```
import pandas as pd
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

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

```
#path ="/content/drive/MyDrive/Data/creditcard.csv"
path = "./creditcard.csv"
df = pd.read_csv(path)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

```
import plotly.express as px
def plotScatterMatrix(m, dim):
  m = pd.concat([m,df[['Class']]], axis = 1)
  fig = px.scatter_matrix(m, dimensions=dim, color='Class')
  fig.update_traces(diagonal_visible=False)
  fig.show()
```

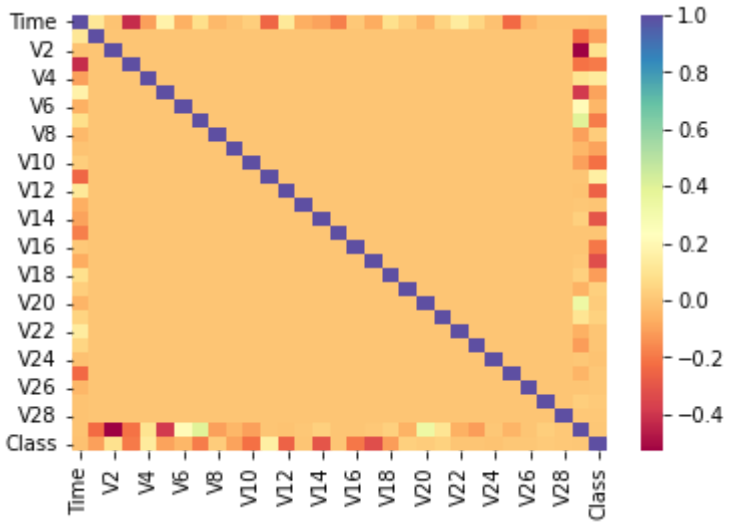
```
dimension = []
for i in range(1,29):
  j = 'V'+str(i)
  dimension.append(j)
print(dimension)
```

```
['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14
```

```
dfpca = df.drop(['Time','Class'],axis=1)
for i in range(0,len(dimension),5):
  dim = dimension[i:i+5]
  m = dfpca.iloc[:,i:i+5]
  plotScatterMatrix(m, dim)
```

```
sns.heatmap(df.corr(),cmap='Spectral',linecolor='black')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f94a935df50>
```

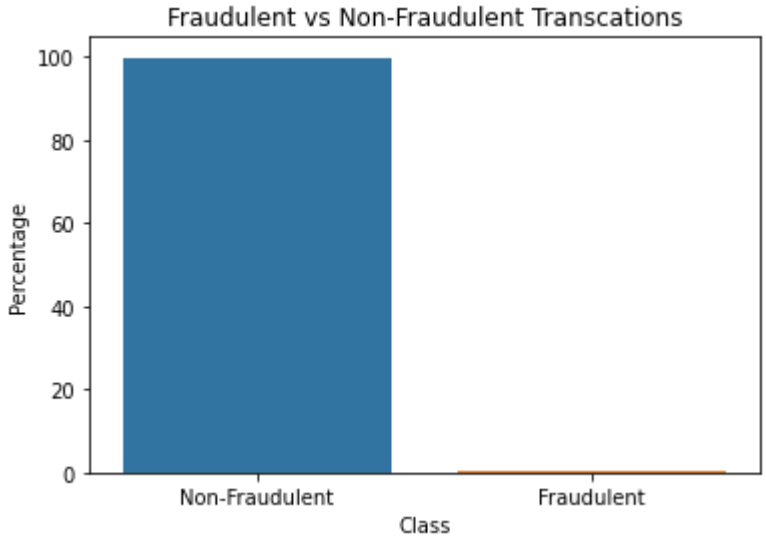



```
def analyze(df_new):
    print("Dataset Features")
    print(df_new.columns.values)
    print("=" * 50)

    print("Dataset Features Details")
    print(df_new.info())
    print("=" * 50)

  # view distribution of numerical features across the data set
    print("Dataset Numerical Features")
    print(df_new.describe())
    print("=" * 50)

analyze(df)
```

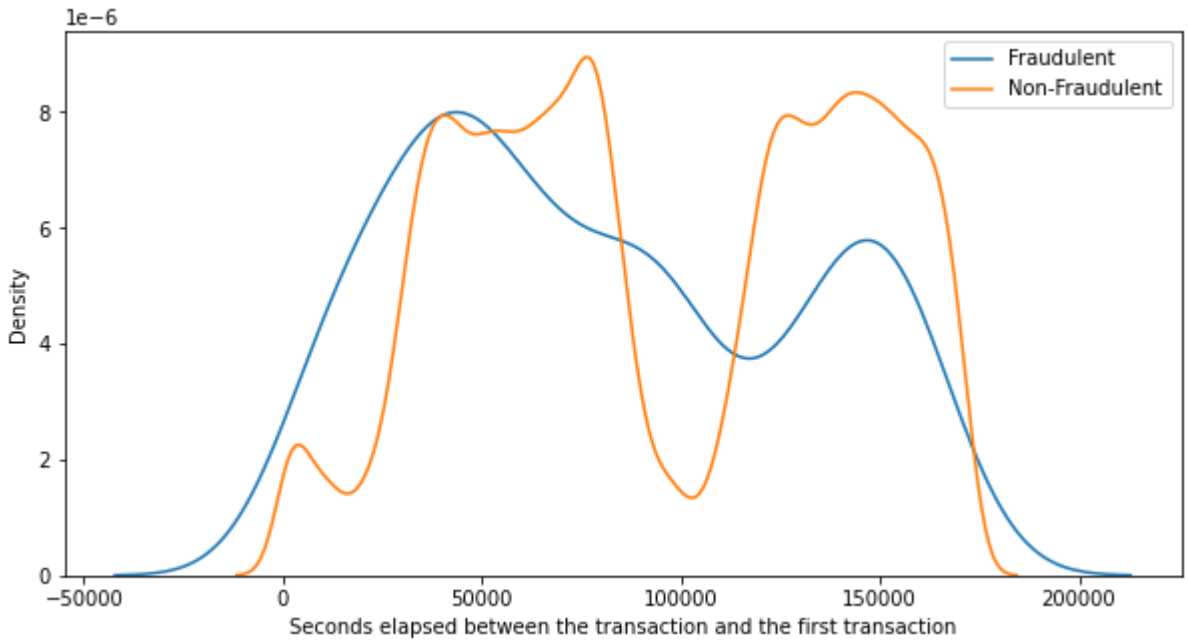
```
Dataset Features
['Time' 'V1' 'V2' 'V3' 'V4' 'V5' 'V6' 'V7' 'V8' 'V9' 'V10' 'V11' 'V12'
 'V13' 'V14' 'V15' 'V16' 'V17' 'V18' 'V19' 'V20' 'V21' 'V22' 'V23' 'V24'
 'V25' 'V26' 'V27' 'V28' 'Amount' 'Class']
==================================================
Dataset Features Details
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
None
==================================================
Dataset Numerical Features
                Time            V1  ...         Amount          Class
count  284807.000000  2.848070e+05  ...  284807.000000  284807.000000
mean    94813.859575  3.919560e-15  ...      88.349619       0.001727
std     47488.145955  1.958696e+00  ...     250.120109       0.041527
min         0.000000 -5.640751e+01  ...       0.000000       0.000000
25%     54201.500000 -9.203734e-01  ...       5.600000       0.000000
50%     84692.000000  1.810880e-02  ...      22.000000       0.000000
75%    139320.500000  1.315642e+00  ...      77.165000       0.000000
max    172792.000000  2.454930e+00  ...   25691.160000       1.000000

[8 rows x 31 columns]
==================================================
```

```
df=df.drop_duplicates()
category = df['Class'].value_counts()
```

```
normal = round((category[0]/df['Class'].count()*100),2)
fraud = round((category[1]/df['Class'].count()*100),2)

fraud_percent = {'Class':['Non-Fraudulent', 'Fraudulent'], 'Percentage':[normal, fraud]}
df_fraud_percent = pd.DataFrame(fraud_percent)
sns.barplot(x='Class',y='Percentage', data=df_fraud_percent)
plt.title('Fraudulent vs Non-Fraudulent Transcations')
plt.show()
```




```
# Creating fraudulent dataframe
fraud_data = df[df['Class'] == 1]
# Creating non fraudulent dataframe
data_non = df[df['Class'] == 0]
plt.figure(figsize=(10,5))
ax = sns.distplot(fraud_data['Time'],label='Fraudulent',hist=False)
ax = sns.distplot(data_non['Time'],label='Non-Fraudulent',hist=False)
ax.set(xlabel='Seconds elapsed between the transaction and the first transaction')
ax.legend(loc=0)
plt.show()
```




```
# Dropping the Time column to prepare data for model training
df.drop('Time', axis=1, inplace=True)
```

```
# Import library
from sklearn.model_selection import train_test_split
# X: Feature Variables
X = df.drop(['Class'], axis=1)
# Y: Target Variable
y = df['Class']
# Splitting data 80:20
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2, ran
```

```
#Feature Scaling
#We need to scale only the Amount column as all other columns are already scaled by the PCA

# Standardization method
from sklearn.preprocessing import StandardScaler
stdscaler = StandardScaler()
```

```
#Fit and transform train set
X_train['Amount'] = stdscaler.fit_transform(X_train[['Amount']])
X_train.head()
```

	V1	V2	V3	V4	V5	V6	V7	V8	
49578	-1.550810	0.731824	2.549028	2.733753	0.250049	2.870749	0.354236	0.007231	-0.037
56032	-1.305339	1.059107	1.586936	1.650134	1.669107	-0.680811	0.717706	0.072364	-1.664
272370	-1.498534	1.268183	-1.360645	-1.310035	2.744320	3.666680	0.234403	0.502133	0.313
13917	1.198304	0.378397	0.541355	0.835317	-0.386506	-0.829408	-0.137542	-0.121206	1.122
132654	-0.833194	0.769305	0.865990	-2.250385	1.018626	0.090372	0.751702	0.149752	-0.268

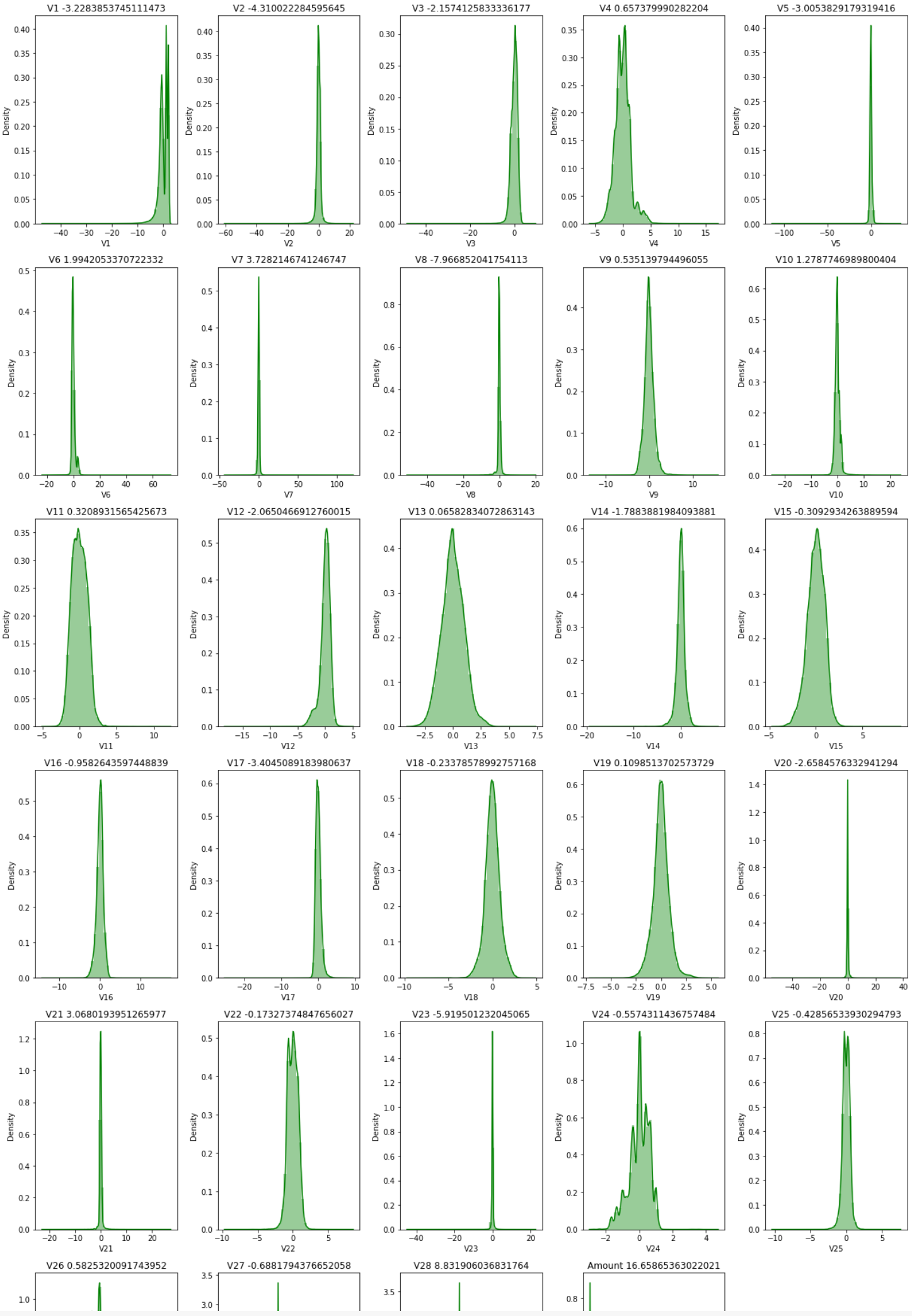
```
# Transform the test set
X_test['Amount'] = stdscaler.transform(X_test[['Amount']])
X_test.head()
```

	V1	V2	V3	V4	V5	V6	V7	V8	
174240	1.933240	-1.316851	0.425610	-0.354184	-1.854467	-0.021007	-1.602457	0.219034	0.271
277496	-1.592290	1.511862	-1.410847	-1.175748	2.867225	3.524474	0.495902	1.614884	-1.271
105198	-0.976564	0.585510	-0.003142	-0.945268	3.539277	2.916666	0.460135	0.831828	-1.095
227782	1.780293	-1.063671	0.680026	0.841410	-1.653529	0.392116	-1.514345	0.467485	2.093
116080	-1.153910	1.533617	0.434139	-0.495930	0.378126	-0.067809	0.530507	0.229062	0.300

```
features = X_test.columns
```

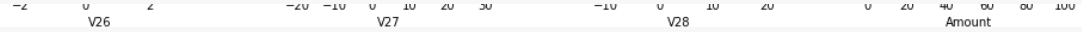
```
# Plotting the distribution of the variables (skewness)
k=0
```

```
plt.figure(figsize=(17,28))
plt.title("Distribution of all the Features")
for col in features :
    k=k+1
    plt.subplot(6, 5,k)
    sns.distplot(X_train[col], color='g')
    plt.title(col+' '+str(X_train[col].skew()))
    plt.tight_layout()
```



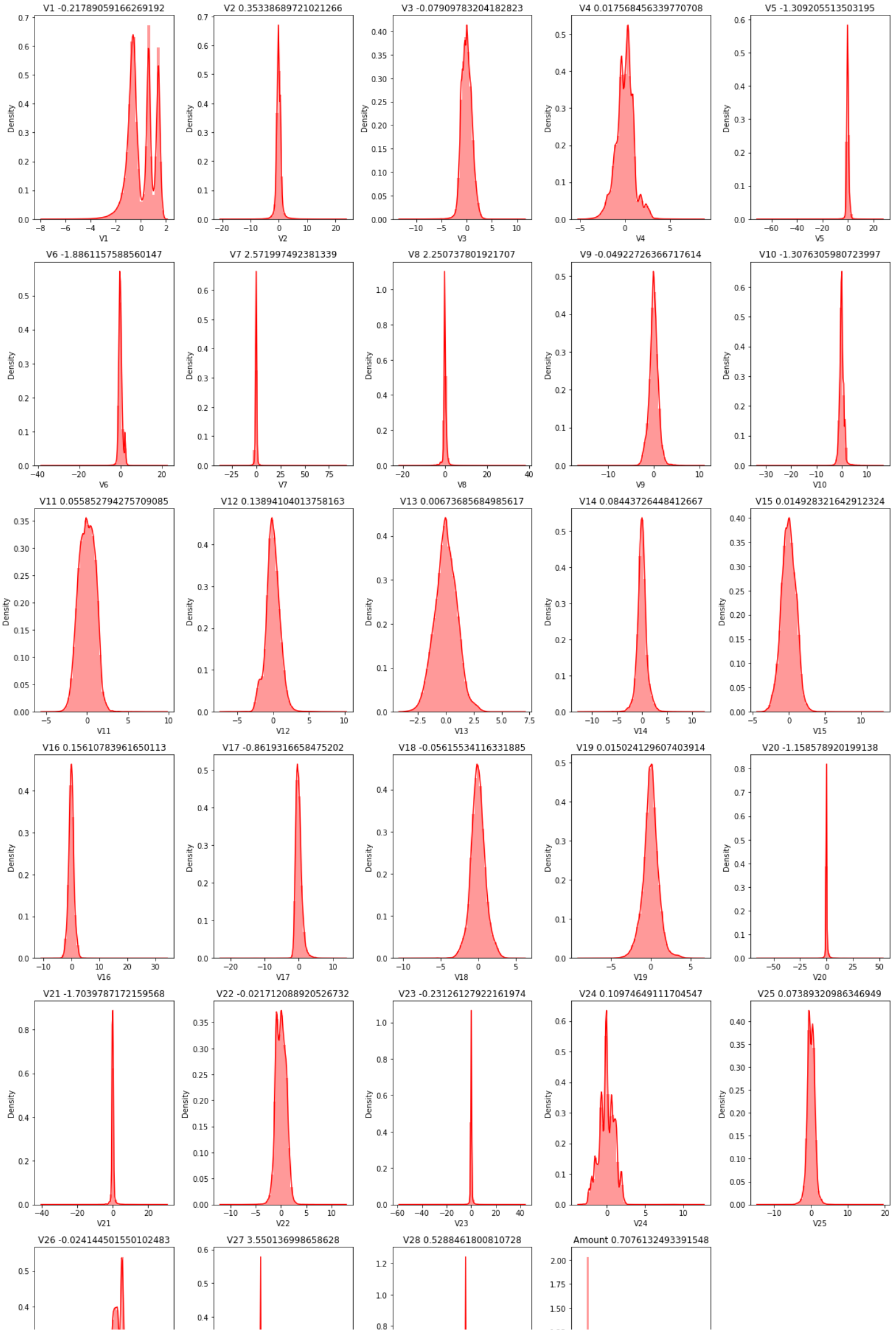

```
# Importing PowerTransformer
```

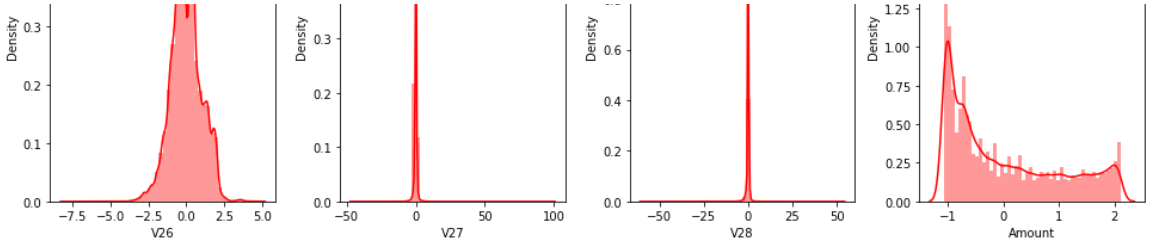
```
from sklearn.preprocessing import PowerTransformer
# Instantiate the powertransformer
pt = PowerTransformer(method='yeo-johnson', standardize=True, copy=False)
# Fit and transform the PT on training data
X_train[features] = pt.fit_transform(X_train)
```




```
# Transform the test set
X_test[features] = pt.transform(X_test)
```

```
# Plotting the distribution of the Transformed variables (skewness)
k=0
plt.figure(figsize=(17,28))
for col in features :
    k=k+1
    plt.subplot(6, 5, k)
    sns.distplot(X_train[col], color='r')
    plt.title(col+' '+str(X_train[col].skew()))
    plt.tight_layout()
```


