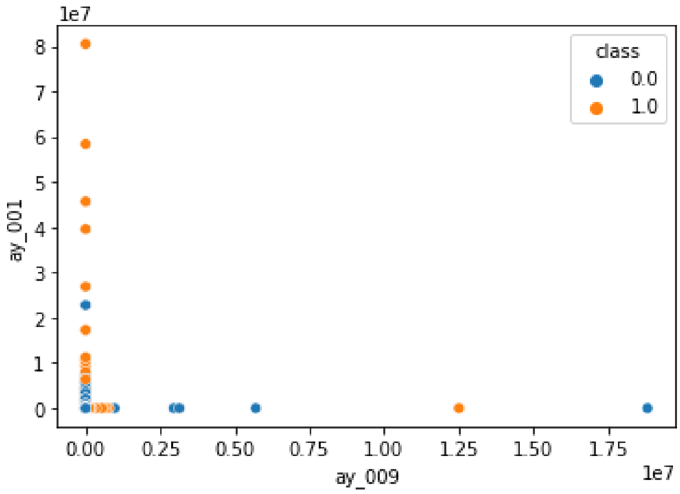
In [ ]:

```
#ploting scatterplot using two feature ay_009 and ay_001
```

```
sns.scatterplot(train_top_feature["ay_009"],train_top_feature["ay_001"],hue=train_top_fea
ture["class"])
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd528988110>
```

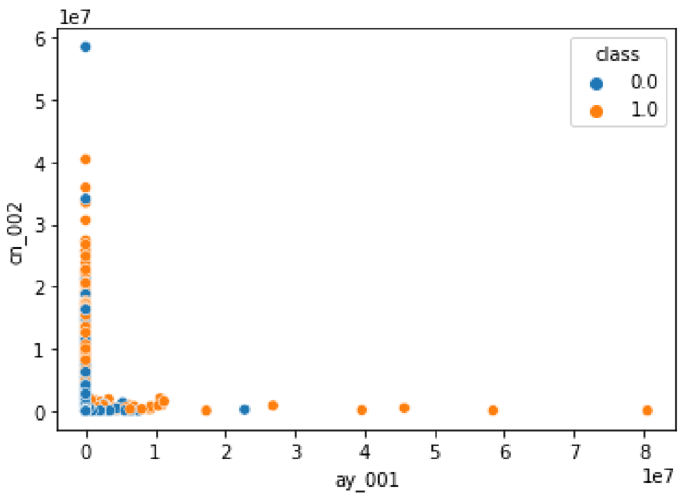


In [ ]:

```
#ploting scatterplot using two feature ay_001 and cn_002

sns.scatterplot(train_top_feature["ay_001"],train_top_feature["cn_002"],hue=train_top_fea
ture["class"])
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd528933e50>
```

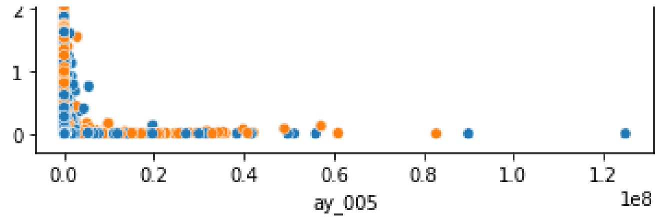


In [ ]:

```
#ploting scatterplot using two feature ay_005 and cn_002

sns.scatterplot(train_top_feature["ay_005"],train_top_feature["cn_002"],hue=train_top_fea
ture["class"])
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd52d81f790>
```



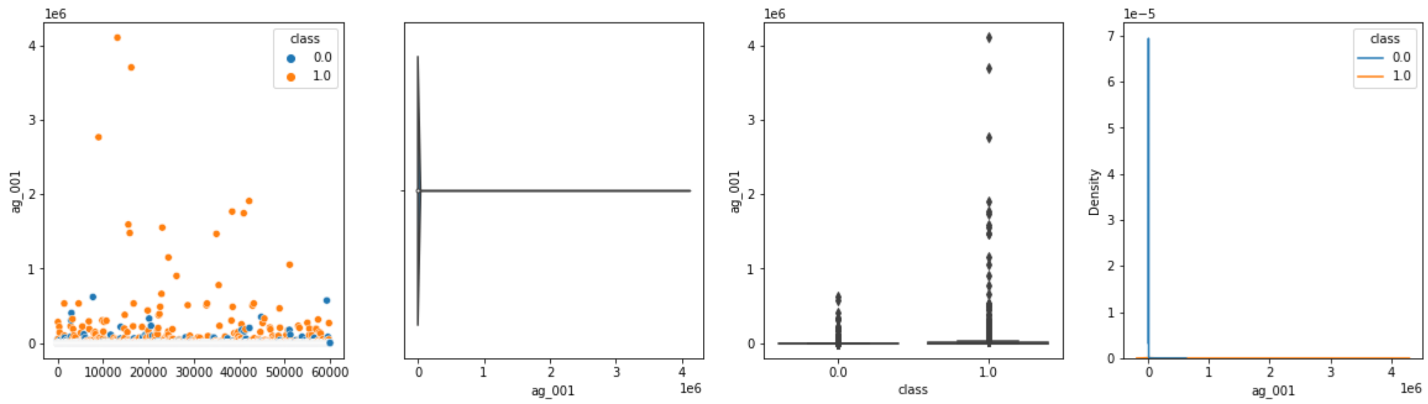
Observation

We can see that all 9 scatter plot feature are not correlated to each other.

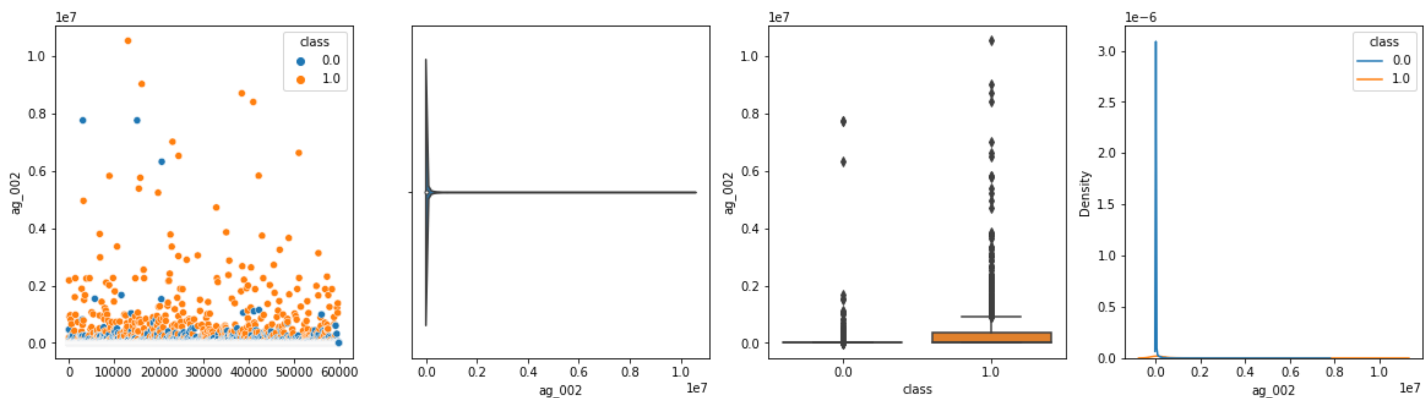
Univariate analysis of the top_15 features

In [ ]:

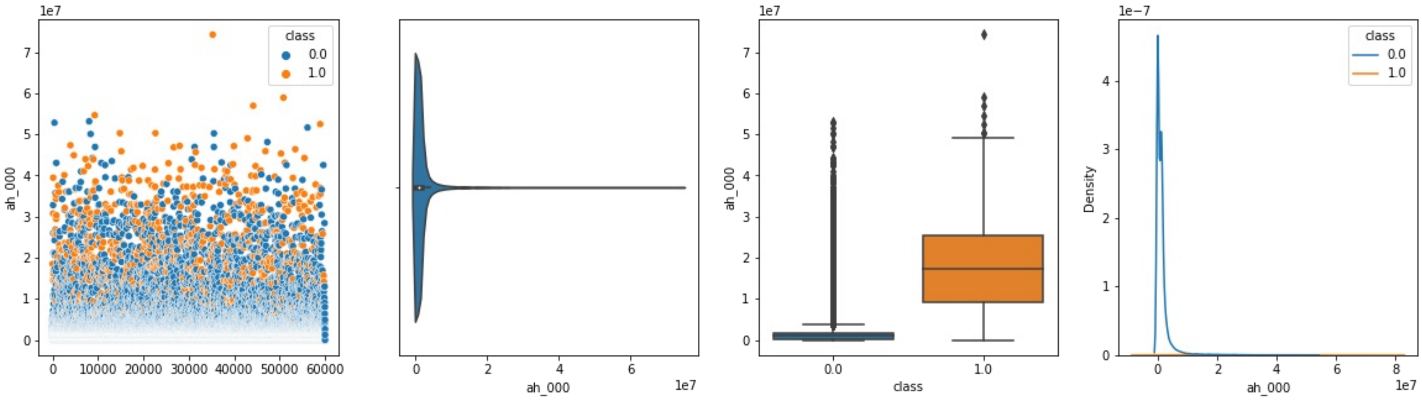
```
def feature_plot(x):
  for i in x.columns.tolist(): # we are creating list of all fature name
    if i != 'class':

      fig, ax = plt.subplots(1,4, figsize = (20, 5)) # here we are going to plot the 4 diff plot

      sns.scatterplot(x.index, x[i], hue = x['class'], ax = ax[0]) # here we ploting scatter plot to visulize the data
      sns.violinplot(x = x[i], ax = ax[1]) # here we are ploting violin plot
      sns.boxplot(x = x['class'], y = x[i], ax = ax[2]) # here we ploting boxplot
      sns.kdeplot(data = x , x = x[i], hue = 'class', ax =ax[3])
      plt.show()
      print(i)
      print('*'*200)



x = train_top_feature
feature_plot(x)
```