V1 -0.21789059166269192
V2 0.35338689721021266
V3 -0.07909783204182823
V4 0.017568456339770708
V5 -1.309205513503195
V6 -1.8861157588560147
V7 2.571997492381339
V8 2.250737801921707
V9 -0.04922726366717614
V10 -1.3076305980723997
V11 0.055852794275709085
V12 0.13894104013758163
V13 0.00673685684985617
V14 0.08443726448412667
V15 0.014928321642912324
V16 0.15610783961650113
V17 -0.8619316658475202
V18 -0.05615534116331885
V19 0.015024129607403914
V20 -1.158578920199138
V21 -1.7039787172159568
V22 -0.021712088920526732
V23 -0.23126127922161974
V24 0.10974649111704547
V25 0.07389320986346949
V26 -0.024144501550102483
V27 3.550136998658628
V28 0.5288461800810728
Amount 0.7076132493391548
Density



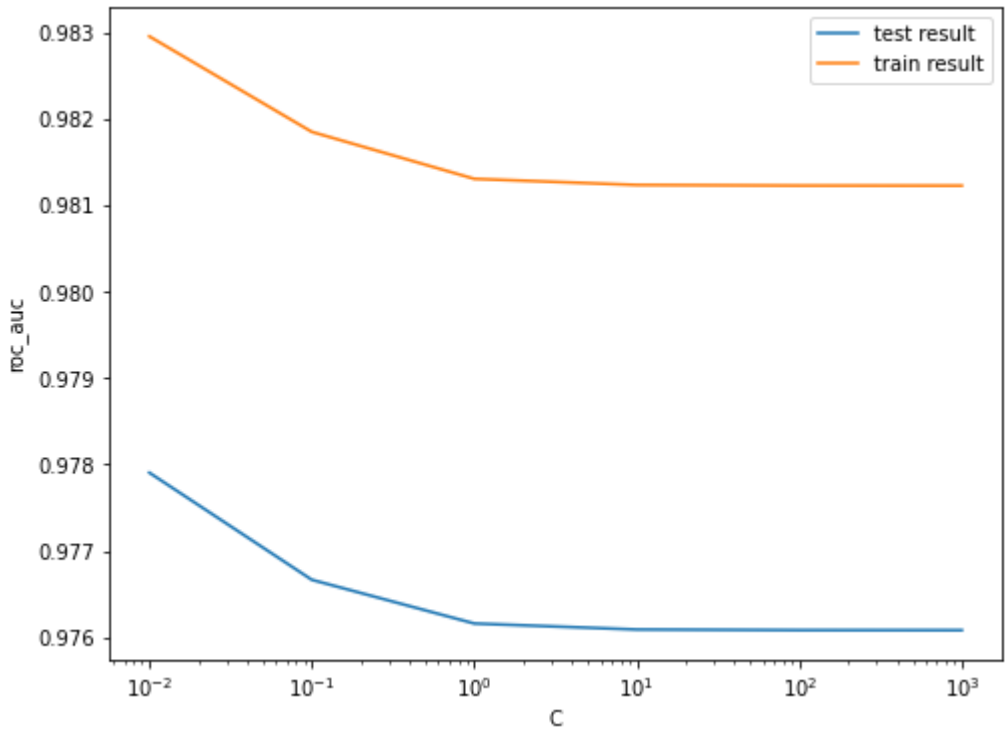
LR : Logistic Reg

```
# Importing scikit logistic regression module
from sklearn.linear_model import LogisticRegression
# Impoting metrics
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
# Importing libraries for cross validation
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

```
# Creating KFold object with 5 splits
folds = KFold(n_splits=5, shuffle=True, random_state=4)
# Specify parameters
parameters = {"C": [0.01, 0.1, 1, 10, 100, 1000]}
# Specifing score as recall as we are more focused on acheiving the higher sensitivity than
model_cv = GridSearchCV(estimator = LogisticRegression(),
                        param_grid = parameters,
                        scoring= 'roc_auc',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# Fit the model
model_cv.fit(X_train, y_train)
# results of grid search CV
cv_results = pd.DataFrame(model_cv.cv_results_)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

```
#Plotting the roc_auc
plt.figure(figsize=(8, 6))
plt.plot(cv_results['param_C'], cv_results['mean_test_score'])
plt.plot(cv_results['param_C'], cv_results['mean_train_score'])
plt.xlabel('C')
plt.ylabel('roc_auc')
plt.legend(['test result', 'train result'], loc=0)
plt.xscale('log')
```




```
# Best score with best C
best = model_cv.best_score_
best_C = model_cv.best_params_['C']
print(" The largest test roc_auc is {} for C = {}".format(best, best_C))
```

```
      The largest test roc_auc is 0.9779060493412117 for C = 0.01
```

```
def printMetrics(train,pred):
  confusion = metrics.confusion_matrix(train, pred)
  TP = confusion[1,1] # true positive
  TN = confusion[0,0] # true negatives
  FP = confusion[0,1] # false positives
  FN = confusion[1,0] # false negatives
  print("Confusion Matrix:\n{}\n".format(confusion))

  print("Accuracy: {}".format(metrics.accuracy_score(train, pred)))
  print("Sensitivity: {}".format(TP / float(TP+FN)))
  print("Specificity: {}".format(TN / float(TN+FP)))
  print("F1-Score: {}\n".format(f1_score(train, pred)))
  print(classification_report(train,pred))
```

```
def draw_roc( actual, probs ):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs, drop_intermediate = False )
    auc_score = metrics.roc_auc_score( actual, probs )
    print("auc_score: {}\n".format(auc_score))
    plt.figure(figsize=(5, 5))
    plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
```

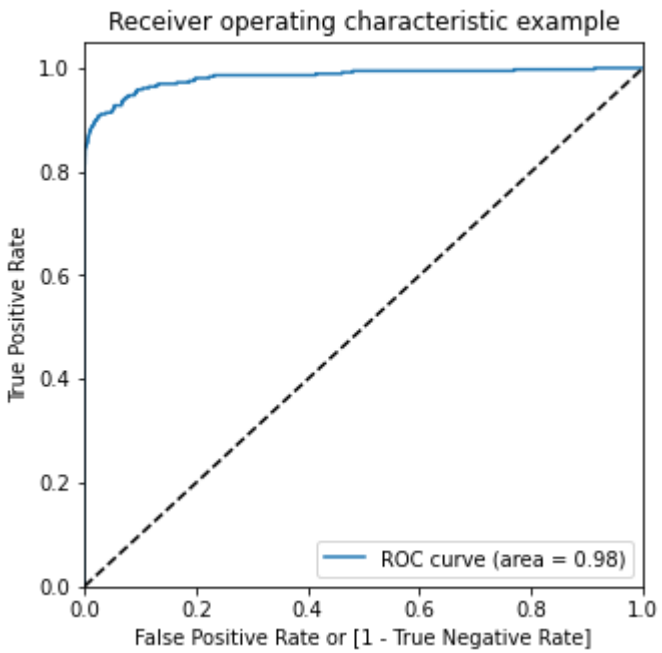
```
    plt.legend(loc="lower right")
    plt.show()
```

```
# Instantiate the model with best C
logistic_imb = LogisticRegression(C=0.01)
#Fit the model on the train set
logistic_imb_model = logistic_imb.fit(X_train, y_train)
# Predictions on the train set
y_train_pred = logistic_imb_model.predict(X_train)

printMetrics(y_train, y_train_pred)
```

```
Confusion Matrix:
[[226590     30]
 [   155    205]]

Accuracy: 0.999184950215878
Sensitivity: 0.5694444444444444
Specificity: 0.9998676198040773
F1-Score: 0.6890756302521008

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226620
           1       0.87      0.57      0.69       360

    accuracy                           1.00    226980
   macro avg       0.94      0.78      0.84    226980
weighted avg       1.00      1.00      1.00    226980
```

```
y_train_pred_proba = logistic_imb_model.predict_proba(X_train)[:,1]
draw_roc(y_train, y_train_pred_proba)
```

```
auc_score: 0.9826763108090881
```



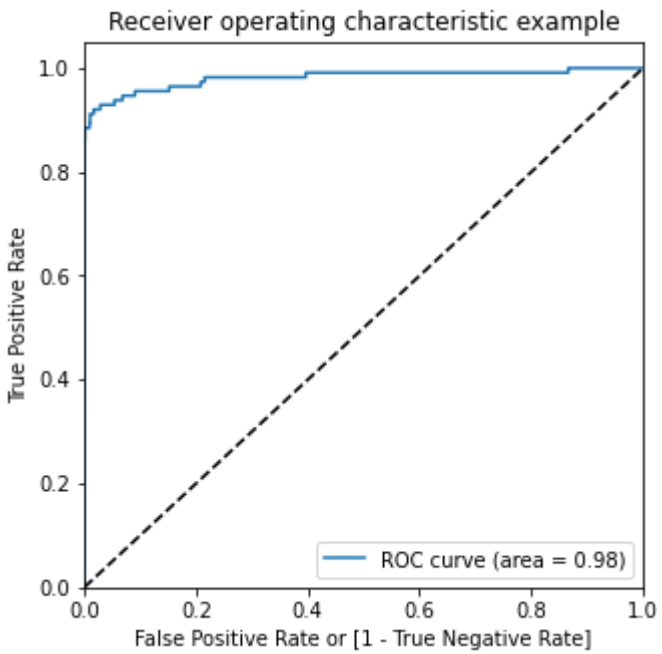

```
y_test_pred = logistic_imb_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

```
Confusion Matrix:
```

```
[[56627     6]
 [   47    66]]

Accuracy: 0.999066013463504
Sensitivity: 0.584070796460177
Specificity: 0.9998940547030883
F1-Score: 0.7135135135135136

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56633
           1       0.92      0.58      0.71       113

    accuracy                           1.00     56746
   macro avg       0.96      0.79      0.86     56746
weighted avg       1.00      1.00      1.00     56746
```

```
# Predicted probability
y_test_pred_proba = logistic_imb_model.predict_proba(X_test)[:,1]
draw_roc(y_test, y_test_pred_proba)
```

```
auc_score: 0.9810565746322892
```




```
# X
from xgboost import XGBClassifier
```

```
# creating a KFold object
folds = 3

# specify range of hyperparameters
param_grid = {'learning_rate': [0.2, 0.6],
              'subsample': [0.3, 0.6, 0.9]}

# specify model
xgb_model = XGBClassifier(max_depth=2, n_estimators=200)

# set up GridSearchCV()
```

```
model_cv = GridSearchCV(estimator = xgb_model,
                        param_grid = param_grid,
                        scoring= 'roc_auc',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)
# fit the model
model_cv.fit(X_train, y_train)
```

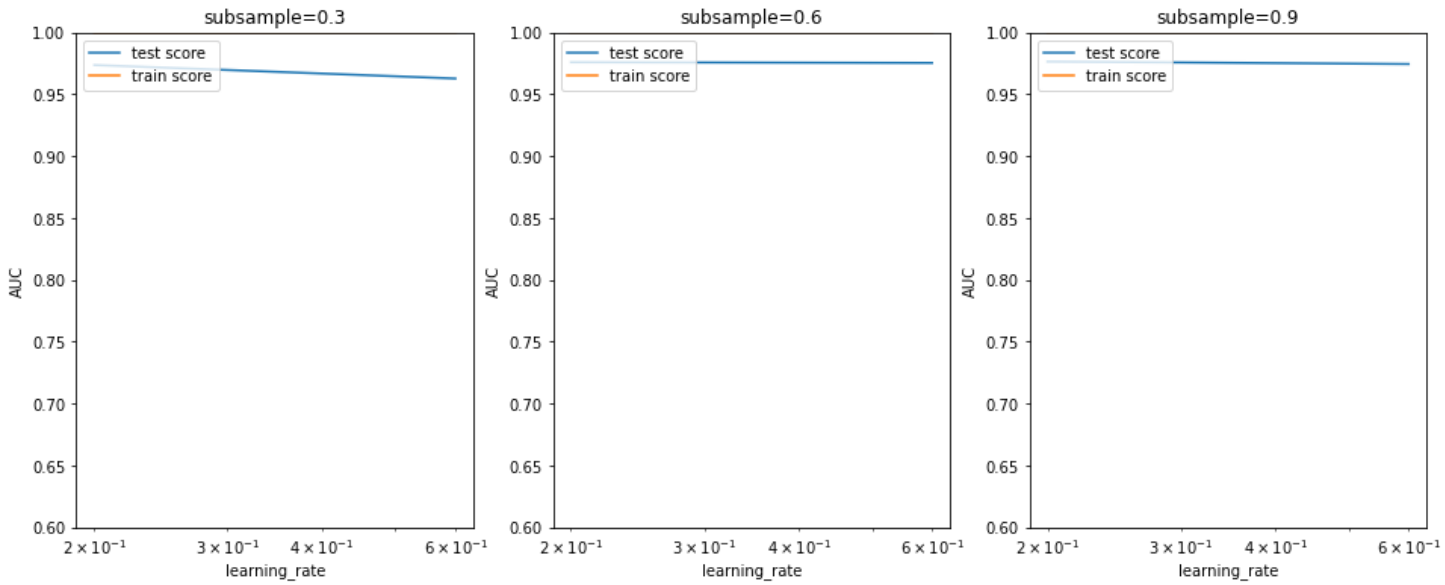
```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
GridSearchCV(cv=3, estimator=XGBClassifier(max_depth=2, n_estimators=200),
             param_grid={'learning_rate': [0.2, 0.6],
                         'subsample': [0.3, 0.6, 0.9]},
             return_train_score=True, scoring='roc_auc', verbose=1)
```

```
# cv results
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	p
0	37.013414	0.123609	0.405644	0.000808	0.2	
1	47.112583	0.123419	0.405348	0.002198	0.2	
2	48.426478	0.241868	0.402455	0.004484	0.2	
3	36.450557	0.111351	0.418051	0.003915	0.6	
4	46.134787	0.101257	0.418219	0.001137	0.6	
5	47.734525	0.134308	0.421771	0.001251	0.6	

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

param_grid = {'learning_rate': [0.2, 0.6],
              'subsample': [0.3, 0.6, 0.9]}
```

```
for n, subsample in enumerate(param_grid['subsample']):


    # subplot 1/n
    plt.subplot(1,len(param_grid['subsample']), n+1)
    df = cv_results[cv_results['param_subsample']==subsample]

    plt.plot(df["param_learning_rate"], df["mean_test_score"])
    plt.plot(df["param_learning_rate"], df["mean_train_score"])
    plt.xlabel('learning_rate')
    plt.ylabel('AUC')
    plt.title("subsample={0}".format(subsample))
    plt.ylim([0.60, 1])
    plt.legend(['test score', 'train score'], loc='upper left')
    plt.xscale('log')
```




```
model_cv.best_params_
```

```
{'learning_rate': 0.2, 'subsample': 0.9}
```

```
# chosen hyperparameters
# 'objective':'binary:logistic' outputs probability rather than label, which we need for cal
parameters = {'learning_rate': 0.2,
          'max_depth': 2,
          'n_estimators':200,
          'subsample':0.9,
         'objective':'binary:logistic'}

# fit model on training data
xgb_imb_model = XGBClassifier(params = parameters)
xgb_imb_model.fit(X_train, y_train)
```

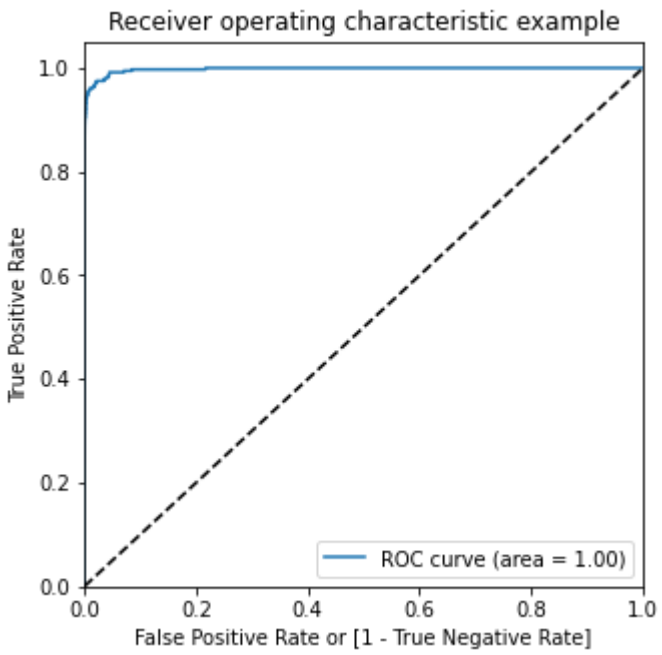
```
XGBClassifier(params={'learning_rate': 0.2, 'max_depth': 2, 'n_estimators': 200,
                      'objective': 'binary:logistic', 'subsample': 0.9})
```

```
# Predictions on the train set
y_train_pred = xgb_imb_model.predict(X_train)
printMetrics(y_train,y_train_pred)
```

```
Confusion Matrix:
[[226611      9]
 [    65    295]]

Accuracy: 0.9996739800863512
Sensitivity: 0.8194444444444444
Specificity: 0.9999602859412232
F1-Score: 0.8885542168674699

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226620
           1       0.97      0.82      0.89       360

    accuracy                           1.00    226980
   macro avg       0.99      0.91      0.94    226980
weighted avg       1.00      1.00      1.00    226980
```

```
# Predicted probability
y_train_pred_proba_imb_xgb = xgb_imb_model.predict_proba(X_train)[:,1]
draw_roc(y_train, y_train_pred_proba_imb_xgb)
```

```
auc_score: 0.9977183353925808
```




```
# Predictions on the test set
y_test_pred = xgb_imb_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

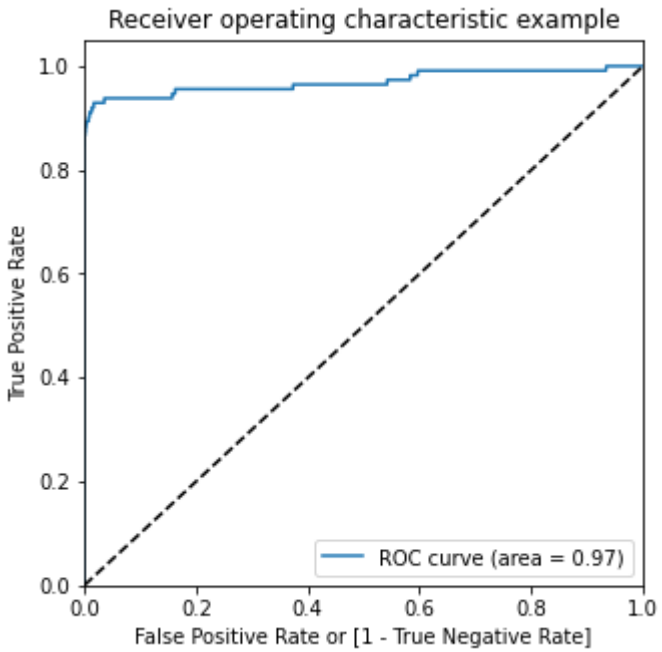
```
Confusion Matrix:
[[56627     6]
 [   19    94]]
```

```
Accuracy: 0.9995594403129736
Sensitivity: 0.831858407079646
Specificity: 0.9998940547030883
F1-Score: 0.8826291079812206
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56633
1	0.94	0.83	0.88	113
accuracy			1.00	56746
macro avg	0.97	0.92	0.94	56746
weighted avg	1.00	1.00	1.00	56746

```
# Predicted probability
y_test_pred_proba = xgb_imb_model.predict_proba(X_test)[:,1]
draw_roc(y_test, y_test_pred_proba)
```

```
auc_score: 0.9695336953703937
```




```
# Decision tree
# Importing decision tree classifier
from sklearn.tree import DecisionTreeClassifier
```

```
# Create the parameter grid
param_grid = {
    'max_depth': range(5, 15, 5),
    'min_samples_leaf': range(50, 150, 50),
    'min_samples_split': range(50, 150, 50),
}


# Instantiate the grid search model
dtree = DecisionTreeClassifier()

grid_search = GridSearchCV(estimator = dtree,
                           param_grid = param_grid,
```

```
                           scoring= 'roc_auc',
                           cv = 3,
                           verbose = 1)

# Fit the grid search to the data
grid_search.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': range(5, 15, 5),
                         'min_samples_leaf': range(50, 150, 50),
                         'min_samples_split': range(50, 150, 50)},
             scoring='roc_auc', verbose=1)
```

```
# cv results
cv_results = pd.DataFrame(grid_search.cv_results_)
cv_results
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param
0	4.102946	0.027701	0.028627	0.000448	5	
1	4.084292	0.018965	0.028334	0.000396	5	
2	4.071128	0.002395	0.029098	0.000632	5	
3	4.103142	0.018678	0.029979	0.001629	5	
4	8.094975	0.037446	0.031497	0.000850	10	
5	8.079125	0.092536	0.031380	0.000461	10	
6	8.016101	0.037240	0.031786	0.000886	10	
7	8.033376	0.043462	0.031881	0.001135	10	

```
# Printing the optimal sensitivity score and hyperparameters
print("Best roc_auc:-", grid_search.best_score_)
print(grid_search.best_estimator_)
```

```
Best roc_auc:- 0.9328336348660017
DecisionTreeClassifier(max_depth=10, min_samples_leaf=100,
                       min_samples_split=100)
```

```
# Model with optimal hyperparameters
dt_imb_model = DecisionTreeClassifier(criterion = "gini",
                                      random_state = 100,
                                      max_depth=5,
                                      min_samples_leaf=100,
                                      min_samples_split=100)

dt_imb_model.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=5, min_samples_leaf=100, min_samples_split=100,
                       random_state=100)
```

```
# Predictions on the train set
y_train_pred = dt_imb_model.predict(X_train)
printMetrics(y_train,y_train_pred)
```

```
Confusion Matrix:
[[226501    119]
 [    65    295]]

Accuracy: 0.9991893558903868
Sensitivity: 0.8194444444444444
Specificity: 0.9994748918895067
F1-Score: 0.7622739018087856

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226620
           1       0.71      0.82      0.76       360

    accuracy                           1.00    226980
   macro avg       0.86      0.91      0.88    226980
weighted avg       1.00      1.00      1.00    226980
```

```
# Predicted probability
y_train_pred_proba = dt_imb_model.predict_proba(X_train)[:,1]
draw_roc(y_train, y_train_pred_proba)
```

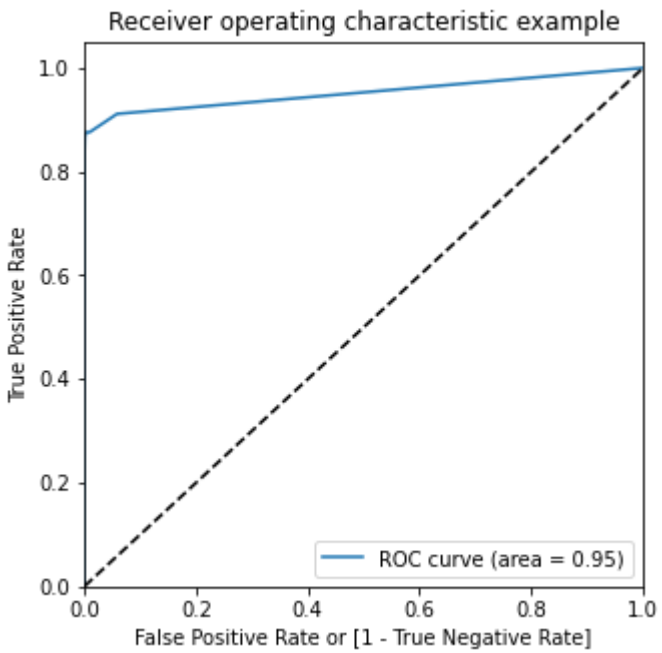
```
auc_score: 0.9541318800929601
```

```
# Predictions on the test set
y_test_pred = dt_imb_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

```
Confusion Matrix:
[[56599    34]
 [   18    95]]

Accuracy: 0.9990836358509851
Sensitivity: 0.8407079646017699
Specificity: 0.9993996433175004
F1-Score: 0.7851239669421488

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56633
           1       0.74      0.84      0.79       113

    accuracy                           1.00     56746
   macro avg       0.87      0.92      0.89     56746
weighted avg       1.00      1.00      1.00     56746
```

```
# Predicted probability
y_test_pred_proba = dt_imb_model.predict_proba(X_test)[:,1]
draw_roc(y_test,y_test_pred_proba)
```

```
auc_score: 0.9517775448786934
```



Random Forest

```
# Importing random forest classifier
from sklearn.ensemble import RandomForestClassifier
```

```
param_grid = {
    'max_depth': range(5,10,5),
    'min_samples_leaf': range(50, 150, 50),
    'min_samples_split': range(50, 150, 50),
    'n_estimators': [100,200,300],
    'max_features': [10, 20]
}
# Create a based model
rf = RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf,
                           param_grid = param_grid,
                           cv = 2,
                           n_jobs = -1,
                           verbose = 1,
                           return_train_score=True)

# Fit the model
grid_search.fit(X_train, y_train)
```

```
Fitting 2 folds for each of 24 candidates, totalling 48 fits
GridSearchCV(cv=2, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'max_depth': range(5, 10, 5), 'max_features': [10, 20],
                         'min_samples_leaf': range(50, 150, 50),
                         'min_samples_split': range(50, 150, 50),
                         'n_estimators': [100, 200, 300]},
             return_train_score=True, verbose=1)
```

```
# printing the optimal accuracy score and hyperparameters
print('We can get accuracy of',grid_search.best_score_,'using',grid_search.best_params_)
```

```
We can get accuracy of 0.9992113842629307 using {'max_depth': 5, 'max_features': 20, 'm
```

```
# model with the best hyperparameters

rfc_imb_model = RandomForestClassifier(bootstrap=True,
                               max_depth=5,
                               min_samples_leaf=50,
                               min_samples_split=50,
                               max_features=10,
                               n_estimators=100)
```

```
# Fit the model
rfc_imb_model.fit(X_train, y_train)
```

```
RandomForestClassifier(max_depth=5, max_features=10, min_samples_leaf=50,
                       min_samples_split=50)
```

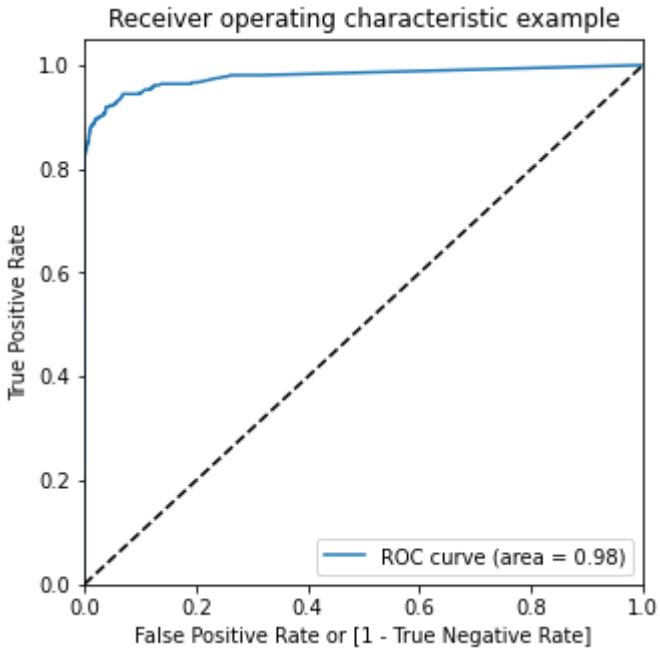
```
# Predictions on the train set
y_train_pred = rfc_imb_model.predict(X_train)
printMetrics(y_train,y_train_pred)
```

```
Confusion Matrix:
[[226575     45]
 [   102    258]]
```

```
Accuracy: 0.9993523658472112
Sensitivity: 0.7166666666666667
Specificity: 0.999801429706116
F1-Score: 0.7782805429864253

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226620
           1       0.85      0.72      0.78       360

    accuracy                           1.00    226980
   macro avg       0.93      0.86      0.89    226980
weighted avg       1.00      1.00      1.00    226980
```

```
# Predicted probability
y_train_pred_proba = rfc_imb_model.predict_proba(X_train)[:,1]
draw_roc(y_train, y_train_pred_proba)
```

```
auc_score: 0.9782301319389287
```



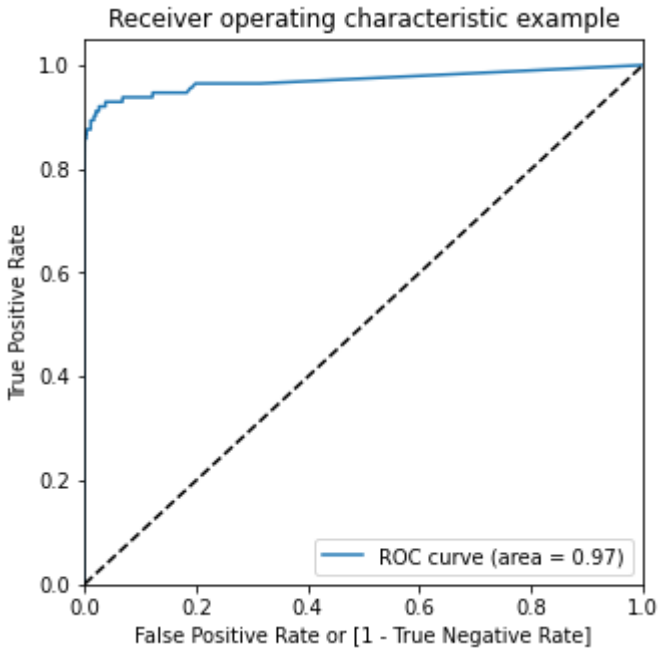

```
# Predictions on the test set
y_test_pred = rfc_imb_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

```
Confusion Matrix:
[[56616    17]
 [   27    86]]

Accuracy: 0.9992246149508336
Sensitivity: 0.7610619469026548
Specificity: 0.9996998216587502
F1-Score: 0.7962962962962963

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56633
           1       0.83      0.76      0.80       113
```

```
          accuracy                           1.00     56746
         macro avg       0.92      0.88      0.90     56746
      weighted avg       1.00      1.00      1.00     56746
```

```
# Predicted probability
y_test_pred_proba = rfc_imb_model.predict_proba(X_test)[:,1]
draw_roc(y_test, y_test_pred_proba)
```

```
auc_score: 0.9704572789653738
```



Choosing best model on the imbalanced data

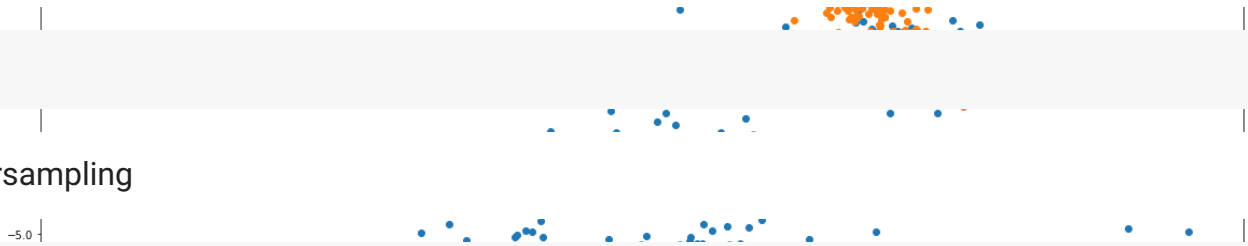
```
# Features of XGBoost model

var_imp = []
for i in xgb_imb_model.feature_importances_:
    var_imp.append(i)
print('Top var =', var_imp.index(np.sort(xgb_imb_model.feature_importances_)[-1])+1)
print('2nd Top var =', var_imp.index(np.sort(xgb_imb_model.feature_importances_)[-2])+1)
print('3rd Top var =', var_imp.index(np.sort(xgb_imb_model.feature_importances_)[-3])+1)
# Variable on Index-16 and Index-13 seems to be the top 2 variables
top_var_index = var_imp.index(np.sort(xgb_imb_model.feature_importances_)[-1])
second_top_var_index = var_imp.index(np.sort(xgb_imb_model.feature_importances_)[-2])

X_train_1 = X_train.to_numpy()[np.where(y_train==1.0)]
X_train_0 = X_train.to_numpy()[np.where(y_train==0.0)]

np.random.shuffle(X_train_0)

import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = [20, 20]

plt.scatter(X_train_1[:, top_var_index], X_train_1[:, second_top_var_index], label='Actual C
plt.scatter(X_train_0[:X_train_1.shape[0], top_var_index], X_train_0[:X_train_1.shape[0], se
```

```
            label='Actual Class-0 Examples')
plt.legend()
```

```
Top var = 17
2nd Top var = 14
3rd Top var = 10
<matplotlib.legend.Legend at 0x7f94ac8fbbd0>
```


```
print('Train auc =', metrics.roc_auc_score(y_train, y_train_pred_proba_imb_xgb))
fpr, tpr, thresholds = metrics.roc_curve(y_train, y_train_pred_proba_imb_xgb)
threshold = thresholds[np.argmax(tpr-fpr)]
print("Threshold=",threshold)
```

```
Train auc = 0.9977183353925808
Threshold= 0.0016673726
```

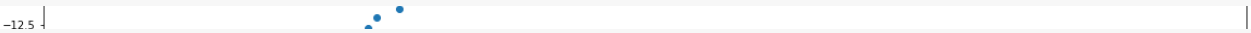


Undersampling


```
# Importing undersampler library
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter
```



```
# instantiating the random undersampler
rus = RandomUnderSampler()
# resampling X, y
X_train_rus, y_train_rus = rus.fit_resample(X_train, y_train)
```




```
# Befor sampling class distribution
print('Before sampling class distribution:-',Counter(y_train))
# new class distribution
print('New class distribution:-',Counter(y_train_rus))
```

```
Before sampling class distribution:- Counter({0: 226620, 1: 360})
New class distribution:- Counter({0: 360, 1: 360})
```

```
# Creating KFold object with 5 splits
folds = KFold(n_splits=5, shuffle=True, random_state=4)

# Specify params
parameters = {"C": [0.01, 0.1, 1, 10, 100, 1000]}

# Specifing score as roc-auc
model_cv = GridSearchCV(estimator = LogisticRegression(),
                        param_grid = parameters,
                        scoring= 'roc_auc',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)
```

```
# Fit the model
model_cv.fit(X_train_rus, y_train_rus)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
GridSearchCV(cv=KFold(n_splits=5, random_state=4, shuffle=True),
             estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1, 10, 100, 1000]},
             return_train_score=True, scoring='roc_auc', verbose=1)
```

```
# results of grid search CV
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	pa
0	0.018558	0.005048	0.006162	0.001503	0.01	
1	0.019692	0.005959	0.005264	0.000218	0.1	
2	0.022176	0.005184	0.005240	0.000674	1	{
3	0.030039	0.001989	0.005087	0.000242	10	{'C
4	0.051937	0.006954	0.005518	0.000453	100	
5	0.048336	0.008135	0.005305	0.000205	1000	

```
# plot of C versus train and validation scores

plt.figure(figsize=(8, 6))
plt.plot(cv_results['param_C'], cv_results['mean_test_score'])
plt.plot(cv_results['param_C'], cv_results['mean_train_score'])
plt.xlabel('C')
plt.ylabel('roc_auc')
plt.legend(['test result', 'train result'], loc='upper left')
plt.xscale('log')
```


```
# Best score with best C
best_score = model_cv.best_score_
best_C = model_cv.best_params_['C']

print(" The highest test roc_auc is {0} at C = {1}".format(best_score, best_C))
```

```
 The highest test roc_auc is 0.9774708115127192 at C = 0.01
```

```
# Instantiate the model with best C
logistic_bal_rus = LogisticRegression(C=0.1)
```

```
# Fit the model on the train set
logistic_bal_rus_model = logistic_bal_rus.fit(X_train_rus, y_train_rus)
```

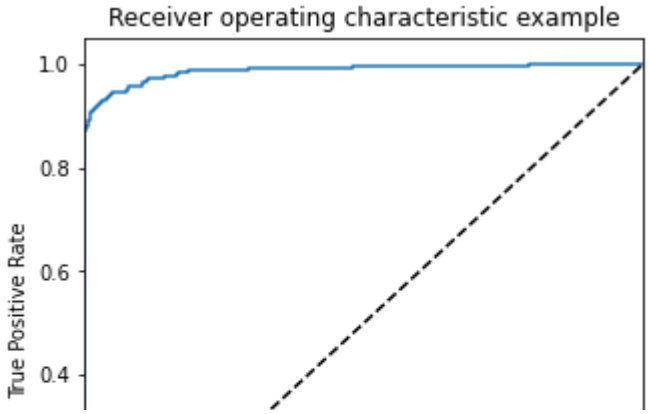
Train Set

```
# Predictions on the train set
y_train_pred = logistic_bal_rus_model.predict(X_train_rus)
printMetrics(y_train_rus,y_train_pred)
```

```
Confusion Matrix:
[[355   5]
 [ 33 327]]

Accuracy: 0.9472222222222222
Sensitivity: 0.9083333333333333
Specificity: 0.9861111111111112
F1-Score: 0.9450867052023121

              precision    recall  f1-score   support

           0       0.91      0.99      0.95       360
           1       0.98      0.91      0.95       360

    accuracy                           0.95       720
   macro avg       0.95      0.95      0.95       720
weighted avg       0.95      0.95      0.95       720
```