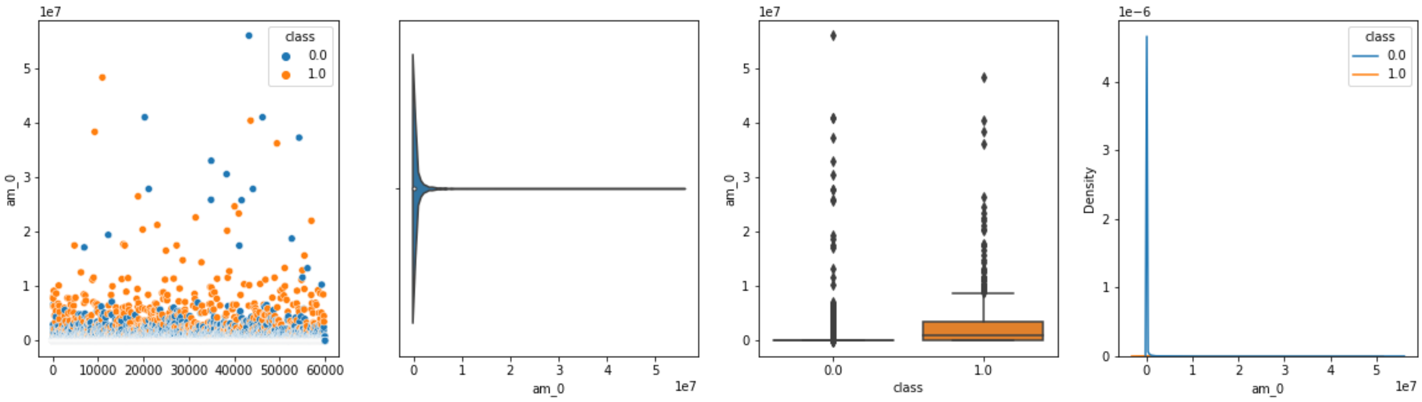

```
ag_001
********************************************************************************************************************************************************************************************************
```