```
# Predicted probability
y_train_pred_proba = logistic_bal_rus_model.predict_proba(X_train_rus)[:,1]
draw_roc(y_train_rus, y_train_pred_proba)
```

```
auc_score: 0.9874845679012346
```



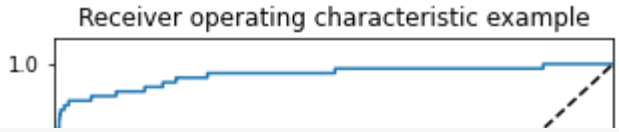
Prediction on the test set

```
# Prediction on the test set
y_test_pred = logistic_bal_rus_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

```
Confusion Matrix:
[[55277  1356]
 [    9   104]]

Accuracy: 0.9759454410883587
Sensitivity: 0.9203539823008849
Specificity: 0.976056362897957
F1-Score: 0.1322314049586777

              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56633
           1       0.07      0.92      0.13       113

    accuracy                           0.98     56746
   macro avg       0.54      0.95      0.56     56746
weighted avg       1.00      0.98      0.99     56746
```

```
# Predicted probability
y_test_pred_proba = logistic_bal_rus_model.predict_proba(X_test)[:,1]
draw_roc(y_test, y_test_pred_proba)
```

```
auc_score: 0.9776022579161686
```



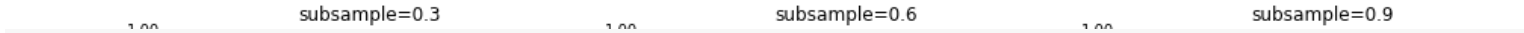

```
# hyperparameter tuning with XGBoost

# creating a KFold object
folds = 3

# specify range of hyperparameters
param_grid = {'learning_rate': [0.2, 0.6],
              'subsample': [0.3, 0.6, 0.9]}


# specify model
xgb_model = XGBClassifier(max_depth=2, n_estimators=200)

# set up GridSearchCV()
model_cv = GridSearchCV(estimator = xgb_model,
                        param_grid = param_grid,
                        scoring= 'roc_auc',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# fit the model
model_cv.fit(X_train_rus, y_train_rus)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
GridSearchCV(cv=3, estimator=XGBClassifier(max_depth=2, n_estimators=200),
             param_grid={'learning_rate': [0.2, 0.6],
                         'subsample': [0.3, 0.6, 0.9]},
             return_train_score=True, scoring='roc_auc', verbose=1)
```

```
# cv results
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	p
0	0.098411	0.001663	0.004170	0.000159	0.2	
1	0.120241	0.003088	0.004197	0.000109	0.2	
2	0.117338	0.000825	0.004177	0.000034	0.2	

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

param_grid = {'learning_rate': [0.2, 0.6],
              'subsample': [0.3, 0.6, 0.9]}


for n, subsample in enumerate(param_grid['subsample']):


    # subplot 1/n
    plt.subplot(1,len(param_grid['subsample']), n+1)
    df = cv_results[cv_results['param_subsample']==subsample]

    plt.plot(df["param_learning_rate"], df["mean_test_score"])
    plt.plot(df["param_learning_rate"], df["mean_train_score"])
    plt.xlabel('learning_rate')
    plt.ylabel('AUC')
    plt.title("subsample={0}".format(subsample))
    plt.ylim([0.60, 1])
    plt.legend(['test score', 'train score'], loc='upper left')
    plt.xscale('log')
```




```
model_cv.best_params_
```

```
{'learning_rate': 0.2, 'subsample': 0.9}
```

```
# chosen hyperparameters
# 'objective':'binary:logistic' outputs probability rather than label, which we need for cal
parameters = {'learning_rate': 0.2,
          'max_depth': 2,
          'n_estimators':200,
          'subsample':0.6,
         'objective':'binary:logistic'}

# fit model on training data
xgb_bal_rus_model = XGBClassifier(params = parameters)
xgb_bal_rus_model.fit(X_train_rus, y_train_rus)
```

```
XGBClassifier(params={'learning_rate': 0.2, 'max_depth': 2, 'n_estimators': 200,
                      'objective': 'binary:logistic', 'subsample': 0.6})
```

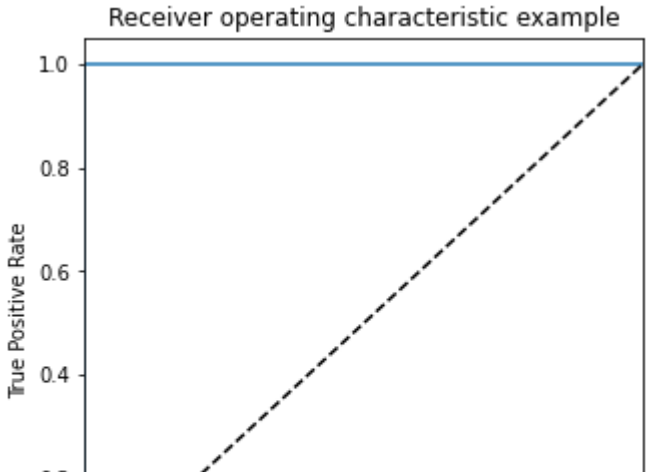
Prediction on the train set

```
# Predictions on the train set
y_train_pred = xgb_bal_rus_model.predict(X_train_rus)
printMetrics(y_train_rus,y_train_pred)
```

```
Confusion Matrix:
[[360   0]
 [  1 359]]

Accuracy: 0.9986111111111111
Sensitivity: 0.9972222222222222
Specificity: 1.0
F1-Score: 0.9986091794158554

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       360
           1       1.00      1.00      1.00       360

    accuracy                           1.00       720
   macro avg       1.00      1.00      1.00       720
weighted avg       1.00      1.00      1.00       720
```

```
# Predicted probability
y_train_pred_proba = xgb_bal_rus_model.predict_proba(X_train_rus)[:,1]
draw_roc(y_train_rus, y_train_pred_proba)
```

```
auc_score: 1.0
```



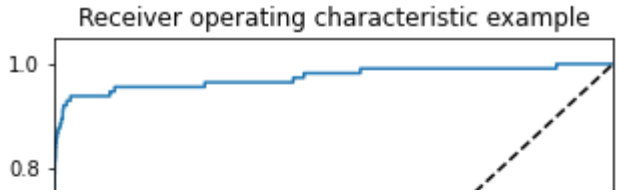
Prediction on the test set

```
# Predictions on the test set
y_test_pred = xgb_bal_rus_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

```
Confusion Matrix:
[[54662  1971]
 [    7   106]]

Accuracy: 0.9651429175624714
Sensitivity: 0.9380530973451328
Specificity: 0.9651969699645083
F1-Score: 0.09680365296803653

              precision    recall  f1-score   support

           0       1.00      0.97      0.98     56633
           1       0.05      0.94      0.10       113

    accuracy                           0.97     56746
   macro avg       0.53      0.95      0.54     56746
weighted avg       1.00      0.97      0.98     56746
```

```
# Predicted probability
y_test_pred_proba = xgb_bal_rus_model.predict_proba(X_test)[:,1]
draw_roc(y_test, y_test_pred_proba)
```

```
auc_score: 0.9738061972998324
```



Decision Tree

```
# Create the parameter grid
param_grid = {
    'max_depth': range(5, 15, 5),
    'min_samples_leaf': range(50, 150, 50),
    'min_samples_split': range(50, 150, 50),
}


# Instantiate the grid search model
dtree = DecisionTreeClassifier()

grid_search = GridSearchCV(estimator = dtree,
                           param_grid = param_grid,
                           scoring= 'roc_auc',
                           cv = 3,
                           verbose = 1)

# Fit the grid search to the data
grid_search.fit(X_train_rus,y_train_rus)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': range(5, 15, 5),
                         'min_samples_leaf': range(50, 150, 50),
                         'min_samples_split': range(50, 150, 50)},
             scoring='roc_auc', verbose=1)
```

```
# cv results
cv_results = pd.DataFrame(grid_search.cv_results_)
cv_results
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param
0	0.007949	0.001617	0.003744	0.000350	5	
1	0.006749	0.000055	0.003372	0.000098	5	
2	0.005120	0.000059	0.003376	0.000246	5	
3	0.005575	0.000095	0.003569	0.000231	5	

```
# Printing the optimal sensitivity score and hyperparameters
print("Best roc_auc:-", grid_search.best_score_)
print(grid_search.best_estimator_)
```

```
Best roc_auc:- 0.9617361111111112
DecisionTreeClassifier(max_depth=5, min_samples_leaf=50, min_samples_split=50)
```

```
# Model with optimal hyperparameters
dt_bal_rus_model = DecisionTreeClassifier(criterion = "gini",
                                  random_state = 100,
                                  max_depth=5,
                                  min_samples_leaf=50,
                                  min_samples_split=50)

dt_bal_rus_model.fit(X_train_rus, y_train_rus)
```

```
DecisionTreeClassifier(max_depth=5, min_samples_leaf=50, min_samples_split=50,
                       random_state=100)
```

Prediction on the train set

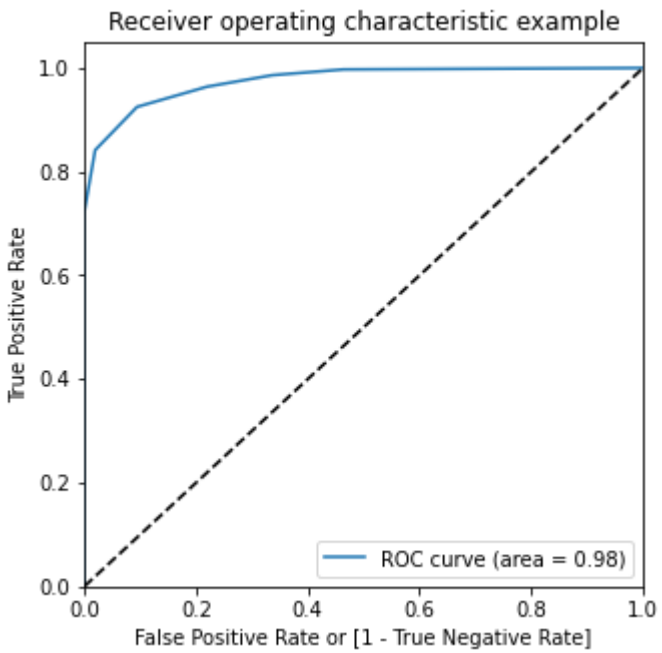
```
# Predictions on the train set
y_train_pred = dt_bal_rus_model.predict(X_train_rus)
printMetrics(y_train_rus,y_train_pred)
```

```
Confusion Matrix:
[[326  34]
 [ 27 333]]

Accuracy: 0.9152777777777777
Sensitivity: 0.925
Specificity: 0.9055555555555556
F1-Score: 0.9160935350756534

              precision    recall  f1-score   support

           0       0.92      0.91      0.91       360
           1       0.91      0.93      0.92       360

    accuracy                           0.92       720
```

```
       macro avg       0.92      0.92      0.92       720
    weighted avg       0.92      0.92      0.92       720
```

```
# Predicted probability
y_train_pred_proba = dt_bal_rus_model.predict_proba(X_train_rus)[:,1]
draw_roc(y_train_rus, y_train_pred_proba)
```

```
    auc_score: 0.9753395061728395
```



Prediction on the test set

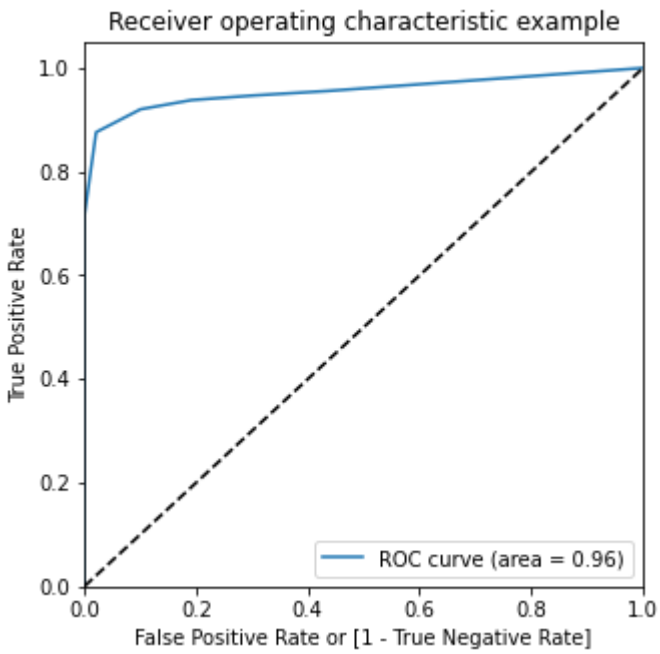
```
# Predictions on the test set
y_test_pred = dt_bal_rus_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

```
    Confusion Matrix:
    [[50940  5693]
     [    9   104]]

    Accuracy: 0.8995171465830191
    Sensitivity: 0.9203539823008849
    Specificity: 0.8994755707802871
    F1-Score: 0.03519458544839256

                  precision    recall  f1-score   support

               0       1.00      0.90      0.95     56633
               1       0.02      0.92      0.04       113

        accuracy                           0.90     56746
       macro avg       0.51      0.91      0.49     56746
    weighted avg       1.00      0.90      0.95     56746
```

```
# Predicted probability
y_test_pred_proba = dt_bal_rus_model.predict_proba(X_test)[:,1]
```

```
draw_roc(y_test, y_test_pred_proba)
```

```
auc_score: 0.9556262656204855
```



Random forest

```
param_grid = {
    'max_depth': range(5,10,5),
    'min_samples_leaf': range(50, 150, 50),
    'min_samples_split': range(50, 150, 50),
    'n_estimators': [100,200,300],
    'max_features': [10, 20]
}
# Create a based model
rf = RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf,
                           param_grid = param_grid,
                           scoring= 'roc_auc',
                           cv = 2,
                           n_jobs = -1,
                           verbose = 1,
                           return_train_score=True)

# Fit the model
grid_search.fit(X_train_rus, y_train_rus)
```

```
Fitting 2 folds for each of 24 candidates, totalling 48 fits
GridSearchCV(cv=2, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'max_depth': range(5, 10, 5), 'max_features': [10, 20],
                         'min_samples_leaf': range(50, 150, 50),
                         'min_samples_split': range(50, 150, 50),
                         'n_estimators': [100, 200, 300]},
             return_train_score=True, scoring='roc_auc', verbose=1)
```

```
# printing the optimal accuracy score and hyperparameters
print('We can get roc-auc of',grid_search.best_score_,'using',grid_search.best_params_)
```

```
We can get roc-auc of 0.9757407407407407 using {'max_depth': 5, 'max_features': 20, 'mi
```

```
# model with the best hyperparameters

rfc_bal_rus_model = RandomForestClassifier(bootstrap=True,
                             max_depth=5,
                             min_samples_leaf=50,
                             min_samples_split=50,
                             max_features=10,
                             n_estimators=200)
```

```
# Fit the model
rfc_bal_rus_model.fit(X_train_rus, y_train_rus)
```

```
RandomForestClassifier(max_depth=5, max_features=10, min_samples_leaf=50,
                       min_samples_split=50, n_estimators=200)
```

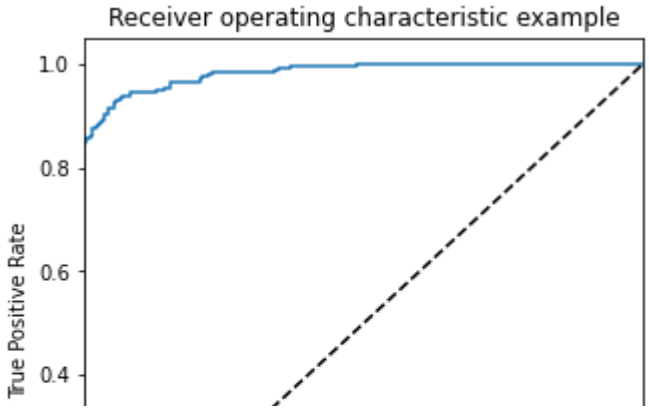
Prediction on the train set

```
# Predictions on the train set
y_train_pred = rfc_bal_rus_model.predict(X_train_rus)
printMetrics(y_train_rus,y_train_pred)
```

```
Confusion Matrix:
[[355   5]
 [ 47 313]]

Accuracy: 0.9277777777777778
Sensitivity: 0.8694444444444445
Specificity: 0.9861111111111112
F1-Score: 0.9233038348082595

              precision    recall  f1-score   support

           0       0.88      0.99      0.93       360
           1       0.98      0.87      0.92       360

    accuracy                           0.93       720
   macro avg       0.93      0.93      0.93       720
weighted avg       0.93      0.93      0.93       720
```

```
# Predicted probability
y_train_pred_proba = rfc_bal_rus_model.predict_proba(X_train_rus)[:,1]
draw_roc(y_train_rus, y_train_pred_proba)
```

```
auc_score: 0.9835570987654321
```



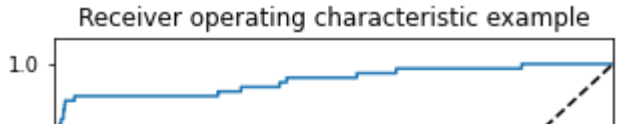
Prediction on the test set

```
# Predictions on the test set
y_test_pred = rfc_bal_rus_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

```
Confusion Matrix:
[[55836   797]
 [   12   101]]

Accuracy: 0.9857434885278258
Sensitivity: 0.8938053097345132
Specificity: 0.9859269330602299
F1-Score: 0.19980217606330364

              precision    recall  f1-score   support

           0       1.00      0.99      0.99     56633
           1       0.11      0.89      0.20       113

    accuracy                           0.99     56746
   macro avg       0.56      0.94      0.60     56746
weighted avg       1.00      0.99      0.99     56746
```

```
# Predicted probability
y_test_pred_proba = rfc_bal_rus_model.predict_proba(X_test)[:,1]
draw_roc(y_test, y_test_pred_proba)
```

```
auc_score: 0.968261414238454
```



Oversampling

```
# Importing oversampler library
from imblearn.over_sampling import RandomOverSampler
```

```
# instantiating the random oversampler
ros = RandomOverSampler()
# resampling X, y
X_train_ros, y_train_ros = ros.fit_resample(X_train, y_train)
```

```
# Befor sampling class distribution
print('Before sampling class distribution:-',Counter(y_train))
# new class distribution
print('New class distribution:-',Counter(y_train_ros))
```

```
Before sampling class distribution:- Counter({0: 226620, 1: 360})
New class distribution:- Counter({0: 226620, 1: 226620})
```

Logistic Regression

```
# Creating KFold object with 5 splits
folds = KFold(n_splits=5, shuffle=True, random_state=4)

# Specify params
parameters = {"C": [0.01, 0.1, 1, 10, 100, 1000]}

# Specifing score as roc-auc
model_cv = GridSearchCV(estimator = LogisticRegression(),
                        param_grid = parameters,
                        scoring= 'roc_auc',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# Fit the model
model_cv.fit(X_train_ros, y_train_ros)
```