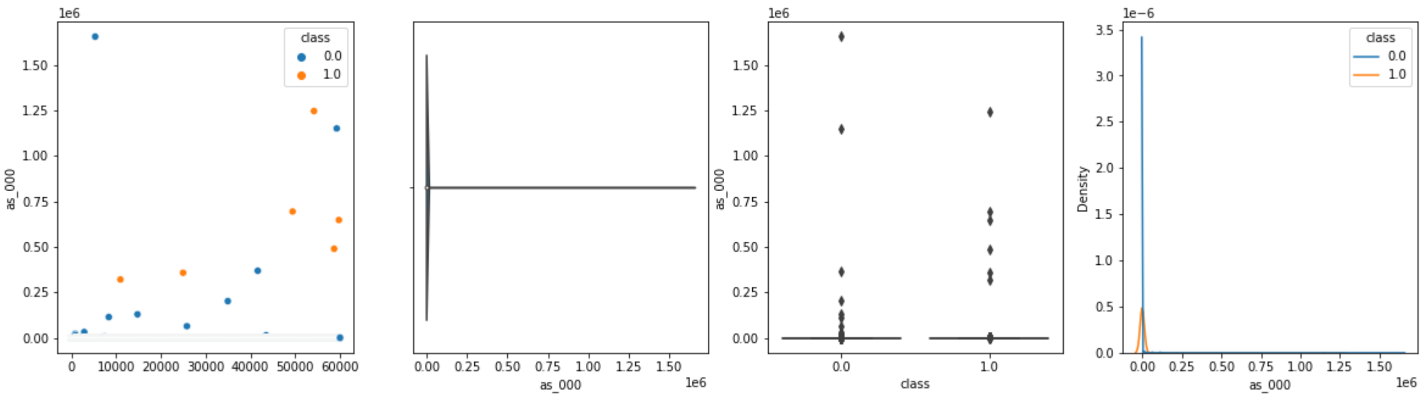
ag_002
****
****
****



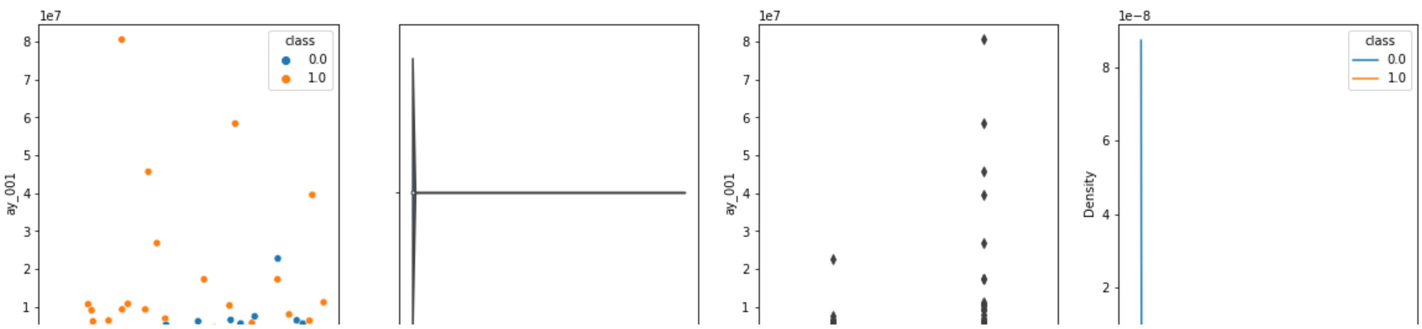
ah_000
****
****
****



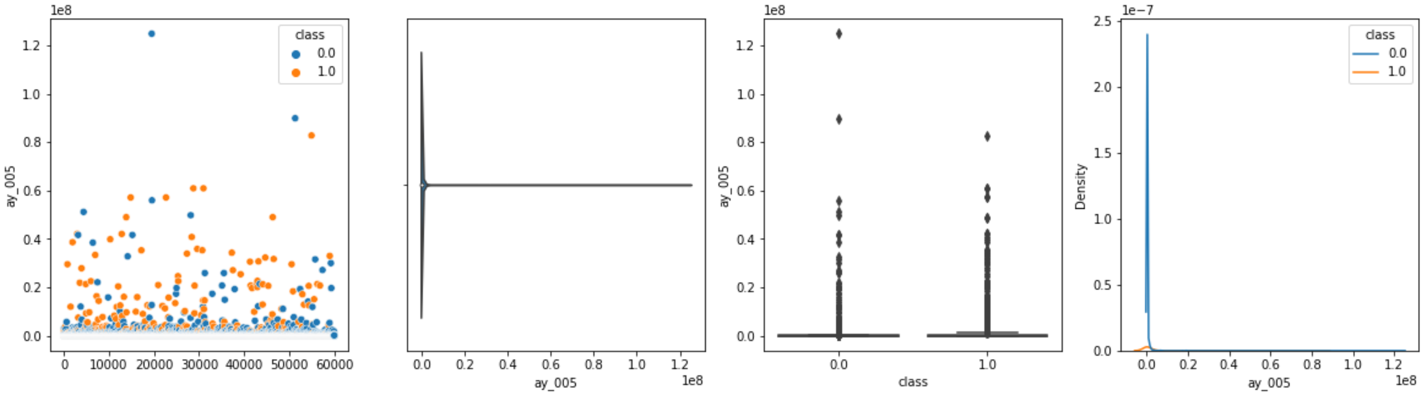
am_0
****
****
****



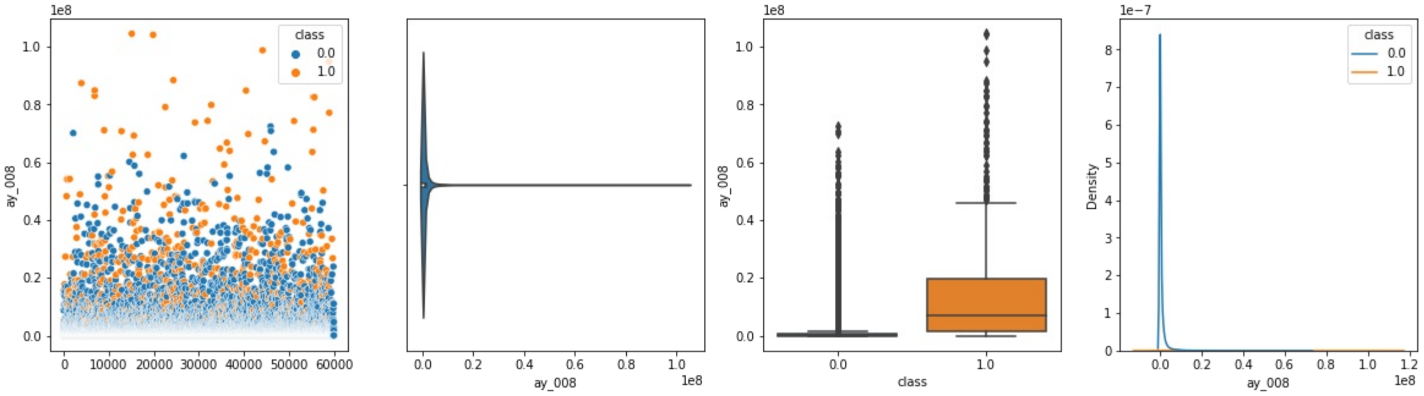
as_000
****
****
****



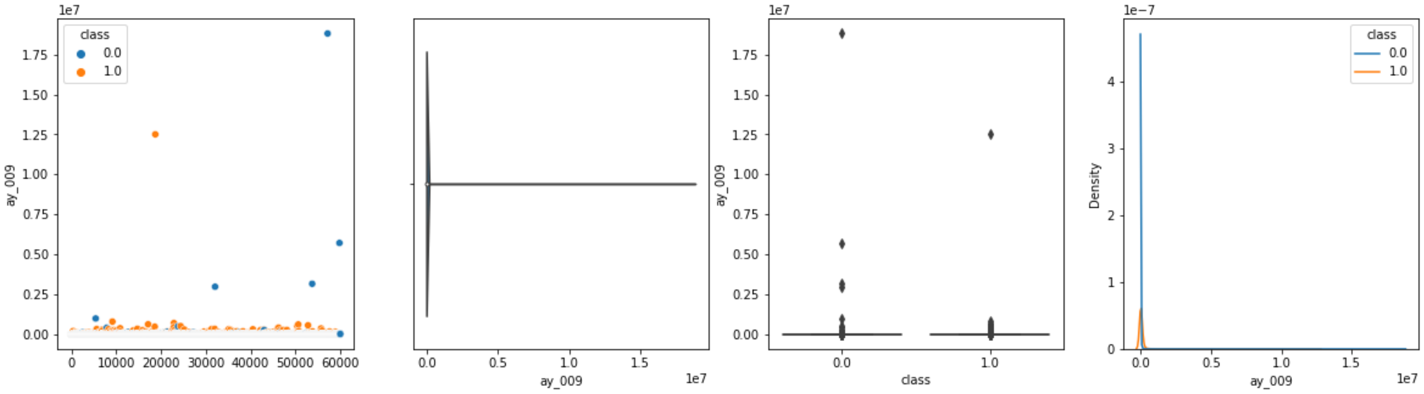
ay_001
****
****
****



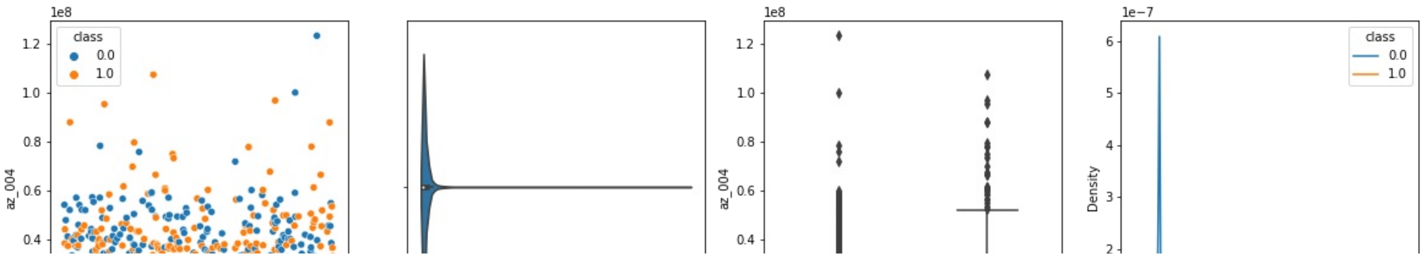
ay_005
****
****
****

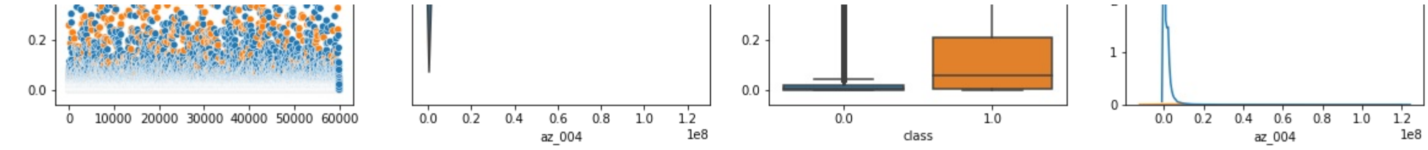


ay_008
****
****
****

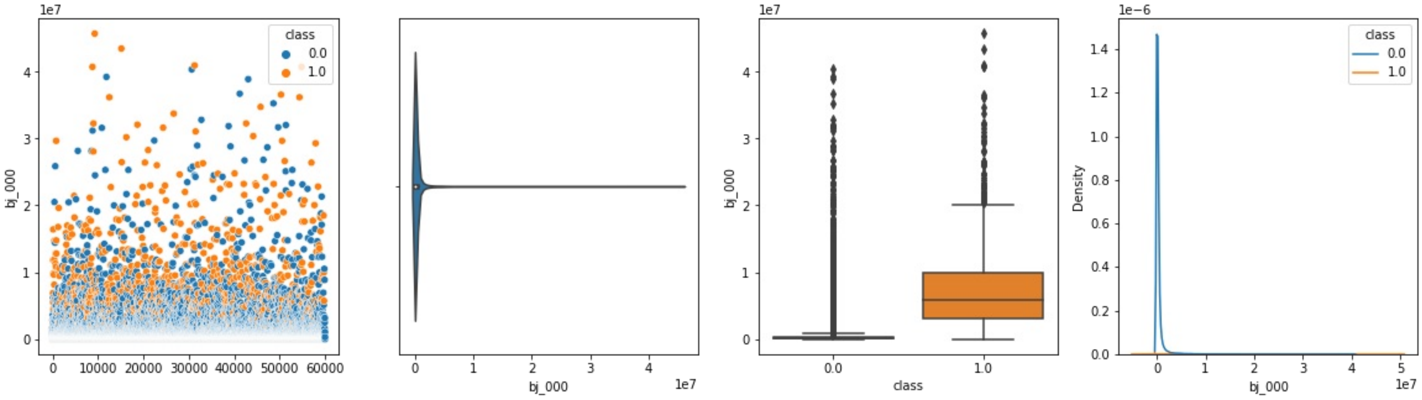


ay_009
****
****
****

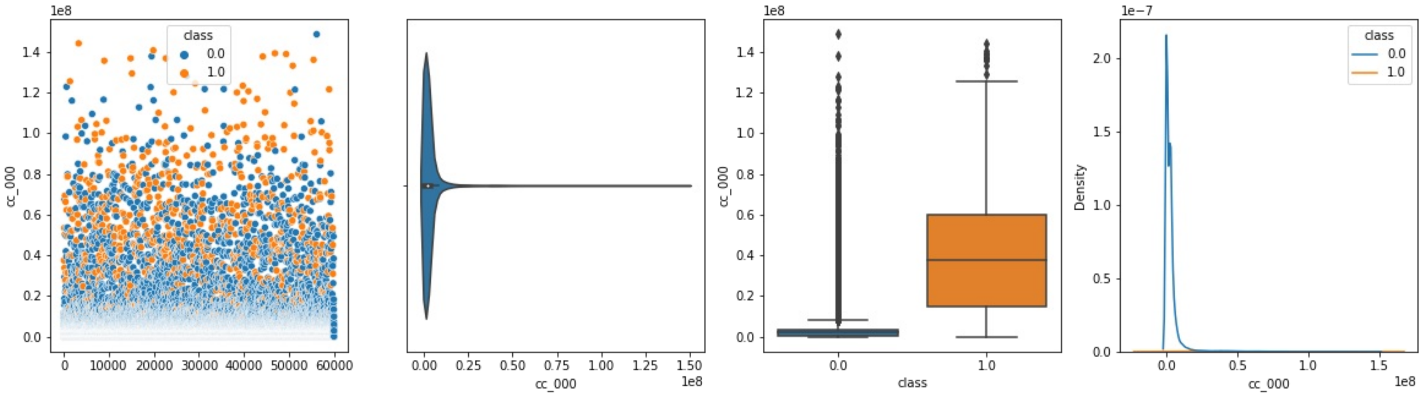




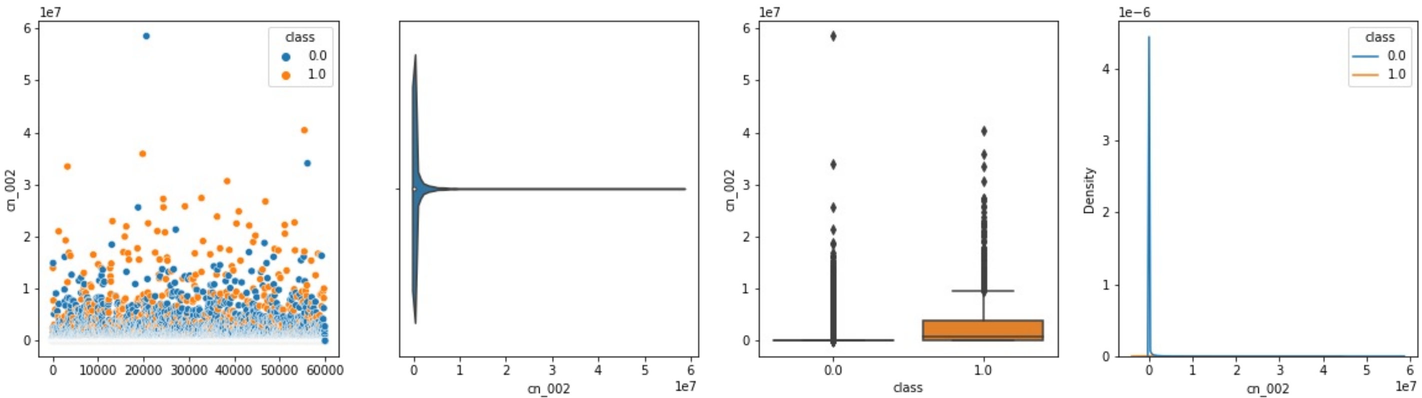
az_004
****
****
****



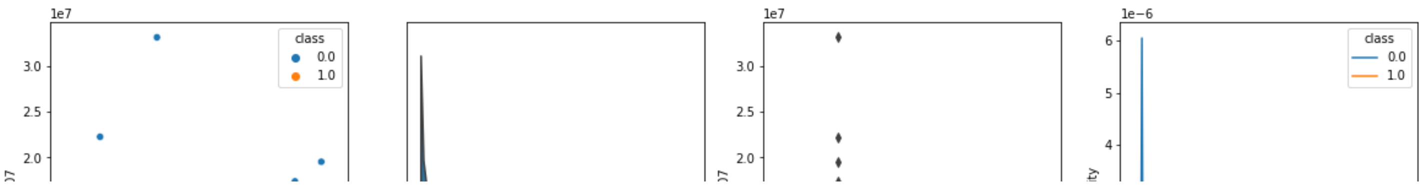
bj_000
****
****
****

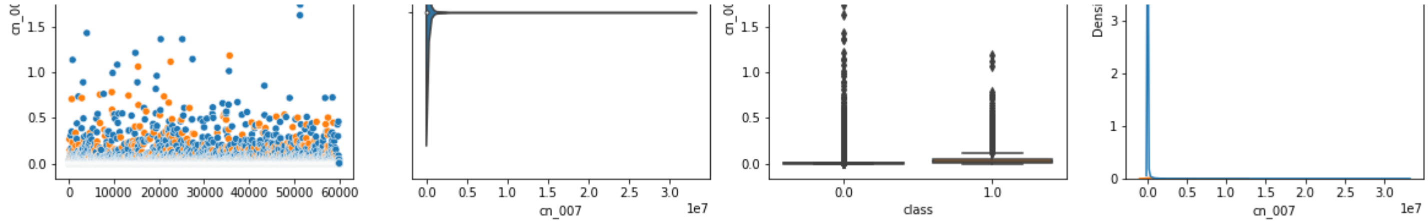


cc_000
****
****
****

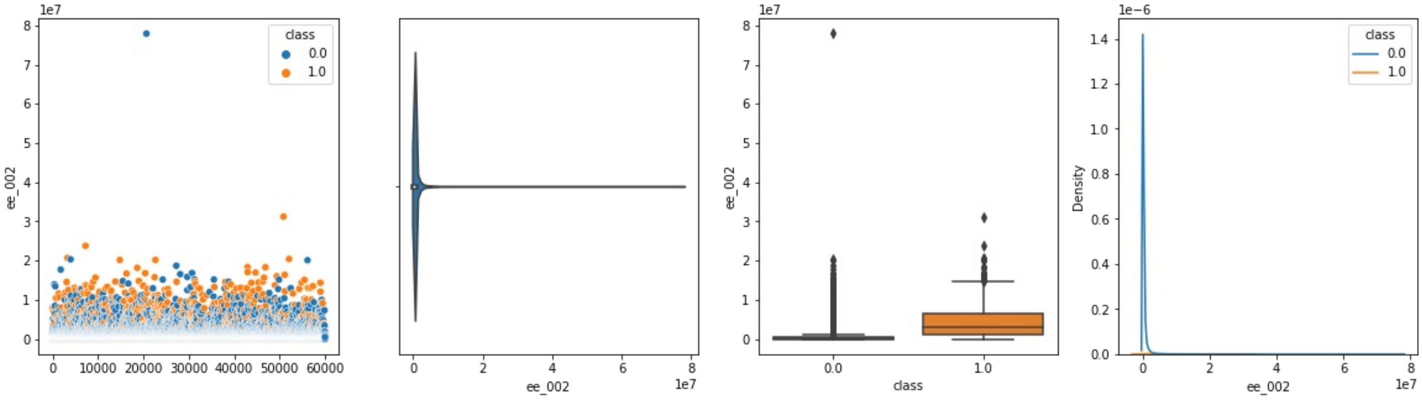


cn_002
****
****
****






```
cn_007
*****************************************************************************************************************************************************************************
```




```
ee_002
*****************************************************************************************************************************************************************************
```

In [ ]:

```

```

Observation:

1) In this feature ag_000, ay_002 majority of value are positive class. if we increase the value then it higher chance of APS failure.

2) In this Feature as_000, ay_001, ay_005, ay_009, cn_007 majority of values are negetive datapoint there are high chance of no failure in APS part.

3) in this feature ah_000, ay_008, cc_000, bg_000 there are very high chance of APS failure.

analysis of the outlier in given top_15 feature

we can see that in our given dataset we dont have exact name and meaning of the every feature for that it become very hard to know about features and there uses. here I am going to do analysis of outlier in top 15 features

In [ ]:

```
print(train_top_feature.columns)
```

```
Index(['class', 'ag_001', 'ag_002', 'ah_000', 'am_0', 'as_000', 'ay_001',
       'ay_005', 'ay_008', 'ay_009', 'az_004', 'bj_000', 'cc_000', 'cn_002',
       'cn_007', 'ee_002'],
      dtype='object')
```