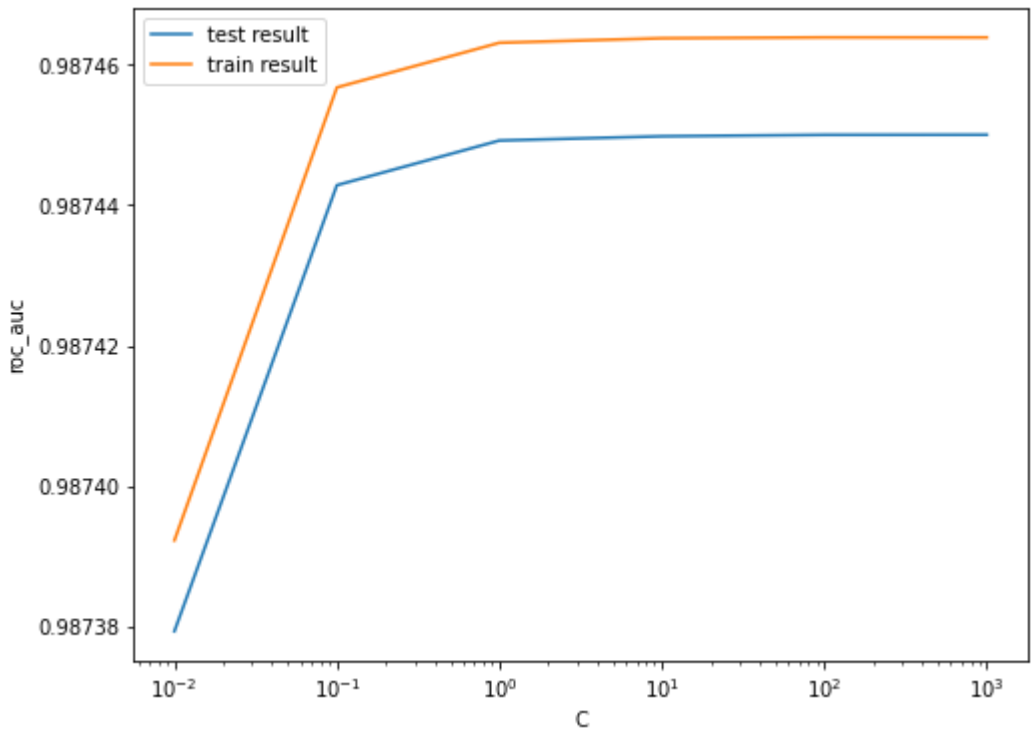
```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
GridSearchCV(cv=KFold(n_splits=5, random_state=4, shuffle=True),
             estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1, 10, 100, 1000]},
             return_train_score=True, scoring='roc_auc', verbose=1)
```

```
# results of grid search CV
cv_results = pd.DataFrame(model_cv.cv_results_)
```

```
cv_results
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split(
0	3.083592	0.170520	0.057131	0.001564	0.01	{'C': 0.01}	
1	3.402848	0.206236	0.054024	0.000608	0.1	{'C': 0.1}	
2	3.660433	0.180846	0.060969	0.001520	1	{'C': 1}	
3	3.809635	0.139382	0.060846	0.001838	10	{'C': 10}	
4	3.596082	0.197376	0.058060	0.002654	100	{'C': 100}	
5	3.619099	0.136464	0.061002	0.005739	1000	{'C': 1000}	

```
# plot of C versus train and validation scores

plt.figure(figsize=(8, 6))
plt.plot(cv_results['param_C'], cv_results['mean_test_score'])
plt.plot(cv_results['param_C'], cv_results['mean_train_score'])
plt.xlabel('C')
plt.ylabel('roc_auc')
plt.legend(['test result', 'train result'], loc='upper left')
plt.xscale('log')
```




```
# Best score with best C
best_score = model_cv.best_score_
best_C = model_cv.best_params_['C']

print(" The highest test roc_auc is {0} at C = {1}".format(best_score, best_C))
```

```
The highest test roc_auc is 0.9874500252816076 at C = 1000
```

Logistic regression with optimal C

```
# Instantiate the model with best C
logistic_bal_ros = LogisticRegression(C=0.1)
```

```
# Fit the model on the train set
logistic_bal_ros_model = logistic_bal_ros.fit(X_train_ros, y_train_ros)
```

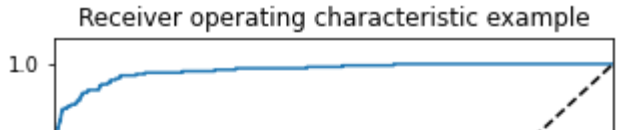
Prediction on the train set

```
# Predictions on the train set
y_train_pred = logistic_bal_ros_model.predict(X_train_ros)
printMetrics(y_train_ros,y_train_pred)
```

```
Confusion Matrix:
[[220204   6416]
 [ 18348 208272]]

Accuracy: 0.9453622804695084
Sensitivity: 0.9190362721736828
Specificity: 0.9716882887653341
F1-Score: 0.9438849964197341

              precision    recall  f1-score   support

           0       0.92      0.97      0.95    226620
           1       0.97      0.92      0.94    226620

    accuracy                           0.95    453240
   macro avg       0.95      0.95      0.95    453240
weighted avg       0.95      0.95      0.95    453240
```

```
# Predicted probability
y_train_pred_proba = logistic_bal_ros_model.predict_proba(X_train_ros)[:,1]
draw_roc(y_train_ros, y_train_pred_proba)
```

```
auc_score: 0.9874531913744706
```



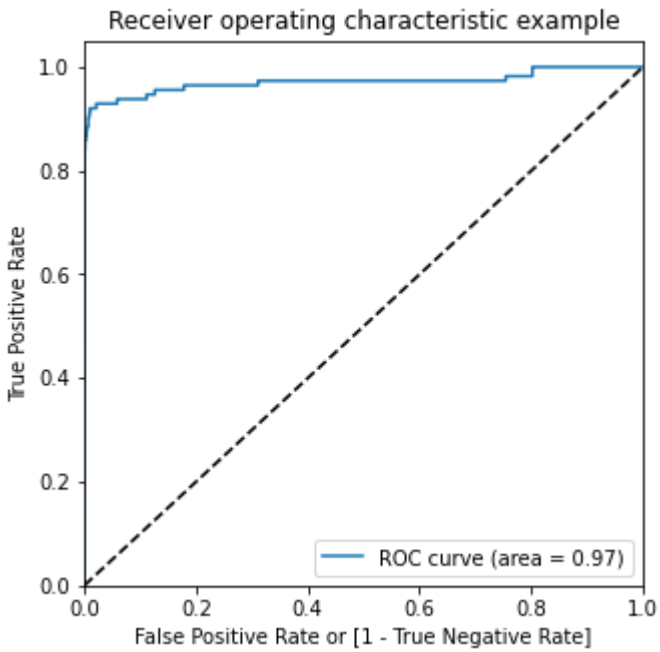
Prediction on the test set

```
# Prediction on the test set
y_test_pred = logistic_bal_ros_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

```
Confusion Matrix:
[[54967  1666]
 [    8   105]]

Accuracy: 0.9705001233567123
Sensitivity: 0.9292035398230089
Specificity: 0.9705825225575194
F1-Score: 0.11146496815286623

              precision    recall  f1-score   support

           0       1.00      0.97      0.99     56633
           1       0.06      0.93      0.11       113

    accuracy                           0.97     56746
   macro avg       0.53      0.95      0.55     56746
weighted avg       1.00      0.97      0.98     56746
```

```
# Predicted probability
y_test_pred_proba = logistic_bal_ros_model.predict_proba(X_test)[:,1]
draw_roc(y_test, y_test_pred_proba)
```

```
auc_score: 0.9714574306952902
```



XGBoost

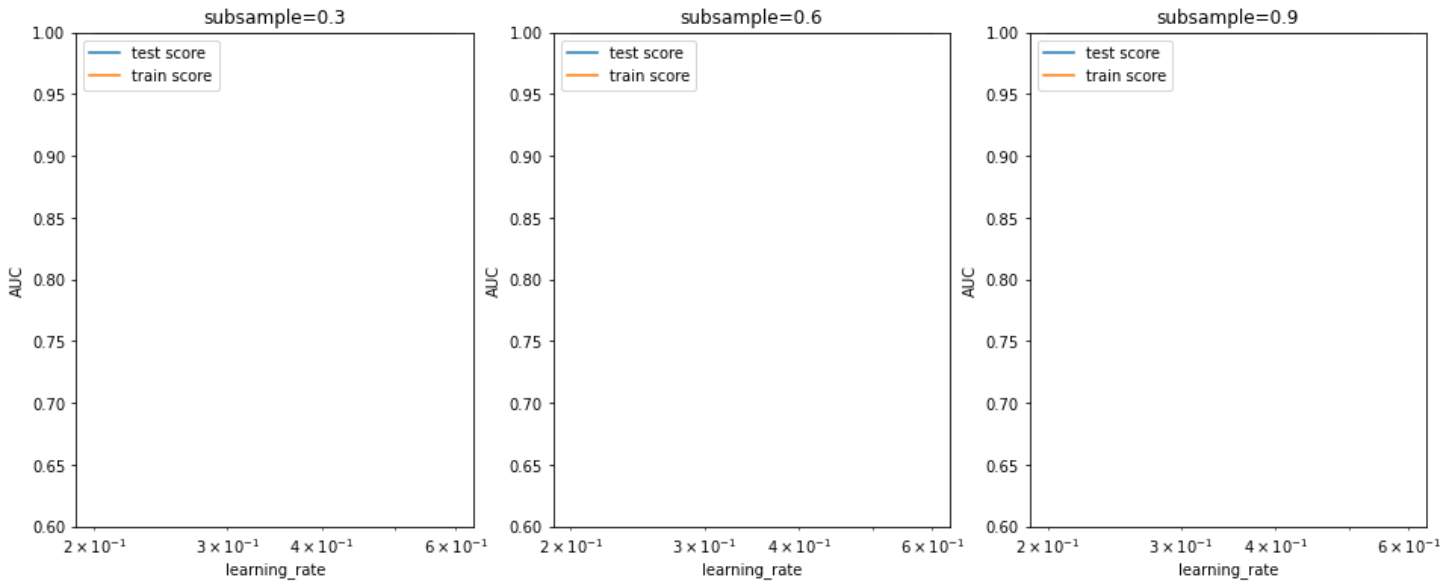
```
# hyperparameter tuning with XGBoost

# creating a KFold object
folds = 3

# specify range of hyperparameters
param_grid = {'learning_rate': [0.2, 0.6],
              'subsample': [0.3, 0.6, 0.9]}



# specify model
xgb_model = XGBClassifier(max_depth=2, n_estimators=200)

# set up GridSearchCV()
model_cv = GridSearchCV(estimator = xgb_model,
                        param_grid = param_grid,
                        scoring= 'roc_auc',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# fit the model
model_cv.fit(X_train_ros, y_train_ros)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
GridSearchCV(cv=3, estimator=XGBClassifier(max_depth=2, n_estimators=200),
             param_grid={'learning_rate': [0.2, 0.6],
                         'subsample': [0.3, 0.6, 0.9]},
             return_train_score=True, scoring='roc_auc', verbose=1)
```

```
# cv results
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	p
0	75.386732	0.847432	0.811115	0.015922	0.2	
1	89.852803	2.574452	0.822057	0.018485	0.2	

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

param_grid = {'learning_rate': [0.2, 0.6],
              'subsample': [0.3, 0.6, 0.9]}


for n, subsample in enumerate(param_grid['subsample']):


    # subplot 1/n
    plt.subplot(1,len(param_grid['subsample']), n+1)
    df = cv_results[cv_results['param_subsample']==subsample]

    plt.plot(df["param_learning_rate"], df["mean_test_score"])
    plt.plot(df["param_learning_rate"], df["mean_train_score"])
    plt.xlabel('learning_rate')
    plt.ylabel('AUC')
    plt.title("subsample={0}".format(subsample))
    plt.ylim([0.60, 1])
    plt.legend(['test score', 'train score'], loc='upper left')
    plt.xscale('log')
```




```
model_cv.best_params_
```

```
{'learning_rate': 0.6, 'subsample': 0.9}
```

```
# chosen hyperparameters
parameters = {'learning_rate': 0.6,
          'max_depth': 2,
          'n_estimators':200,
          'subsample':0.9,
         'objective':'binary:logistic'}

# fit model on training data
xgb_bal_ros_model = XGBClassifier(params = parameters)
xgb_bal_ros_model.fit(X_train_ros, y_train_ros)
```

```
XGBClassifier(params={'learning_rate': 0.6, 'max_depth': 2, 'n_estimators': 200,
                      'objective': 'binary:logistic', 'subsample': 0.9})
```

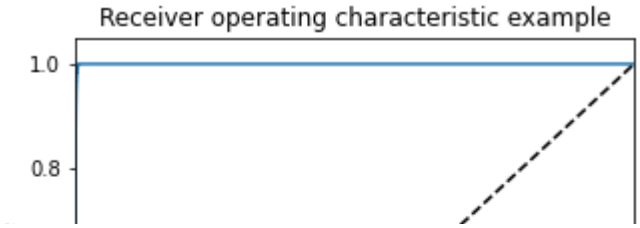
Prediction on the train set

```
# Predictions on the train set
y_train_pred = xgb_bal_ros_model.predict(X_train_ros)
printMetrics(y_train_ros,y_train_pred)
```

```
Confusion Matrix:
[[225175   1445]
 [     0 226620]]

Accuracy: 0.9968118436148619
Sensitivity: 1.0
Specificity: 0.9936236872297237
F1-Score: 0.9968219756534744

              precision    recall  f1-score   support

           0       1.00      0.99      1.00    226620
           1       0.99      1.00      1.00    226620

    accuracy                           1.00    453240
   macro avg       1.00      1.00      1.00    453240
weighted avg       1.00      1.00      1.00    453240
```

```
# Predicted probability
y_train_pred_proba = xgb_bal_ros_model.predict_proba(X_train_ros)[:,1]
draw_roc(y_train_ros, y_train_pred_proba)
```

```
auc_score: 0.9996625032660831
```



Prediction on the test set

```
# Predictions on the test set
y_test_pred = xgb_bal_ros_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

```
Confusion Matrix:
[[56226   407]
 [   11   102]]

Accuracy: 0.9926338420329186
Sensitivity: 0.9026548672566371
Specificity: 0.99281337735949
F1-Score: 0.32797427652733124

              precision    recall  f1-score   support

           0       1.00      0.99      1.00     56633
           1       0.20      0.90      0.33       113

    accuracy                           0.99     56746
   macro avg       0.60      0.95      0.66     56746
weighted avg       1.00      0.99      0.99     56746
```

```
# Predicted probability
y_test_pred_proba = xgb_bal_ros_model.predict_proba(X_test)[:,1]
draw_roc(y_test, y_test_pred_proba)
```

avg score: 0.975315214604074

Decision Tree

```
# Create the parameter grid
param_grid = {
    'max_depth': range(5, 15, 5),
    'min_samples_leaf': range(50, 150, 50),
    'min_samples_split': range(50, 150, 50),
}


# Instantiate the grid search model
dtree = DecisionTreeClassifier()

grid_search = GridSearchCV(estimator = dtree,
                           param_grid = param_grid,
                           scoring= 'roc_auc',
                           cv = 3,
                           verbose = 1)

# Fit the grid search to the data
grid_search.fit(X_train_ros,y_train_ros)
```

```
    Fitting 3 folds for each of 8 candidates, totalling 24 fits
    GridSearchCV(cv=3, estimator=DecisionTreeClassifier(),
                 param_grid={'max_depth': range(5, 15, 5),
                             'min_samples_leaf': range(50, 150, 50),
                             'min_samples_split': range(50, 150, 50)},
                 scoring='roc_auc', verbose=1)
```

```
# cv results
cv_results = pd.DataFrame(grid_search.cv_results_)
cv_results
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_
0	5.018437	0.151387	0.060709	0.003268	
1	4.992461	0.062468	0.060271	0.000329	

```
# Printing the optimal sensitivity score and hyperparameters
print("Best roc_auc:-", grid_search.best_score_)
print(grid_search.best_estimator_)
```

```
Best roc_auc:- 0.9988974090166254
DecisionTreeClassifier(max_depth=10, min_samples_leaf=100,
                       min_samples_split=100)
```

4	9.000366	0.446542	0.064407	0.000606	

```
# Model with optimal hyperparameters
dt_bal_ros_model = DecisionTreeClassifier(criterion = "gini",
                                  random_state = 100,
                                  max_depth=10,
                                  min_samples_leaf=100,
                                  min_samples_split=50)

dt_bal_ros_model.fit(X_train_ros, y_train_ros)
```

```
DecisionTreeClassifier(max_depth=10, min_samples_leaf=100, min_samples_split=50,
                       random_state=100)
```

Prediction on the train set

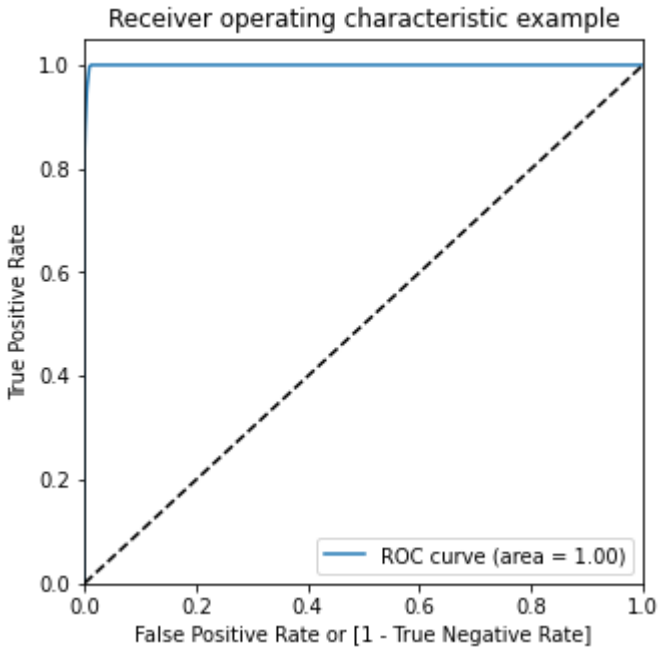
```
# Predictions on the train set
y_train_pred = dt_bal_ros_model.predict(X_train_ros)
printMetrics(y_train_ros,y_train_pred)
```

```
Confusion Matrix:
[[223882   2738]
 [     0 226620]]

Accuracy: 0.9939590503927279
Sensitivity: 1.0
Specificity: 0.9879181007854558
F1-Score: 0.9939953243358232

              precision    recall  f1-score   support

           0       1.00      0.99      0.99    226620
           1       0.99      1.00      0.99    226620

    accuracy                           0.99    453240
   macro avg       0.99      0.99      0.99    453240
weighted avg       0.99      0.99      0.99    453240
```

```
# Predicted probability
```

```
y_train_pred_proba = dt_bal_ros_model.predict_proba(X_train_ros)[:,1]
draw_roc(y_train_ros, y_train_pred_proba)
```

```
auc_score: 0.9992724833371253
```