In [ ]:

```
def get_outlier(x):
  for i in x.columns.tolist():
    if i != 'class':
```

```
        data_dis = x[i].describe()
        min_thresold = x[i].quantile(0.05)
        max_thresold = x[i].quantile(0.95)
        print(i)
        print(data_dis)
        print(min_thresold)
        print(max_thresold)
        print('*'*200)

x = train_top_feature
get_outlier(x)
```

```
ag_001
count    6.000000e+04
mean     9.693017e+02
std      3.400916e+04
min     -1.512056e+02
25%      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
max      4.109372e+06
Name: ag_001, dtype: float64
0.0
0.0
******************************************************************************************
******************************************************************************************
********************
ag_002
count    6.000000e+04
mean     8.552339e+03
std      1.494873e+05
min     -4.355937e+03
25%      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
max      1.055286e+07
Name: ag_002, dtype: float64
0.0
330.19963120213856
******************************************************************************************
******************************************************************************************
********************
ah_000
count    6.000000e+04
mean     1.797480e+06
std      4.168846e+06
min      0.000000e+00
25%      2.880150e+04
50%      9.933330e+05
75%      1.592020e+06
max      7.424732e+07
Name: ah_000, dtype: float64
1754.0
6722331.799999996
******************************************************************************************
******************************************************************************************
********************
am_0
count    6.000000e+04
mean     9.292183e+04
std      8.452472e+05
min     -4.967076e+04
25%      0.000000e+00
50%      0.000000e+00
75%      2.386000e+03
max      5.590351e+07
Name: am_0, dtype: float64
0.0
226292.39999999944
******************************************************************************************
******************************************************************************************
********************
```

```
as_000
count    6.000000e+04
mean     1.264304e+02
std      1.095223e+04
min     -9.267898e+01
25%      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
max      1.655240e+06
Name: as_000, dtype: float64
0.0
0.0
******************************************************************************************
******************************************************************************************
**********************
ay_001
count    6.000000e+04
mean     1.012440e+04
std      5.322724e+05
min     -5.500681e+03
25%      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
max      8.052538e+07
Name: ay_001, dtype: float64
0.0
0.0
******************************************************************************************
******************************************************************************************
**********************
ay_005
count    6.000000e+04
mean     1.111321e+05
std      1.386864e+06
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      3.979900e+04
max      1.249489e+08
Name: ay_005, dtype: float64
0.0
259547.79999999967
******************************************************************************************
******************************************************************************************
**********************
ay_008
count    6.000000e+04
mean     1.042500e+06
std      3.970809e+06
min      0.000000e+00
25%      7.274000e+03
50%      9.239600e+04
75%      6.063320e+05
max      1.045670e+08
Name: ay_008, dtype: float64
0.0
4003360.0
******************************************************************************************
******************************************************************************************
**********************
ay_009
count    6.000000e+04
mean     1.154179e+03
std      9.741164e+04
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
max      1.882466e+07
Name: ay_009, dtype: float64
0.0
0.0
```

```
*******************************************************************************************
*******************************************************************************************
**********************
az_004
count    6.000000e+04
mean     1.463944e+06
std      4.164118e+06
min      0.000000e+00
25%      1.538000e+03
50%      7.758800e+04
75%      1.762460e+06
max      1.230471e+08
Name: az_004, dtype: float64
36.0
5190050.999999997
*******************************************************************************************
*******************************************************************************************
**********************
bj_000
count    6.000000e+04
mean     5.073617e+05
std      1.812721e+06
min      0.000000e+00
25%      8.318000e+03
50%      1.529470e+05
75%      3.321800e+05
max      4.573632e+07
Name: bj_000, dtype: float64
1652.0
1713086.0
*******************************************************************************************
*******************************************************************************************
**********************
cc_000
count    6.000000e+04
mean     3.714760e+06
std      9.389178e+06
min      0.000000e+00
25%      6.557300e+04
50%      2.053699e+06
75%      3.355328e+06
max      1.486152e+08
Name: cc_000, dtype: float64
5184.0
11948397.399999997
*******************************************************************************************
*******************************************************************************************
**********************
cn_002
count    6.000000e+04
mean     1.598198e+05
std      1.067992e+06
min     -5.086192e+04
25%      0.000000e+00
50%      0.000000e+00
75%      7.944500e+03
max      5.850861e+07
Name: cn_002, dtype: float64
0.0
490791.1999999999
*******************************************************************************************
*******************************************************************************************
**********************
cn_007
count    6.000000e+04
mean     6.388406e+04
std      4.042037e+05
min      0.000000e+00
25%      6.200000e+01
50%      9.787000e+03
75%      3.087950e+04
max      3.314373e+07
```

```
Name: cn_007, dtype: float64
0.0
182814.49999999924
******************************************************************************************
******************************************************************************************
**********************
ee_002
count    6.000000e+04
mean     4.416533e+05
std      1.150251e+06
min      0.000000e+00
25%      2.818000e+03
50%      2.292280e+05
75%      4.359115e+05
max      7.793393e+07
Name: ee_002, dtype: float64
44.0
1501448.6999999986
******************************************************************************************
******************************************************************************************
**********************
```

Observation:

1. **we can clearly see in boxplot that almost every feature have some outliers.**
2. **some feature like ag_001, ag_002, as_000, ay_001, ay_009 have more than 75% value are zero. and very few value are very high.**
3. **there are some like cn_00, ay_005, am_0 have more than 50% value are zero.**

Performance Matrix

1. **The this given problem we have to calculate the Total_cost and cost function is Total_cost = (Cost_1** ***No of Instance) + (Cost_2*** **No of Instance)**
2. **Where Cost_1 is refers to the cost of Unnesary cheack of the APS system and cost_2 refers to the cost of missing a fualty APS part which may cause of a breakdown of other part also.**
3. **In given problem cost of Cost_1 is 10 and Cost_2 is 500.**
4. **if we use F1 score as Performance Matrix then we can use False Negetive(FN) for Cost_1 and False Positive(FP) for the Cost_2.**
5. **After EDA I found data is higly Imbalance and there are lots of outlier for these all problem we are going to use Macro F1 Score as Performance Matrics.**
6. **Basically In Macro F1 Score we calculate the individual F1 score of every variable and took the average of all F1 score and it help to deal we imbalance data.**

Feature engineering

Loading train and test data

In [ ]:

```python
train_data = pd.read_csv('/content/drive/MyDrive/train_imp_df') # loading train_data
print(train_data.shape)
```

```
(60000, 166)
```

In [ ]:

```python
test_data = pd.read_csv('/content/drive/MyDrive/test_imp_df') # loading test data
print(test_data.shape)
```

```
(16000, 166)
```

here we are seprating the class and feature in X, Y files

In [ ]:

```
# making x_train, X_test, y_train and y_test from train and test data
Y_train = train_data['class']
X_train = train_data.drop('class', axis = 1)

Y_test = test_data['class']
X_test = test_data.drop('class', axis = 1)
```

In [ ]:

```
print(Y_train.shape)
print(X_train.shape)
print(Y_test.shape)
print(X_test.shape)
```

```
(60000,)
(60000, 165)
(16000,)
(16000, 165)
```

Here we are doing scaling of train and test data using standard scaler

Standardization used to scale the feature values. In standardization functionality does not limit value between 0 and 1, so any outlier in data will not be impacted due to this transformation

In [ ]:

```
# scaling the X_train and X_test data
from sklearn.preprocessing import StandardScaler
scalar =StandardScaler()
scalar.fit(X_train)

X_train_std = scalar.transform(X_train) #
X_test_std = scalar.transform(X_test)

X_train_std = pd.DataFrame(X_train_std, columns = X_train.columns)
X_test_std = pd.DataFrame(X_test_std, columns = X_test.columns)
```

Here we are using the PCA for feature extraction

Principal Component Analysis (PCA) is one of the most commonly used unsupervised machine learning algorithms across a variety of applications: exploratory data analysis, dimensionality reduction, information compression, data de-noising.

In [ ]:

```
from numpy import random
from sklearn.decomposition import PCA

pca = PCA(n_components= 0.95) # here we reducing the dimension of the data with 95% of va
riance
pca.fit_transform(X_train_std)

X_train_pca = pca.transform(X_train_std)
X_test_pca = pca.transform(X_test_std)
```

In [ ]:

```
X_train_pca.shape
```

Out[ ]:

```
(60000, 80)
```

```
(60000, 80)
```

here we can see that we reduced the 50% of the fature and we create the 80 new feature with 95% variance

In [ ]:

```
X_test_pca.shape
```

Out[ ]:

```
(16000, 80)
```

In [ ]:

```
X_train_pca = pd.DataFrame(X_train_pca)
X_test_pca = pd.DataFrame(X_test_pca)
```

In [ ]:

```
X_train_pca.head(2)
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	
0	2.424037	-1.781991	-1.924409	-0.220769	1.052864	-0.308300	0.196385	0.010110	0.344692	0.098782	-0.559783	-0.052007	0.
1	-0.355140	-0.494459	-0.633625	-0.102899	0.549825	-0.488053	-0.030201	-0.008248	-0.024740	0.072605	0.105517	-0.134701	0.

Here we are adding the 80 feature which we find using PCA

In [ ]:

```
X_train_final = pd.concat((X_train_pca, X_train_std), axis = 1) # concating the pca_feat
ure and origina feature
X_test_final = pd.concat((X_test_pca, X_test_std), axis = 1)
```

In [ ]:

```
print(X_train_final.shape)
print(X_test_final.shape)
```

```
(60000, 245)
(16000, 245)
```

In [ ]:

```
X_train_final.head(2)
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	
0	-3.245033	0.363052	0.925147	0.299916	-0.383716	0.448328	0.037163	-0.041685	-0.042848	0.058541	-0.013572	-0.120452	0.
1	-0.876984	-0.473164	-0.533369	-0.553815	0.964213	-0.875762	-0.295314	0.008281	-0.064447	-0.109764	-0.026785	-0.059797	0.

2 rows × 245 columns

In [ ]:

```

```

Ore data is highly Imbalance, to deal with imbalnce data we are using SMOTE

1. **Synthetic Minority Oversampling Technique (SMOTE) is a type of data augmentation for the minority class. When we have imbalance data set then we use SMOTE to balance the data.**
2. **For generating the synthetics point from the minority class, we select a minority class instance 'a' at random and find its K nearest neighbors. The synthetics is then created by choosing one of the K nearest neighbors b at random and connecting a and b to form a line segment in the feature space. The synthetic instance are generated convex combination of the two chosen instance a and b.**

In [ ]:

```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
# define pipeline
over = SMOTE(sampling_strategy=0.3,random_state=42)
under = RandomUnderSampler(sampling_strategy=0.5)
steps = [('o', over), ('u', under)]
pipeline = Pipeline(steps=steps)
# transform the dataset
X_train_final, y_train_final = pipeline.fit_resample(X_train_final, Y_train)
```

In [ ]:

```
X_train_final.shape
```

Out[ ]:

```
(53100, 245)
```

In [ ]:

```
print(y_train_final.value_counts())
```

```
0.0    35400
1.0    17700
Name: class, dtype: int64
```

In [ ]:

```
train_final_data = pd.concat((X_train_final, y_train_final), axis = 1)
train_final_data.shape
```

Out[ ]:

```
(53100, 246)
```

Here we are storing the final train and test data so we can use it without repeating the all process

In [ ]:

```
test_final_data = pd.concat((X_test_final, Y_test), axis = 1)
test_final_data.shape
```

Out[ ]:

```
(16000, 246)
```

In [ ]:

```
train_final_data.to_csv('/content/drive/MyDrive/train_final_data')
test_final_data.to_csv('/content/drive/MyDrive/test_final_data')
```

Model

Here we are going to applying the Machine Learning Algorithm on our preprocessing dataset

In [ ]:

```
train_data = pd.read_csv('/content/drive/MyDrive/train_final_data')
test_data = pd.read_csv('/content/drive/MyDrive/test_final_data')
```

In [ ]:

```
Y_train = train_data['class']
X_train = train_data.drop('class', axis = 1)

Y_test = test_data['class']
X_test = test_data.drop('class', axis = 1)
```

In [ ]:

```
print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)
```

```
(53100, 246) (53100,)
(16000, 246) (16000,)
```

In [ ]:

```
# ploting the confusion matrics
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(x, y):
  cm = confusion_matrix(x, y)
  A =(((cm.T)/(cm.sum(axis=1))).T)
  B =(cm/cm.sum(axis=0))
  labels = [0,1]
  plt.figure(figsize=(20,5))
  plt.subplot(1,3,1);
  sns.heatmap(cm, annot=True, fmt=".3f",cmap='Blues',xticklabels=labels, yticklabels=lab
els)

  # Ploting Confusion Matrix
  plt.xlabel('Predicted labels')
  plt.ylabel('True labels')
  plt.title('Confusion Matrix')

  # ploting Precision Matrix
  plt.subplot(1,3,2)
  sns.heatmap(B, annot=True, fmt=".3f",cmap='Blues',xticklabels=labels, yticklabels=labe
ls)
  plt.xlabel('Predicted labels')
  plt.ylabel('True labels')
  plt.title('Precision Matrix')

  # ploting Recall Matrix
  ax = plt.subplot(1,3,3)
  sns.heatmap(A, annot=True, fmt=".3f",cmap='Blues',xticklabels=labels, yticklabels=labe
ls)
  plt.xlabel('Predicted labels')
  plt.ylabel('True labels')
  plt.title('Recall Matrix')

  # here we are ploting False negetive, False positive and Total cost
  print("*"*50)
  print("False Positive:", cm[0][1])
  print("False Negative:", cm[1][0])
  print("Total cost:", cm[0][1] * 10 + cm[1][0] * 500)
  print("*"*50)
```

In [ ]:

```
# this function return the classification report
from sklearn.metrics import classification_report
def Classification_report(x, y):
    #y_pred = model.predict(X_test)
```

```
    print(classification_report(Y_test, y_pred))
```

this is the confusion matrix function for calculating the cost and ploting the confusion matrix

Logistic Regression

In [ ]:

```
# hyperparameter tunning using GridSearchCV

from sklearn.linear_model import LogisticRegression
from sklearn import datasets,linear_model
from sklearn.model_selection import GridSearchCV

params= [{"C":[10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3]}] # parameter

clf = LogisticRegression(max_iter=300,penalty= 'l2')

model = GridSearchCV(clf,params,scoring = 'f1', cv= 5)
model.fit(X_train,Y_train)

print(model.best_estimator_)
print(model.score(X_test, Y_test))
```

```
LogisticRegression(C=1, max_iter=300)
0.6466019417475729
```

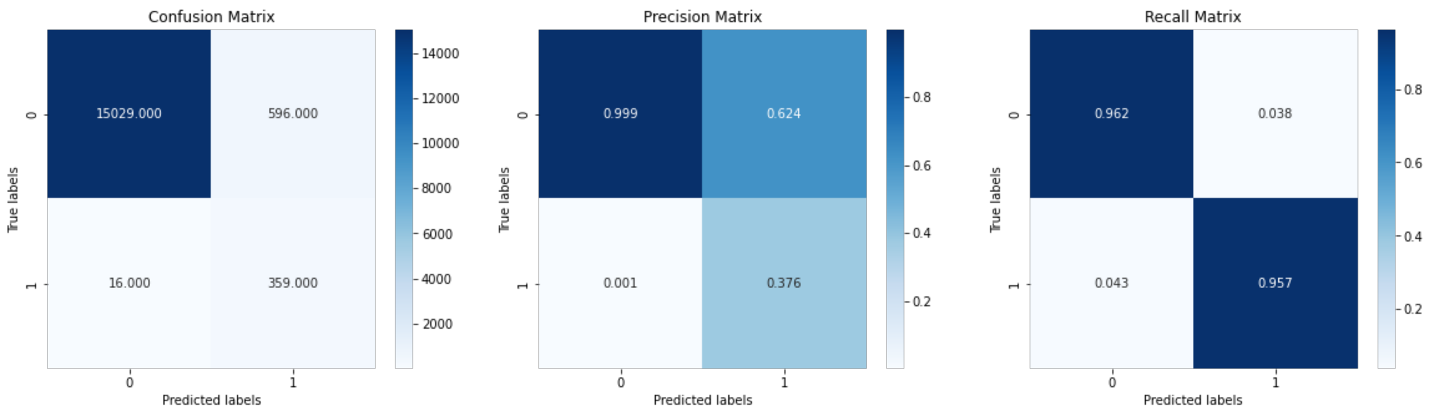
In [ ]:

```
# traing the model with best parameter
clf = LogisticRegression(n_jobs= -1,random_state=100,C= 1,penalty= 'l2')
clf.fit(X_train,Y_train)
y_pred = clf.predict(X_test) # class lable  prediction
```

In [ ]:

```
# ploting the confusion matrics
plot_confusion_matrix(Y_test, y_pred)
```

```
*************************************************
False Positive: 596
False Negative: 16
Total cost: 13960
*************************************************
```



In [ ]:

```
# classification report
Classification_report(X_test, Y_test)
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.96      0.98     15625
         1.0       0.38      0.95      0.55       375
```

```
         1.0       0.38      0.95      0.55       375

    accuracy                           0.96     16000
   macro avg       0.69      0.96      0.76     16000
weighted avg       0.98      0.96      0.97     16000
```

Logistic Regressing Model give total cost 13960

Random Forrest

In [ ]:

```python
# hyper parameter tunning for Rndom Forest model
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier()


params={'n_estimators':[5,10,50, 75, 100, 200, 300],'max_depth':[5, 10, 15, 20, 25, 30]}
# parameter

model=GridSearchCV(clf,param_grid=params,n_jobs=-1,scoring='f1',cv=5)
model.fit(X_train,Y_train)
print("Best estimator is", model.best_params_)

print(model.score(X_test, Y_test))
```

```
Best estimator is {'max_depth': 10, 'n_estimators': 75}
0.13366336633663367
```

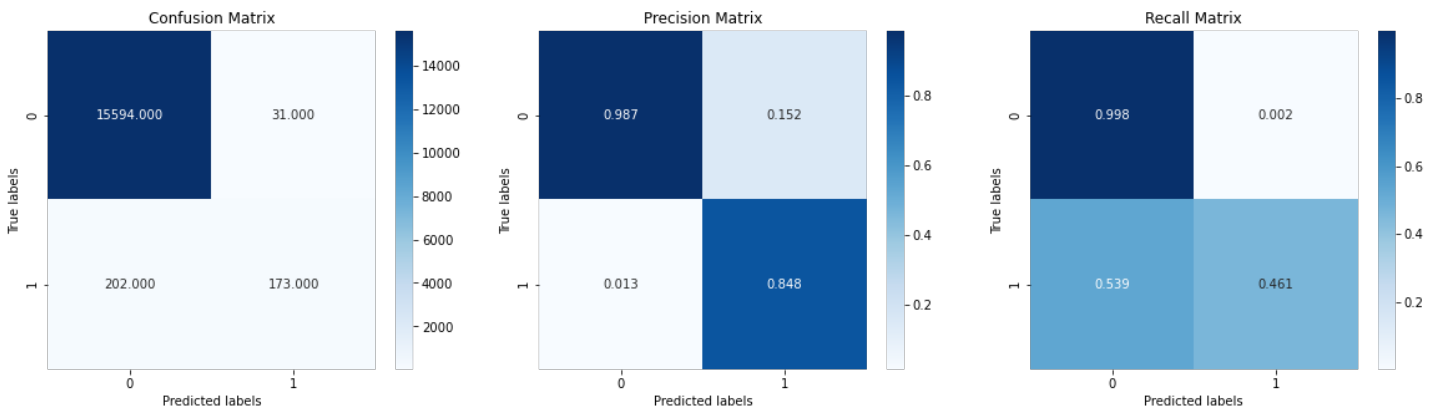
In [ ]:

```python
# traing the model using best parameter
clf = RandomForestClassifier(n_jobs= -1,random_state=42, max_depth=10 ,n_estimators= 75)
clf.fit(X_train,Y_train)
y_pred = clf.predict(X_test)
```

In [ ]:

```python
# ploting confusion matrics
plot_confusion_matrix(Y_test, y_pred)
```

```
**************************************************
False Positive: 31
False Negative: 202
Total cost: 101310
**************************************************
```



In [ ]:

```python
# classification report
Classification_report(X_test, Y_test)
```

```
              precision    recall  f1-score   support
```

```
         0.0       0.98      1.00      0.99     15625
         1.0       0.93      0.07      0.13       375

    accuracy                           0.98     16000
   macro avg       0.95      0.54      0.56     16000
weighted avg       0.98      0.98      0.97     16000
```

random model gives very high false negetive value for that Total cost is very high 134110