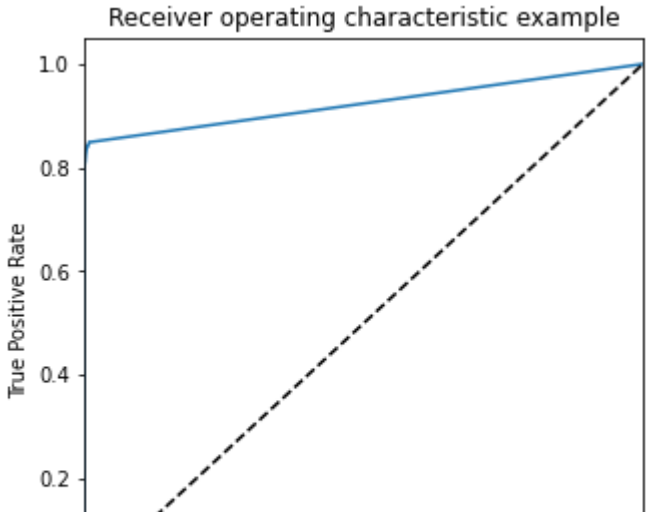
Prediction on the test set

```
# Predictions on the test set
y_test_pred = dt_bal_ros_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

```
Confusion Matrix:
[[55951   682]
 [   17    96]]

Accuracy: 0.9876819511507419
Sensitivity: 0.8495575221238938
Specificity: 0.9879575512510373
F1-Score: 0.21548821548821548

              precision    recall  f1-score   support

           0       1.00      0.99      0.99     56633
           1       0.12      0.85      0.22       113

    accuracy                           0.99     56746
   macro avg       0.56      0.92      0.60     56746
weighted avg       1.00      0.99      0.99     56746
```

```
# Predicted probability
y_test_pred_proba = dt_bal_ros_model.predict_proba(X_test)[:,1]
draw_roc(y_test, y_test_pred_proba)
```

```
auc_score: 0.9235328881234854
```



SMOTE (Synthetic Minority Oversampling Technique)

We are creating synthetic samples by doing upsampling using SMOTE(Synthetic Minority Oversampling Technique).

Double-click (or enter) to edit

```
# Importing SMOTE
from imblearn.over_sampling import SMOTE
```

```
# Instantiate SMOTE
sm = SMOTE(random_state=27)
# Fitting SMOTE to the train set
X_train_smote, y_train_smote = sm.fit_resample(X_train, y_train)
```

```
print('Before SMOTE oversampling X_train shape=',X_train.shape)
print('After SMOTE oversampling X_train shape=',X_train_smote.shape)
```

```
Before SMOTE oversampling X_train shape= (226980, 29)
After SMOTE oversampling X_train shape= (453240, 29)
```

Logistic Regression

```
# Creating KFold object with 5 splits
folds = KFold(n_splits=5, shuffle=True, random_state=4)

# Specify params
parameters = {"C": [0.01, 0.1, 1, 10, 100, 1000]}

# Specifing score as roc-auc
model_cv = GridSearchCV(estimator = LogisticRegression(),
                        param_grid = parameters,
                        scoring= 'roc_auc',
                        cv = folds,
```

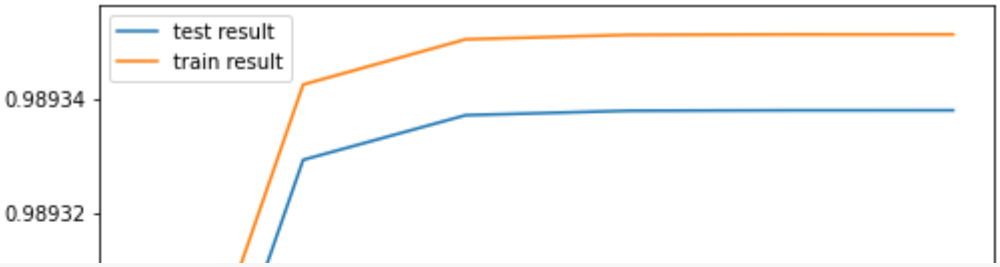
```
                        verbose = 1,
                        return_train_score=True)

# Fit the model
model_cv.fit(X_train_smote, y_train_smote)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
GridSearchCV(cv=KFold(n_splits=5, random_state=4, shuffle=True),
             estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1, 10, 100, 1000]},
             return_train_score=True, scoring='roc_auc', verbose=1)
```

```
# results of grid search CV
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	pa
0	2.645891	0.124924	0.061356	0.004081	0.01	
1	2.780287	0.054343	0.057003	0.000543	0.1	
2	2.870502	0.124850	0.059315	0.003054	1	{
3	2.974335	0.254205	0.059848	0.000676	10	{'C
4	2.918696	0.241031	0.058839	0.001704	100	
5	2.842196	0.196686	0.057196	0.000627	1000	

```
# plot of C versus train and validation scores

plt.figure(figsize=(8, 6))
plt.plot(cv_results['param_C'], cv_results['mean_test_score'])
plt.plot(cv_results['param_C'], cv_results['mean_train_score'])
plt.xlabel('C')
plt.ylabel('roc_auc')
plt.legend(['test result', 'train result'], loc='upper left')
plt.xscale('log')
```




```
# Best score with best C
best_score = model_cv.best_score_
best_C = model_cv.best_params_['C']

print(" The highest test roc_auc is {0} at C = {1}".format(best_score, best_C))
```

```
 The highest test roc_auc is 0.9893379935907044 at C = 1000
```

Logistic regression with optimal C


```
# Instantiate the model with best C
logistic_bal_smote = LogisticRegression(C=0.1)
```

```
# Fit the model on the train set
logistic_bal_smote_model = logistic_bal_smote.fit(X_train_smote, y_train_smote)
```

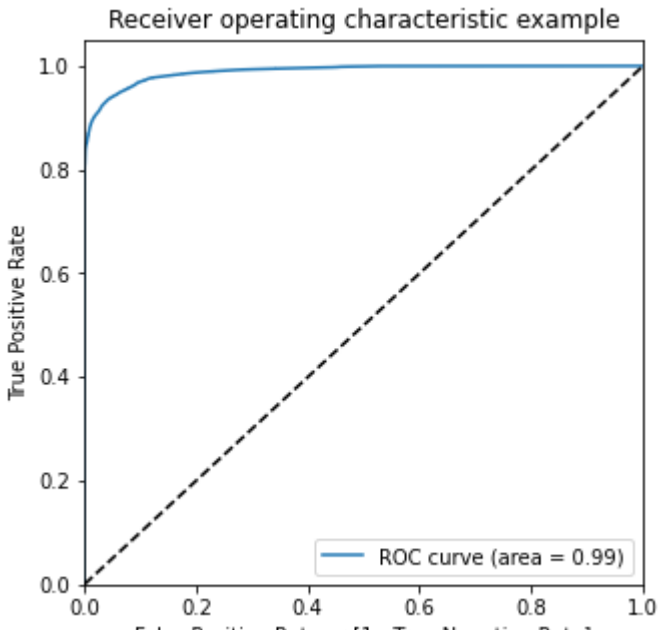
Prediction on the train set

```
# Predictions on the train set
y_train_pred = logistic_bal_smote_model.predict(X_train_smote)
printMetrics(y_train_smote,y_train_pred)
```

```
Confusion Matrix:
[[219714   6906]
 [ 17909 208711]]

Accuracy: 0.9452497573029741
Sensitivity: 0.9209734357073515
Specificity: 0.9695260788985968
F1-Score: 0.9438875535063778

              precision    recall  f1-score   support

           0       0.92      0.97      0.95    226620
           1       0.97      0.92      0.94    226620

    accuracy                           0.95    453240
   macro avg       0.95      0.95      0.95    453240
weighted avg       0.95      0.95      0.95    453240
```

```
# Predicted probability
y_train_pred_proba_log_bal_smote = logistic_bal_smote_model.predict_proba(X_train_smote)[:,1
draw_roc(y_train_smote, y_train_pred_proba_log_bal_smote)
```

```
auc_score: 0.9893406429083762
```



Prediction on the test set

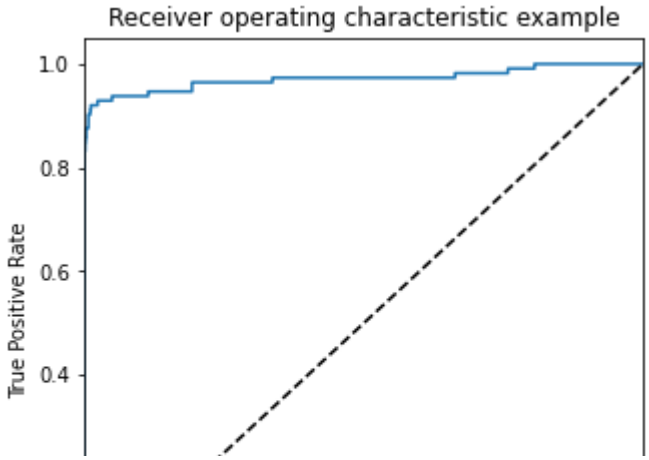
```
# Prediction on the test set
y_test_pred = logistic_bal_smote_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

```
Confusion Matrix:
[[54791  1842]
 [    8   105]]

Accuracy: 0.9673985831600466
Sensitivity: 0.9292035398230089
Specificity: 0.9674747938481097
F1-Score: 0.10194174757281553

              precision    recall  f1-score   support

           0       1.00      0.97      0.98     56633
           1       0.05      0.93      0.10       113

    accuracy                           0.97     56746
   macro avg       0.53      0.95      0.54     56746
weighted avg       1.00      0.97      0.98     56746
```

ROC on the test set

```
# Predicted probability
y_test_pred_proba = logistic_bal_smote_model.predict_proba(X_test)[:,1]
draw_roc(y_test, y_test_pred_proba)
```

```
auc_score: 0.9716343187131428
```



XGBoost

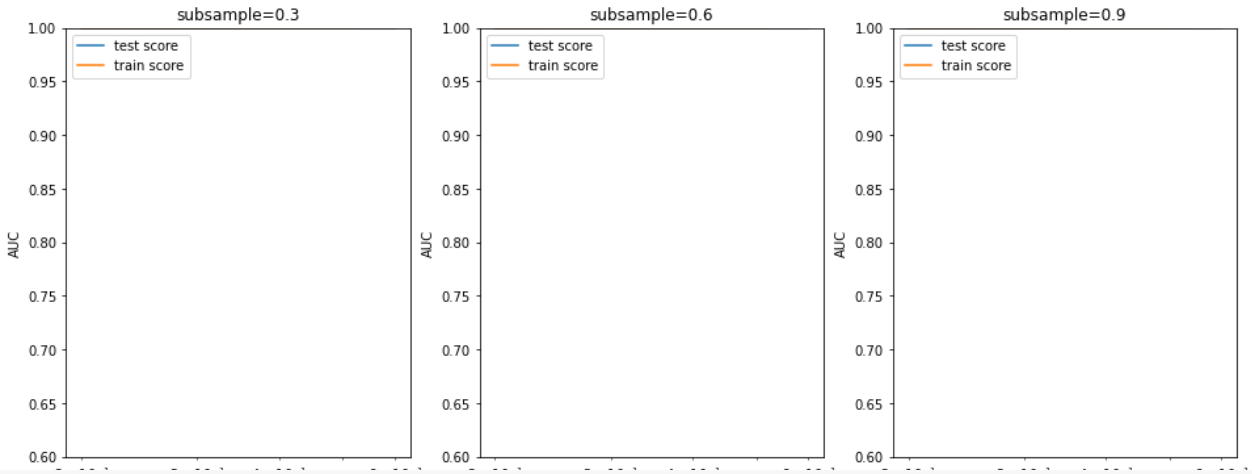

```
# hyperparameter tuning with XGBoost

# creating a KFold object
folds = 3

# specify range of hyperparameters
param_grid = {'learning_rate': [0.2, 0.6],
              'subsample': [0.3, 0.6, 0.9]}


# specify model
xgb_model = XGBClassifier(max_depth=2, n_estimators=200)

# set up GridSearchCV()
model_cv = GridSearchCV(estimator = xgb_model,
                        param_grid = param_grid,
                        scoring= 'roc_auc',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# fit the model
model_cv.fit(X_train_smote, y_train_smote)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
GridSearchCV(cv=3, estimator=XGBClassifier(max_depth=2, n_estimators=200),
             param_grid={'learning_rate': [0.2, 0.6],
                         'subsample': [0.3, 0.6, 0.9]},
             return_train_score=True, scoring='roc_auc', verbose=1)
```

```
# cv results
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learı
0	87.935340	1.303243	0.848209	0.015392	
1	108.401904	0.771585	0.833585	0.000893	
2	108.177118	0.192655	0.822421	0.004827	
3	80.555422	1.394461	0.827711	0.004892	
4	102.529270	1.930227	0.828451	0.001560	

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

param_grid = {'learning_rate': [0.2, 0.6],
              'subsample': [0.3, 0.6, 0.9]}


for n, subsample in enumerate(param_grid['subsample']):


    # subplot 1/n
    plt.subplot(1,len(param_grid['subsample']), n+1)
    df = cv_results[cv_results['param_subsample']==subsample]

    plt.plot(df["param_learning_rate"], df["mean_test_score"])
    plt.plot(df["param_learning_rate"], df["mean_train_score"])
    plt.xlabel('learning_rate')
    plt.ylabel('AUC')
    plt.title("subsample={0}".format(subsample))
    plt.ylim([0.60, 1])
    plt.legend(['test score', 'train score'], loc='upper left')
    plt.xscale('log')
```




```
model_cv.best_params_
```

```
{'learning_rate': 0.6, 'subsample': 0.6}
```

```
# chosen hyperparameters
# 'objective':'binary:logistic' outputs probability rather than label, which we need for cal
parameters = {'learning_rate': 0.6,
          'max_depth': 2,
          'n_estimators':200,
          'subsample':0.9,
         'objective':'binary:logistic'}

# fit model on training data
xgb_bal_smote_model = XGBClassifier(params = parameters)
xgb_bal_smote_model.fit(X_train_smote, y_train_smote)
```

```
XGBClassifier(params={'learning_rate': 0.6, 'max_depth': 2, 'n_estimators': 200,
                      'objective': 'binary:logistic', 'subsample': 0.9})
```

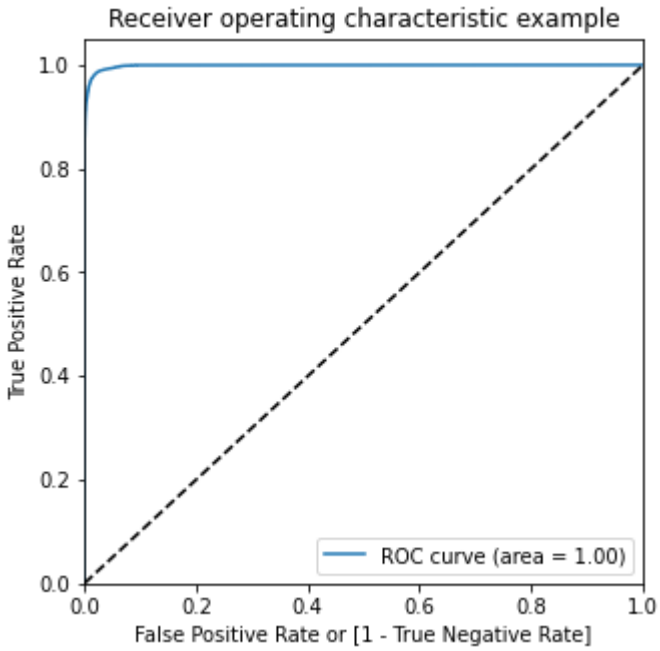
Prediction on the train set

```
# Predictions on the train set
y_train_pred = xgb_bal_smote_model.predict(X_train_smote)
printMetrics(y_train_smote,y_train_pred)
```

```
Confusion Matrix:
[[223654   2966]
 [  6032 220588]]

Accuracy: 0.9801473832847939
Sensitivity: 0.9733827552731444
Specificity: 0.9869120112964433
F1-Score: 0.9800121730708571

              precision    recall  f1-score   support

           0       0.97      0.99      0.98    226620
           1       0.99      0.97      0.98    226620

    accuracy                           0.98    453240
   macro avg       0.98      0.98      0.98    453240
weighted avg       0.98      0.98      0.98    453240
```

```
# Predicted probability
y_train_pred_proba = xgb_bal_smote_model.predict_proba(X_train_smote)[:,1]
draw_roc(y_train_smote, y_train_pred_proba)
```

```
auc_score: 0.9985584322107433
```



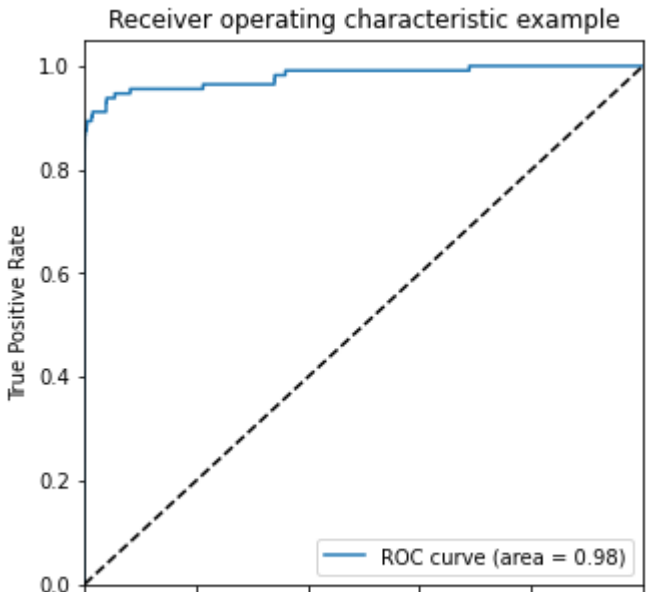
Prediction on the test set

```
# Predictions on the test set
y_test_pred = xgb_bal_smote_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

```
Confusion Matrix:
[[55801   832]
 [   11   102]]

Accuracy: 0.9851443273534698
Sensitivity: 0.9026548672566371
Specificity: 0.985308918828245
F1-Score: 0.19484240687679083

              precision    recall  f1-score   support

           0       1.00      0.99      0.99     56633
           1       0.11      0.90      0.19       113

    accuracy                           0.99     56746
   macro avg       0.55      0.94      0.59     56746
weighted avg       1.00      0.99      0.99     56746
```

```
# Predicted probability
y_test_pred_proba = xgb_bal_smote_model.predict_proba(X_test)[:,1]
draw_roc(y_test, y_test_pred_proba)
```

```
auc_score: 0.9801028325678343
```



Decision Tree

```
# Create the parameter grid
param_grid = {
    'max_depth': range(5, 15, 5),
    'min_samples_leaf': range(50, 150, 50),
    'min_samples_split': range(50, 150, 50),
}


# Instantiate the grid search model
dtree = DecisionTreeClassifier()

grid_search = GridSearchCV(estimator = dtree,
                           param_grid = param_grid,
                           scoring= 'roc_auc',
                           cv = 3,
                           verbose = 1)

# Fit the grid search to the data
grid_search.fit(X_train_smote,y_train_smote)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': range(5, 15, 5),
                         'min_samples_leaf': range(50, 150, 50),
                         'min_samples_split': range(50, 150, 50)},
             scoring='roc_auc', verbose=1)
```

```
# cv results
cv_results = pd.DataFrame(grid_search.cv_results_)
cv_results
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_
0	8.361555	0.013605	0.059040	0.000912	
1	8.299941	0.004916	0.058653	0.001088	
2	8.274418	0.018817	0.058539	0.001343	
3	8.286561	0.083875	0.058818	0.002580	
4	15.628470	0.298380	0.065915	0.002515	
5	16.216832	0.490540	0.065857	0.001107	
6	15.469517	0.120430	0.063013	0.000864	