Decision Tree

In [ ]:

```
# Hyper parameter tunning for Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
clf=DecisionTreeClassifier()


tuned_parameters={'max_depth':[5,10,15,20,25]} # parameter

model=GridSearchCV(clf,param_grid=tuned_parameters,n_jobs=-1,scoring='f1',cv=5)
model.fit(X_train,Y_train)
print("Best estimator is", model.best_params_)
print(model.score(X_test, Y_test))
```

```
Best estimator is {'max_depth': 5}
0.0
```

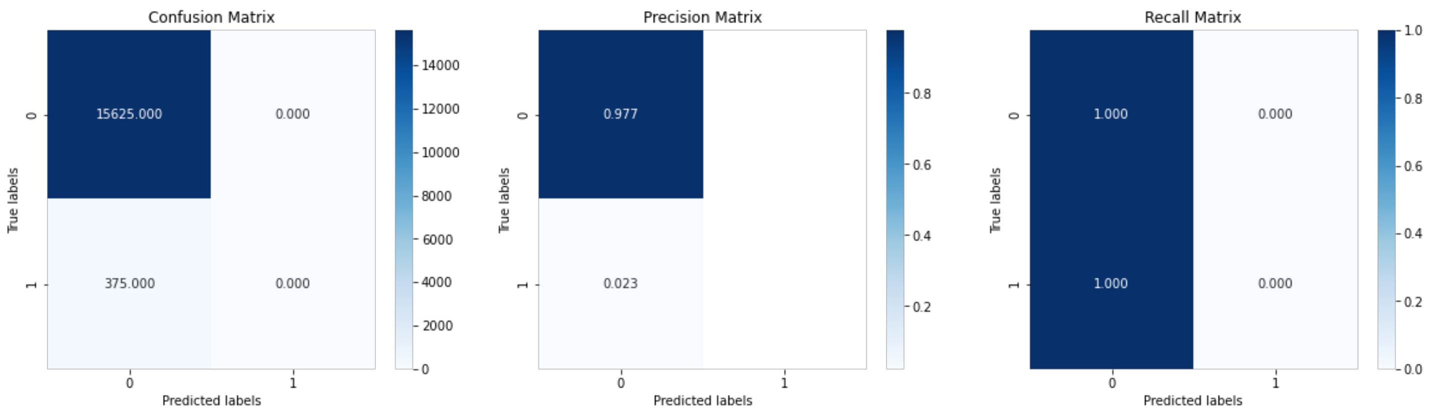
In [ ]:

```
# training model with best parameter
clf = DecisionTreeClassifier(random_state=42,max_depth=5)
clf.fit(X_train,Y_train)
y_pred = clf.predict(X_test)
```

In [ ]:

```
# ploting confusion matrics
plot_confusion_matrix(Y_test, y_pred)
```

```
**************************************************
False Positive: 0
False Negative: 375
Total cost: 187500
**************************************************
```



In [ ]:

```
# classification matrics
Classification_report(X_test, Y_test)
```

```
              precision    recall  f1-score   support
```

```
         0.0       0.98      1.00      0.99     15625
         1.0       0.00      0.00      0.00       375

    accuracy                           0.98     16000
   macro avg       0.49      0.50      0.49     16000
weighted avg       0.95      0.98      0.96     16000
```

Decision Tree has given False Positive 0 but False Negetive is 375 for that total cost is 187500

XGBOOST

In [ ]:

```
# Hyperparametr tunning for XGBoost classifier
from xgboost import XGBClassifier

clf=XGBClassifier()
params = {'n_estimators':[200,300,500,800],'max_depth':[3,5,10]} # parameter

model=GridSearchCV(clf, params,scoring='f1',cv=5)
model.fit(X_train,Y_train)
print("Best estimator is", model.best_params_)
```

```
[19:28:17] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:28:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:29:10] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:29:43] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:30:19] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:30:56] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:31:40] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:32:27] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:33:12] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:33:57] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:34:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:35:49] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
```

```
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:36:58] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:38:09] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:39:17] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:40:21] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:41:56] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:43:30] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:45:04] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:46:44] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:48:20] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:48:57] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:49:31] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:50:07] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:50:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:51:16] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:51:59] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:52:36] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:53:17] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:54:03] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
```

```
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:54:49] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:55:56] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:57:03] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:58:07] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[19:59:12] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:00:15] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:01:53] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:03:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:04:55] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:05:46] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:06:37] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:06:55] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:07:14] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:07:33] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:07:51] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:08:11] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:08:39] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:09:06] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
```

```
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:09:32] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:09:58] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:10:23] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:10:59] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:11:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:12:11] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:12:47] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:13:25] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:14:20] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:15:15] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:16:07] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:16:57] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
[20:17:48] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
Best estimator is {'max_depth': 3, 'n_estimators': 200}
```

In [ ]:

```
# training model with best parameter
clf = XGBClassifier(n_jobs= -1,random_state=42,max_depth=3,n_estimators= 200)
clf.fit(X_train,Y_train)
y_pred = clf.predict(X_test)
```

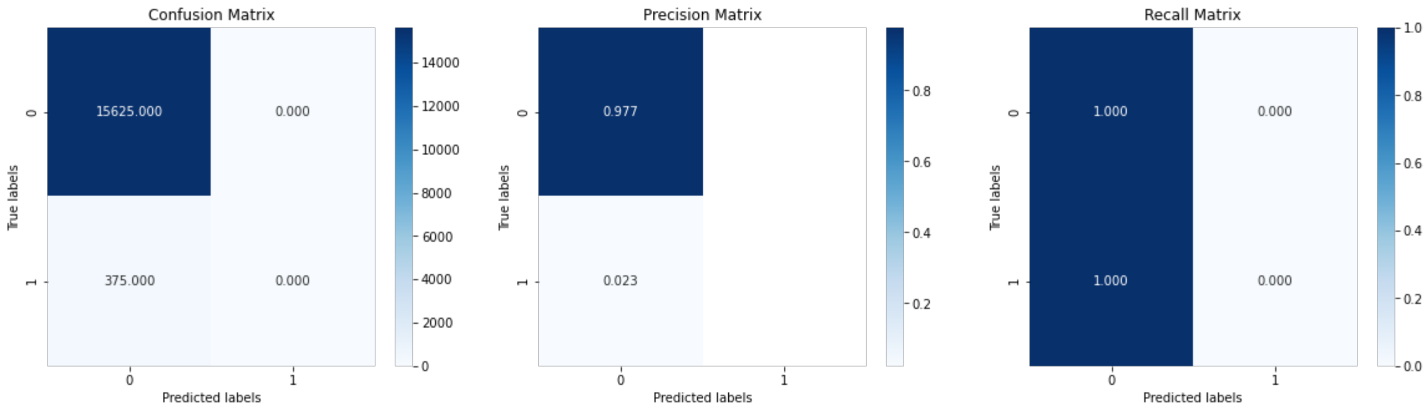
```
[20:29:08] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lear
ner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objec
tive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
```

In [ ]:

```
# ploting confusion matrics
```

```
plot_confusion_matrix(Y_test, y_pred)
```

```
**************************************************
False Positive: 0
False Negative: 375
Total cost: 187500
**************************************************
```



In [ ]:

```
# classification matrics
Classification_report(X_test, Y_test)
```

```
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99     15625
         1.0       0.00      0.00      0.00       375

    accuracy                           0.98     16000
   macro avg       0.49      0.50      0.49     16000
weighted avg       0.95      0.98      0.96     16000
```

XGBOOST has given False Positive 0 but False Negetive is 375 for that total cost is 187500

Naive Bayes

In [ ]:

```
# hyperparameter tunning
clf=GaussianNB()
params = {'var_smoothing': np.random.uniform(1e-16,1e-14,100)} # parameter
model=GridSearchCV(clf, params, scoring='f1',cv=5)
model.fit(X_train,Y_train)
print("Best estimator is", model.best_params_)
```

```
Best estimator is {'var_smoothing': 1.2485655222233953e-16}
```