```
# Printing the optimal sensitivity score and hyperparameters
print("Best roc_auc:-", grid_search.best_score_)
print(grid_search.best_estimator_)
```

```
Best roc_auc:- 0.9978912474726435
DecisionTreeClassifier(max_depth=10, min_samples_leaf=50, min_samples_split=100)
```

```
# Model with optimal hyperparameters
dt_bal_smote_model = DecisionTreeClassifier(criterion = "gini",
                                    random_state = 100,
                                    max_depth=10,
                                    min_samples_leaf=50,
                                    min_samples_split=100)

dt_bal_smote_model.fit(X_train_smote, y_train_smote)
```

```
DecisionTreeClassifier(max_depth=10, min_samples_leaf=50, min_samples_split=100,
                       random_state=100)
```

Prediction on the train set

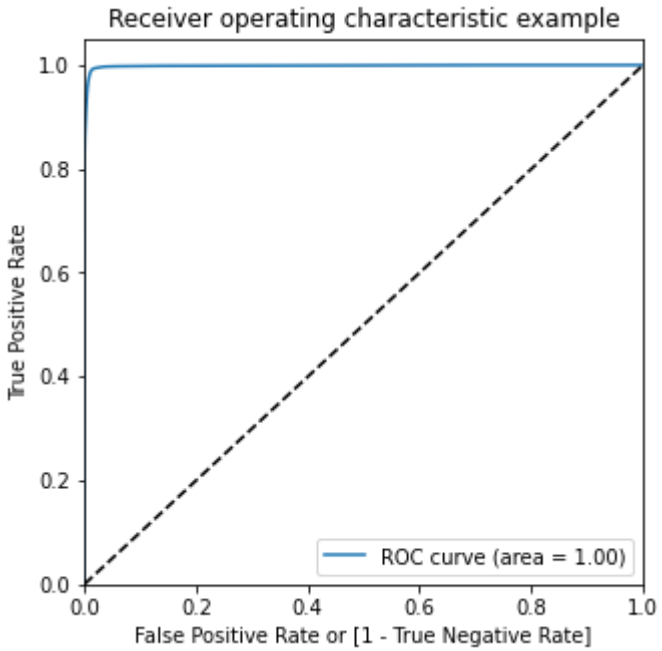
```
# Predictions on the train set
y_train_pred = dt_bal_smote_model.predict(X_train_smote)
printMetrics(y_train_smote,y_train_pred)
```

```
Confusion Matrix:
[[223537   3083]
 [  2017 224603]]
```

```
Accuracy: 0.9887476833465714
Sensitivity: 0.9910996381607978
Specificity: 0.9863957285323449
F1-Score: 0.9887740861886042

              precision    recall  f1-score   support

           0       0.99      0.99      0.99    226620
           1       0.99      0.99      0.99    226620

    accuracy                           0.99    453240
   macro avg       0.99      0.99      0.99    453240
weighted avg       0.99      0.99      0.99    453240
```

```
# Predicted probability
y_train_pred_proba = dt_bal_smote_model.predict_proba(X_train_smote)[:,1]
draw_roc(y_train_smote, y_train_pred_proba)
```

```
auc_score: 0.9984060705029516
```



Prediction on the test set

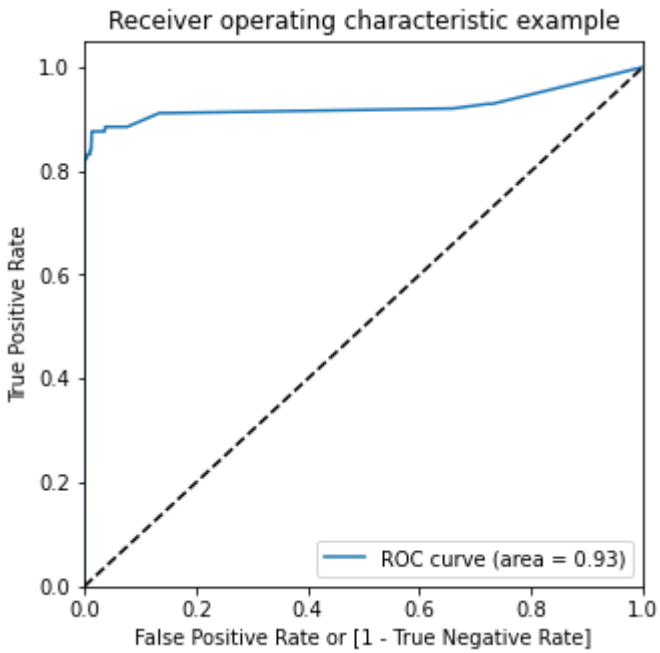
```
# Predictions on the test set
y_test_pred = dt_bal_smote_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

```
Confusion Matrix:
[[55848   785]
 [   14    99]]

Accuracy: 0.9859197124026363
Sensitivity: 0.8761061946902655
Specificity: 0.9861388236540533
F1-Score: 0.19859578736208627

              precision    recall  f1-score   support
```

```
              0       1.00      0.99      0.99     56633
              1       0.11      0.88      0.20       113

       accuracy                           0.99     56746
      macro avg       0.56      0.93      0.60     56746
   weighted avg       1.00      0.99      0.99     56746
```

```
# Predicted probability
y_test_pred_proba = dt_bal_smote_model.predict_proba(X_test)[:,1]
draw_roc(y_test, y_test_pred_proba)
```

```
    auc_score: 0.925130271305904
```



AdaSyn (Adaptive Synthetic Sampling)

```
# Importing adasyn
from imblearn.over_sampling import ADASYN
```

```
# Instantiate adasyn
ada = ADASYN(random_state=0)
X_train_adasyn, y_train_adasyn = ada.fit_resample(X_train, y_train)
```

```
# Befor sampling class distribution
print('Before sampling class distribution:-',Counter(y_train))
# new class distribution
print('New class distribution:-',Counter(y_train_adasyn))
```

```
    Before sampling class distribution:- Counter({0: 226620, 1: 360})
    New class distribution:- Counter({1: 226664, 0: 226620})
```

Logistic Regression

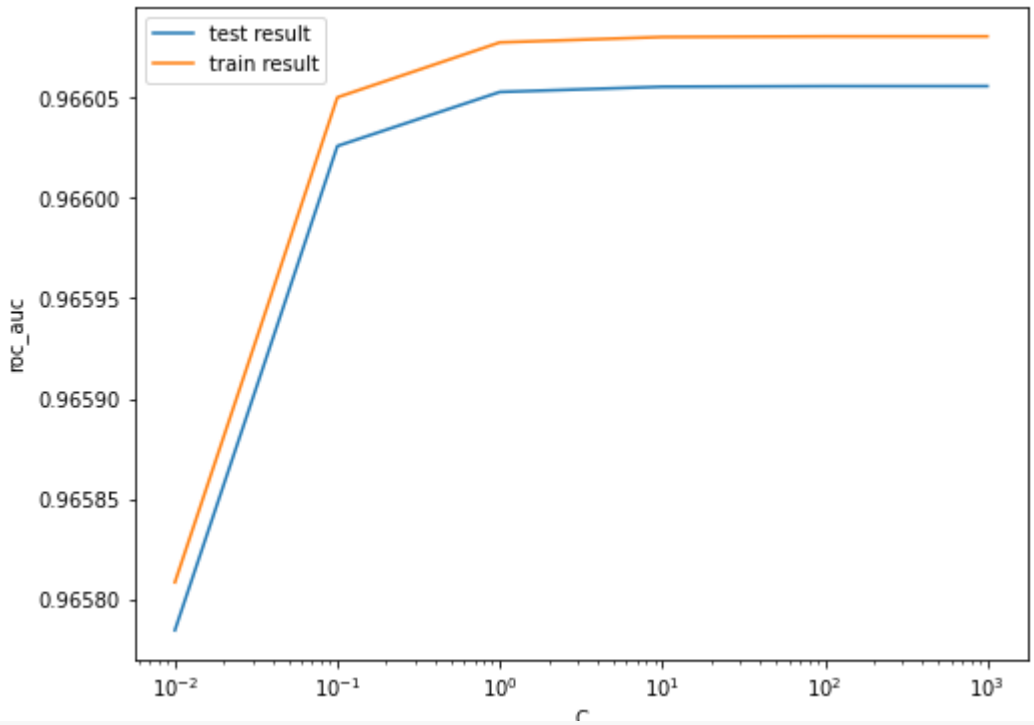
```
# Creating KFold object with 3 splits
folds = KFold(n_splits=3, shuffle=True, random_state=4)

# Specify params
parameters = {"C": [0.01, 0.1, 1, 10, 100, 1000]}

# Specifing score as roc-auc
model_cv = GridSearchCV(estimator = LogisticRegression(),
                        param_grid = parameters,
                        scoring= 'roc_auc',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# Fit the model
model_cv.fit(X_train_adasyn, y_train_adasyn)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
GridSearchCV(cv=KFold(n_splits=3, random_state=4, shuffle=True),
             estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1, 10, 100, 1000]},
             return_train_score=True, scoring='roc_auc', verbose=1)
```

```
# results of grid search CV
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	pa
0	2.542769	0.224000	0.095386	0.003324	0.01	
1	2.409889	0.090091	0.094004	0.000674	0.1	
2	2.536493	0.064795	0.094794	0.001127	1	{
3	2.511512	0.099003	0.093408	0.000746	10	{'C
4	2.491872	0.121463	0.093466	0.001388	100	
5	2.508108	0.124594	0.092004	0.000331	1000	

```
# plot of C versus train and validation scores

plt.figure(figsize=(8, 6))
plt.plot(cv_results['param_C'], cv_results['mean_test_score'])
plt.plot(cv_results['param_C'], cv_results['mean_train_score'])
plt.xlabel('C')
plt.ylabel('roc_auc')
plt.legend(['test result', 'train result'], loc='upper left')
plt.xscale('log')
```




```
# Best score with best C
best_score = model_cv.best_score_
best_C = model_cv.best_params_['C']

print(" The highest test roc_auc is {0} at C = {1}".format(best_score, best_C))
```

```
 The highest test roc_auc is 0.9660557793819096 at C = 1000
```

Logistic regression with optimal C

```
# Instantiate the model with best C
logistic_bal_adasyn = LogisticRegression(C=1000)
```

```
# Fit the model on the train set
logistic_bal_adasyn_model = logistic_bal_adasyn.fit(X_train_adasyn, y_train_adasyn)
```

Prediction on the train set

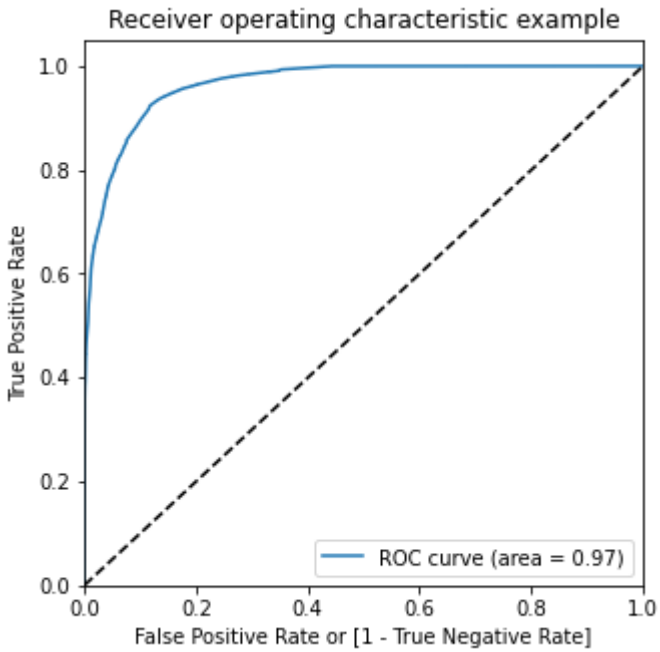
```
# Predictions on the train set
y_train_pred = logistic_bal_adasyn_model.predict(X_train_adasyn)
printMetrics(y_train_adasyn,y_train_pred)
```

```
Confusion Matrix:
[[205065  21555]
 [ 25614 201050]]

Accuracy: 0.8959394110535558
Sensitivity: 0.8869957293615219
```

```
Specificity: 0.9048848292295473
F1-Score: 0.8950094486821927

              precision    recall  f1-score   support

           0       0.89      0.90      0.90    226620
           1       0.90      0.89      0.90    226664

    accuracy                           0.90    453284
   macro avg       0.90      0.90      0.90    453284
weighted avg       0.90      0.90      0.90    453284
```

```
# Predicted probability
y_train_pred_proba = logistic_bal_adasyn_model.predict_proba(X_train_adasyn)[:,1]
draw_roc(y_train_adasyn, y_train_pred_proba)
```

```
auc_score: 0.9660790827981925
```



Prediction on the test set

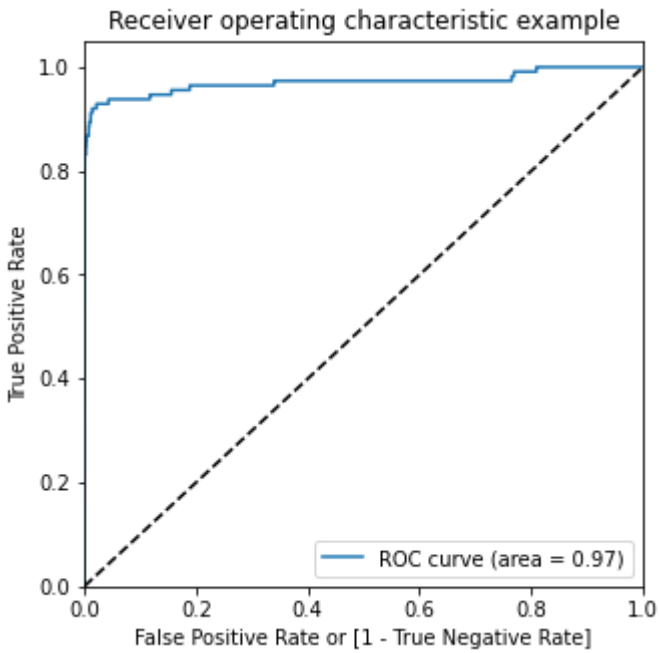
```
# Prediction on the test set
y_test_pred = logistic_bal_adasyn_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

```
Confusion Matrix:
[[51130  5503]
 [    7   106]]

Accuracy: 0.9029006449793818
Sensitivity: 0.9380530973451328
Specificity: 0.9028305051824908
F1-Score: 0.03704998252359315

              precision    recall  f1-score   support
```

```
              0       1.00      0.90      0.95     56633
              1       0.02      0.94      0.04       113

       accuracy                           0.90     56746
      macro avg       0.51      0.92      0.49     56746
   weighted avg       1.00      0.90      0.95     56746
```

```
# Predicted probability
y_test_pred_proba = logistic_bal_adasyn_model.predict_proba(X_test)[:,1]
draw_roc(y_test, y_test_pred_proba)
```

```
    auc_score: 0.9708185555530726
```



Double-click (or enter) to edit

Decision Tree

```
# Create the parameter grid
param_grid = {
    'max_depth': range(5, 15, 5),
    'min_samples_leaf': range(50, 150, 50),
    'min_samples_split': range(50, 150, 50),
}


# Instantiate the grid search model
dtree = DecisionTreeClassifier()

grid_search = GridSearchCV(estimator = dtree,
                           param_grid = param_grid,
                           scoring= 'roc_auc',
                           cv = 3,
                           verbose = 1)
```

```
# Fit the grid search to the data
grid_search.fit(X_train_adasyn,y_train_adasyn)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': range(5, 15, 5),
                         'min_samples_leaf': range(50, 150, 50),
                         'min_samples_split': range(50, 150, 50)},
```

```
# cv results
cv_results = pd.DataFrame(grid_search.cv_results_)
cv_results
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_
0	8.069930	0.042163	0.058630	0.000479	
1	8.069258	0.064079	0.058534	0.000785	
2	8.349008	0.209378	0.060632	0.004418	
3	8.172975	0.030532	0.059952	0.000718	
4	15.390765	0.217763	0.063275	0.000367	
5	15.377678	0.156238	0.064406	0.001803	
6	15.427018	0.061271	0.065295	0.001202	
7	15.391214	0.086432	0.066618	0.001367	

```
# Printing the optimal sensitivity score and hyperparameters
print("Best roc_auc:-", grid_search.best_score_)
print(grid_search.best_estimator_)
```

```
Best roc_auc:- 0.9272859769536357
DecisionTreeClassifier(max_depth=10, min_samples_leaf=100,
                       min_samples_split=100)
```

```
# Model with optimal hyperparameters
dt_bal_adasyn_model = DecisionTreeClassifier(criterion = "gini",
                                 random_state = 100,
                                 max_depth=10,
```

```
                                   min_samples_leaf=100,
                                   min_samples_split=50)

dt_bal_adasyn_model.fit(X_train_adasyn, y_train_adasyn)
```

```
DecisionTreeClassifier(max_depth=10, min_samples_leaf=100, min_samples_split=50,
                       random_state=100)
```

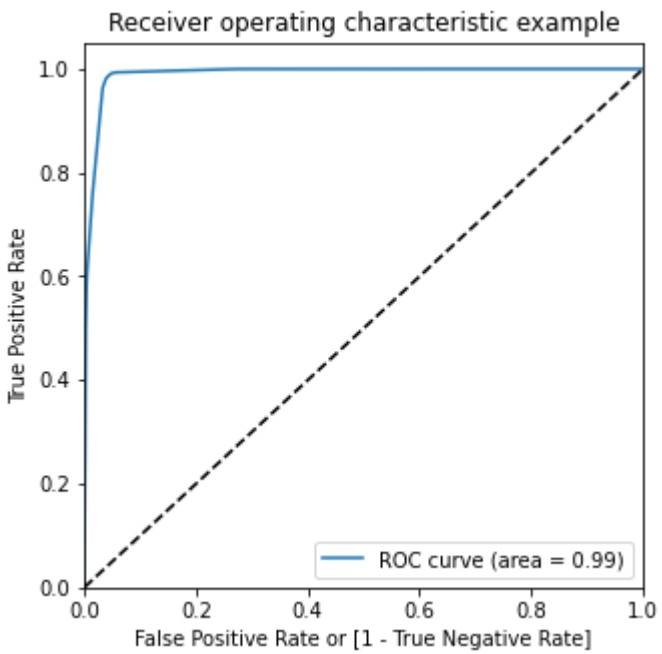
Prediction on the train set

```
# Predictions on the train set
y_train_pred = dt_bal_adasyn_model.predict(X_train_adasyn)
printMetrics(y_train_adasyn, y_train_pred)
```

```
Confusion Matrix:
[[217063   9557]
 [  3249 223415]]

Accuracy: 0.9717483961489927
Sensitivity: 0.9856660078353863
Specificity: 0.9578280822522284
F1-Score: 0.9721388228946384

              precision    recall  f1-score   support

           0       0.99      0.96      0.97    226620
           1       0.96      0.99      0.97    226664

    accuracy                           0.97    453284
   macro avg       0.97      0.97      0.97    453284
weighted avg       0.97      0.97      0.97    453284
```

```
# Predicted probability
y_train_pred_proba = dt_bal_adasyn_model.predict_proba(X_train_adasyn)[:,1]
draw_roc(y_train_adasyn, y_train_pred_proba)
```