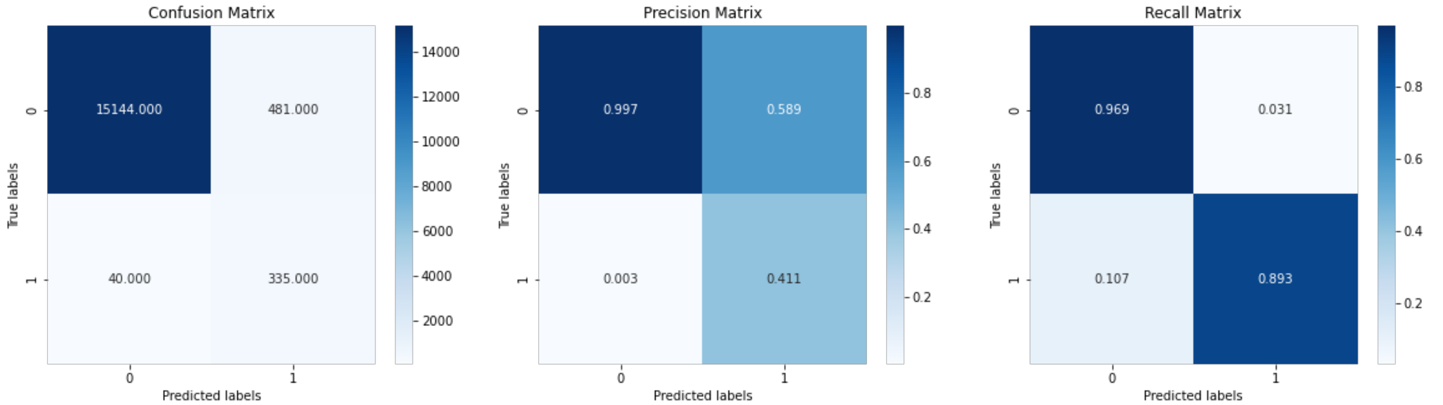
In [ ]:

```
# traing model with best parameter
clf = GaussianNB(var_smoothing = 1.2485655222233953e-16)
clf.fit(X_train,Y_train)
y_pred = clf.predict(X_test)
```

In [ ]:

```
# ploting cunfusion matrics
plot_confusion_matrix(Y_test, y_pred)
```

```
**************************************************
False Positive: 481
False Negative: 40
Total cost: 24810
**************************************************
```



In [ ]:

```
# classification matrics
Classification_report(X_test, Y_test)
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.97      0.98     15625
         1.0       0.41      0.89      0.56       375

    accuracy                           0.97     16000
   macro avg       0.70      0.93      0.77     16000
weighted avg       0.98      0.97      0.97     16000
```

SDG Classifier

In [ ]:

```
# hyper parameter tunning
from sklearn.linear_model import SGDClassifier
clf = SGDClassifier()
param = {'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10,  100]}
model = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight= 'balance', random_st
ate = 42)
model = GridSearchCV(clf, param, scoring = 'f1', cv = 5)
model.fit(X_train, Y_train)
print('Best_estimator is', model.best_params_)
```

```
Best_estimator is {'alpha': 0.001}
```

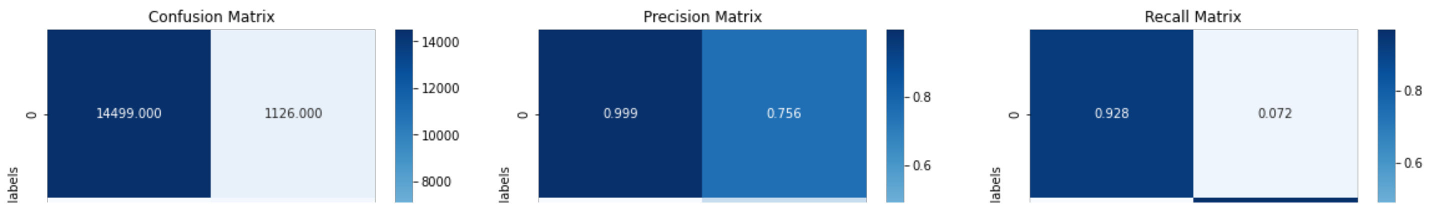
In [ ]:

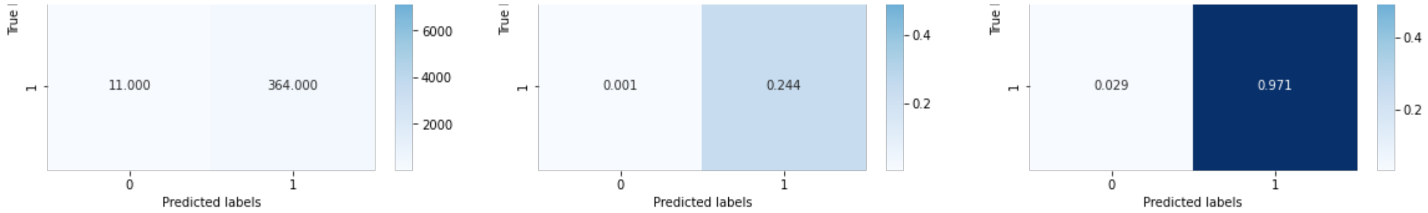
```
# traing model with best parameter
clf = SGDClassifier(alpha = 0.001, penalty = 'l2', random_state = 42)
clf.fit(X_train, Y_train)
y_pred = clf.predict(X_test)
```

In [ ]:

```
# ploting cunfusion matrics
plot_confusion_matrix(Y_test, y_pred)
```

```
****************************************************
False Positive: 1126
False Negative: 11
Total cost: 16760
****************************************************
```





In [ ]:

```
# classificatin report
Classification_report(X_test, Y_test)
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.93      0.96     15625
         1.0       0.24      0.97      0.39       375

    accuracy                           0.93     16000
   macro avg       0.62      0.95      0.68     16000
weighted avg       0.98      0.93      0.95     16000
```

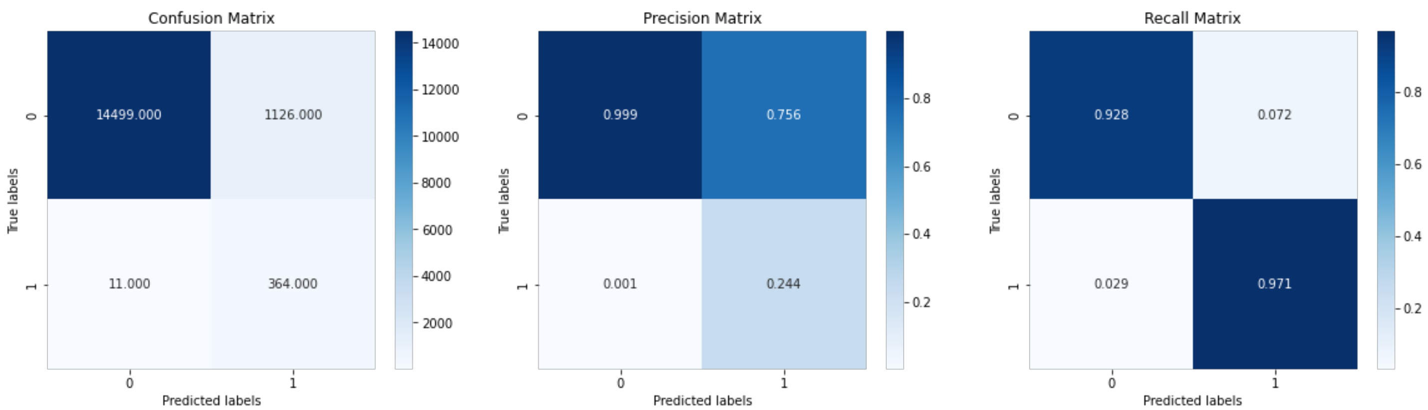
Stacking Classifier

Stacking is an ensemble learning technique that uses predictions for multiple nodes(for example GaussianNB or SVM) to build a new model. This final model is used for making predictions on the test dataset.

In [ ]:

```
from sklearn.ensemble import StackingClassifier

estimators = [('NB', GaussianNB(var_smoothing = 1.2485655222233953e-16)),
              ('SGD', SGDClassifier(alpha = 0.001, penalty = 'l2', random_state = 42))]
clf = StackingClassifier(estimators= estimators, final_estimator= LogisticRegression(n_jo
bs= -1,random_state=100,C= 1,penalty= 'l2'))
clf.fit(X_train, Y_train)
y_pred = clf.predict(X_test)
plot_confusion_matrix(Y_test, y_pred)
```

```
**************************************************
False Positive: 1126
False Negative: 11
Total cost: 16760
**************************************************
```



In [ ]:

```
Classification_report(X_test, Y_test)
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.93      0.96     15625
         1.0       0.24      0.97      0.39       375

    accuracy                           0.93     16000
```

```
   macro avg       0.62      0.95      0.68     16000
weighted avg       0.98      0.93      0.95     16000
```

In stacking Classifier we are using only three model which is Logistic Regression, GaussianNB and SGD classifier because these three models have performed best.

Pretty table

In [ ]:

```
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = [ 'Model' , 'F1 score' ,'Test Cost']
table.add_row(['Logistic Regression Model' , 0.96, 13960])
table.add_row(['Naive Bayes' , 0.97,24810])
table.add_row(['SGDClassifier', 0.68, 16760])
table.add_row(['Decision Tree Model' , 0.98,18750])
table.add_row(['Random Forest Model' , 0.98, 101310])
table.add_row(['XGBoost Model' , 0.98,187500])
table.add_row(['StackingClassifier', 0.68, 16760])
print(table)
```

```
+---------------------------+----------+-----------+
|           Model           | F1 score | Test Cost |
+---------------------------+----------+-----------+
| Logistic Regression Model |   0.96   |   13960   |
|        Naive Bayes        |   0.97   |   24810   |
|       SGDClassifier       |   0.68   |   16760   |
|    Decision Tree Model    |   0.98   |   18750   |
|    Random Forest Model    |   0.98   |   101310  |
|       XGBoost Model       |   0.98   |   187500  |
|     StackingClassifier    |   0.68   |   16760   |
+---------------------------+----------+-----------+
```