```
auc_score: 0.9902968173206373
```



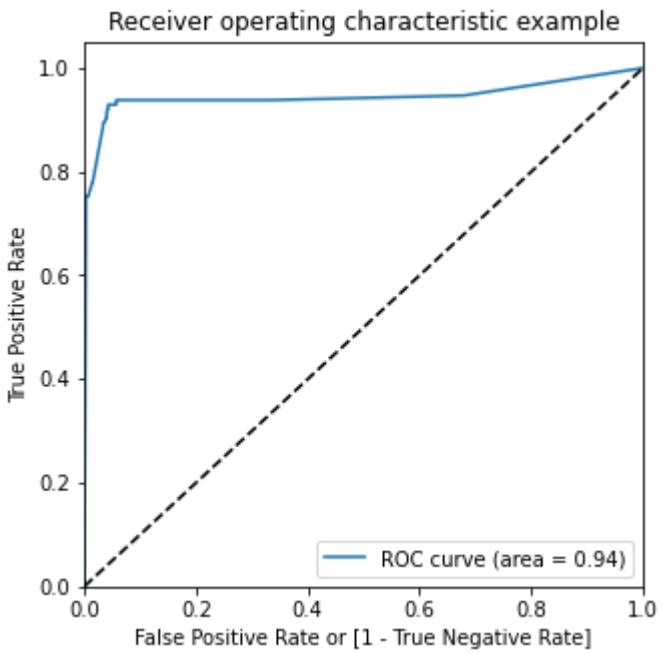
Prediction on the test set

```
# Predictions on the test set
y_test_pred = dt_bal_adasyn_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

```
Confusion Matrix:
[[54187  2446]
 [    8   105]]

Accuracy: 0.9567546611214888
Sensitivity: 0.9292035398230089
Specificity: 0.9568096339589992
F1-Score: 0.07882882882882883

              precision    recall  f1-score   support

           0       1.00      0.96      0.98     56633
           1       0.04      0.93      0.08       113

    accuracy                           0.96     56746
   macro avg       0.52      0.94      0.53     56746
weighted avg       1.00      0.96      0.98     56746
```

```
# Predicted probability
y_test_pred_proba = dt_bal_adasyn_model.predict_proba(X_test)[:,1]
draw_roc(y_test, y_test_pred_proba)
```

```
auc_score: 0.9447836707982729
```



XGBoost

```
# hyperparameter tuning with XGBoost

# creating a KFold object
```

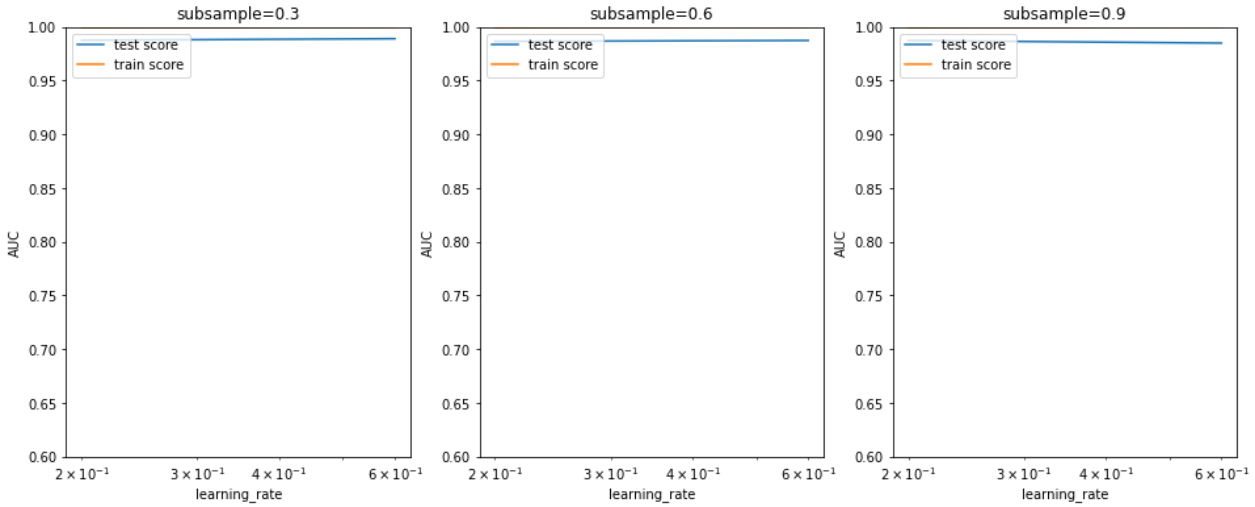
```
folds = 3

# specify range of hyperparameters
param_grid = {'learning_rate': [0.2, 0.6],
             'subsample': [0.3, 0.6, 0.9]}


# specify model
xgb_model = XGBClassifier(max_depth=2, n_estimators=200)

# set up GridSearchCV()
model_cv = GridSearchCV(estimator = xgb_model,
                        param_grid = param_grid,
                        scoring= 'roc_auc',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# fit the model
model_cv.fit(X_train_adasyn, y_train_adasyn)
```

```
    Fitting 3 folds for each of 6 candidates, totalling 18 fits
    GridSearchCV(cv=3, estimator=XGBClassifier(max_depth=2, n_estimators=200),
                 param_grid={'learning_rate': [0.2, 0.6],
                             'subsample': [0.3, 0.6, 0.9]},
                 return_train_score=True, scoring='roc_auc', verbose=1)
```

```
# cv results
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results
```

mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learı

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

param_grid = {'learning_rate': [0.2, 0.6],
              'subsample': [0.3, 0.6, 0.9]}


for n, subsample in enumerate(param_grid['subsample']):


    # subplot 1/n
    plt.subplot(1,len(param_grid['subsample']), n+1)
    df = cv_results[cv_results['param_subsample']==subsample]

    plt.plot(df["param_learning_rate"], df["mean_test_score"])
    plt.plot(df["param_learning_rate"], df["mean_train_score"])
    plt.xlabel('learning_rate')
    plt.ylabel('AUC')
    plt.title("subsample={0}".format(subsample))
    plt.ylim([0.60, 1])
    plt.legend(['test score', 'train score'], loc='upper left')
    plt.xscale('log')
```




```
model_cv.best_params_
```

```
{'learning_rate': 0.6, 'subsample': 0.3}
```

```
# chosen hyperparameters
```

```
parameters = {'learning_rate': 0.6,
          'max_depth': 2,
          'n_estimators':200,
          'subsample':0.3,
         'objective':'binary:logistic'}

# fit model on training data
xgb_bal_adasyn_model = XGBClassifier(params = parameters)
xgb_bal_adasyn_model.fit(X_train_adasyn, y_train_adasyn)
```

```
XGBClassifier(params={'learning_rate': 0.6, 'max_depth': 2, 'n_estimators': 200,
                      'objective': 'binary:logistic', 'subsample': 0.3})
```

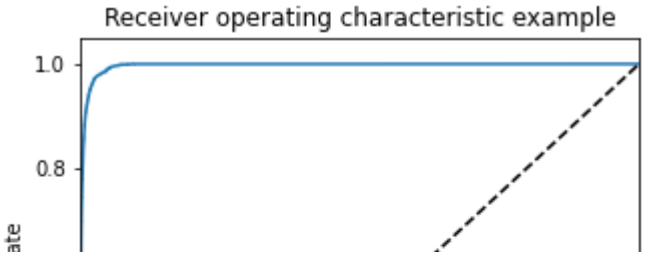
Prediction on the train setPrediction on the train set

```
# Predictions on the train set
y_train_pred = xgb_bal_adasyn_model.predict(X_train_adasyn)
printMetrics(y_train_adasyn,y_train_pred)
```

```
Confusion Matrix:
[[218342   8278]
 [  4248 222416]]

Accuracy: 0.9723661104296644
Sensitivity: 0.9812586030423888
Specificity: 0.9634718912717324
F1-Score: 0.9726122643530888

              precision    recall  f1-score   support

           0       0.98      0.96      0.97    226620
           1       0.96      0.98      0.97    226664

    accuracy                           0.97    453284
   macro avg       0.97      0.97      0.97    453284
weighted avg       0.97      0.97      0.97    453284
```

```
# Predicted probability
y_train_pred_proba = xgb_bal_adasyn_model.predict_proba(X_train_adasyn)[:,1]
draw_roc(y_train_adasyn, y_train_pred_proba)
```

```
auc_score: 0.9963895326516216
```



Prediction on the test set

```
# Predictions on the test set
y_test_pred = xgb_bal_adasyn_model.predict(X_test)
printMetrics(y_test,y_test_pred)
```

```
Confusion Matrix:
[[54442  2191]
 [    8   105]]

Accuracy: 0.961248369929158
Sensitivity: 0.9292035398230089
Specificity: 0.9613123090777462
F1-Score: 0.087173100871731

              precision    recall  f1-score   support

           0       1.00      0.96      0.98     56633
           1       0.05      0.93      0.09       113

    accuracy                           0.96     56746
   macro avg       0.52      0.95      0.53     56746
weighted avg       1.00      0.96      0.98     56746
```

```
# Predicted probability
y_test_pred_proba = xgb_bal_adasyn_model.predict_proba(X_test)[:,1]
draw_roc(y_test, y_test_pred_proba)
```

```
auc_score: 0.9770441699693836
```

```
print('Train auc =', metrics.roc_auc_score(y_train_smote, y_train_pred_proba_log_bal_smote))
fpr, tpr, thresholds = metrics.roc_curve(y_train_smote, y_train_pred_proba_log_bal_smote)
threshold = thresholds[np.argmax(tpr-fpr)]
print("Threshold=",threshold)
```

```
Train auc = 0.9893406429083762
Threshold= 0.4236902733825088
```

```
from sklearn.svm import SVC
clf=SVC()
clf.fit(X_train_adasyn, y_train_adasyn)
y_pred= clf.predict(X_test)
```



```
y_pred
## adasyn
```

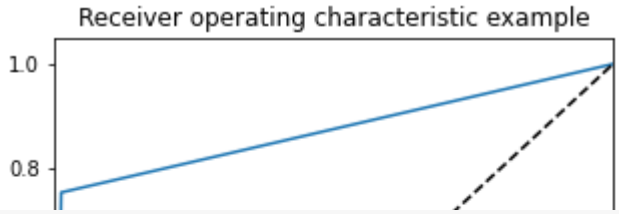
```
array([0, 0, 0, ..., 0, 0, 0])
```

```
printMetrics(y_test,y_pred)
```

```
Confusion Matrix:
[[55929   704]
 [   28    85]]

Accuracy: 0.987100412363867
Sensitivity: 0.7522123893805309
Specificity: 0.9875690851623612
F1-Score: 0.188470066518847

              precision    recall  f1-score   support

           0       1.00      0.99      0.99     56633
           1       0.11      0.75      0.19       113

    accuracy                           0.99     56746
   macro avg       0.55      0.87      0.59     56746
weighted avg       1.00      0.99      0.99     56746
```

```
draw_roc(y_test, y_pred)
```

```
auc_score: 0.8698907372714462
```



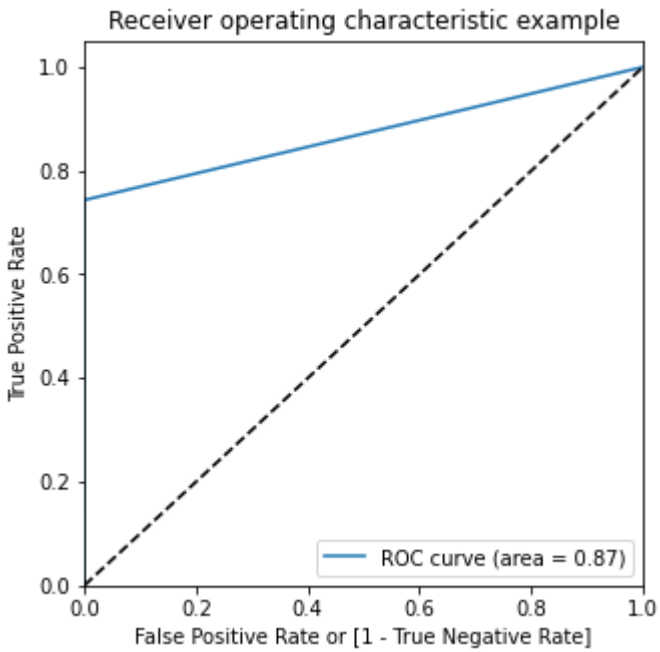

```
# Unbalanced
clf=SVC()
clf.fit(X_train, y_train)
y_pred= clf.predict(X_test)
```

```
printMetrics(y_test,y_pred)
```

```
Confusion Matrix:
[[56629     4]
 [   29    84]]

Accuracy: 0.9994184612131252
Sensitivity: 0.7433628318584071
Specificity: 0.9999293698020588
F1-Score: 0.8358208955223881

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56633
           1       0.95      0.74      0.84       113

    accuracy                           1.00     56746
   macro avg       0.98      0.87      0.92     56746
weighted avg       1.00      1.00      1.00     56746
```

```
draw_roc(y_test, y_pred)
```

```
auc_score: 0.871646100830233
```



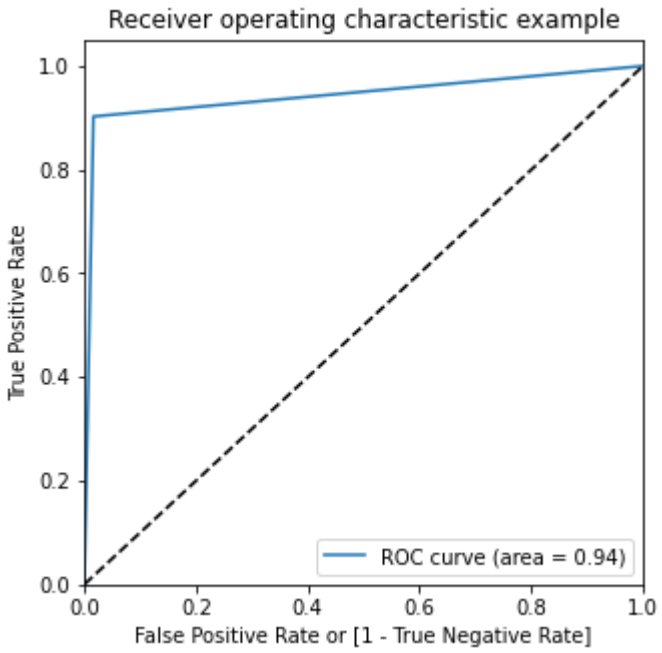
```
# Undersample
clf=SVC()
clf.fit(X_train_rus, y_train_rus)
y_pred_rus_svm= clf.predict(X_test)
```

```
printMetrics(y_test,y_pred_rus_svm)
```

```
    Confusion Matrix:
    [[55707   926]
     [   11   102]]

    Accuracy: 0.9834878229302506
    Sensitivity: 0.9026548672566371
    Specificity: 0.9836491091766285
    F1-Score: 0.17879053461875546

                  precision    recall  f1-score   support

               0       1.00      0.98      0.99     56633
               1       0.10      0.90      0.18       113

        accuracy                           0.98     56746
       macro avg       0.55      0.94      0.59     56746
    weighted avg       1.00      0.98      0.99     56746
```

```
draw_roc(y_test, y_pred_rus_svm)
```

```
    auc_score: 0.9431519882166329
```




```
# oversample
clf=SVC()
clf.fit(X_train_ros, y_train_ros)
y_pred_ross_svm= clf.predict(X_test)
```

```
# SMOTE
clf=SVC()
clf.fit(X_train_smote, y_train_smote)
```

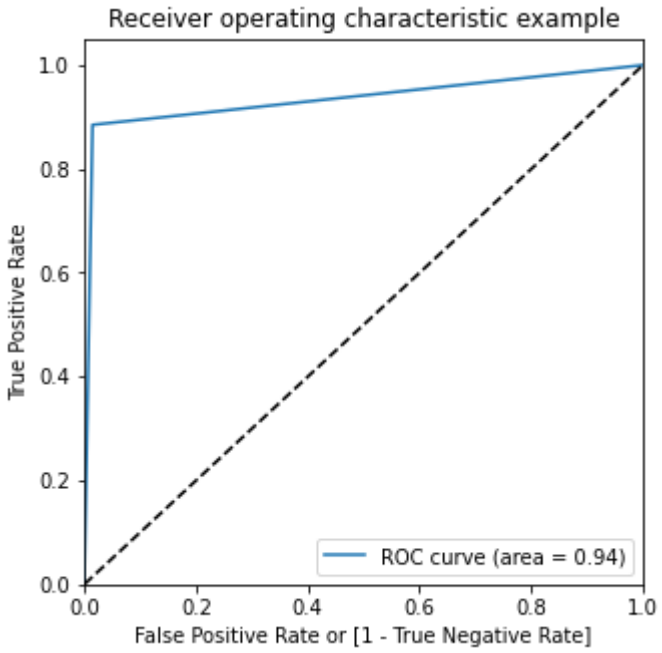
```
y_pred_smote_svm= clf.predict(X_test)
```

```
printMetrics(y_test,y_pred_ross_svm)
```

```
Confusion Matrix:
[[55799   834]
 [   13   100]]

Accuracy: 0.9850738378035456
Sensitivity: 0.8849557522123894
Specificity: 0.9852736037292744
F1-Score: 0.19102196752626552

              precision    recall  f1-score   support

           0       1.00      0.99      0.99     56633
           1       0.11      0.88      0.19       113

    accuracy                           0.99     56746
   macro avg       0.55      0.94      0.59     56746
weighted avg       1.00      0.99      0.99     56746
```

```
draw_roc(y_test, y_pred_ross_svm)
```

```
auc_score: 0.9351146779708319
```



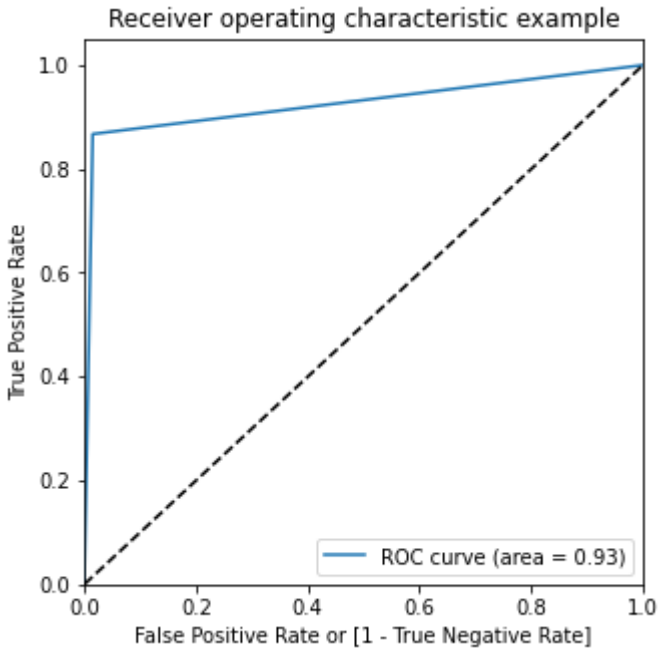

```
printMetrics(y_test,y_pred_smote_svm)
```

```
Confusion Matrix:
[[55781   852]
 [   15    98]]

Accuracy: 0.9847213900539245
Sensitivity: 0.8672566371681416
Specificity: 0.9849557678385393
F1-Score: 0.18438381937911572

              precision    recall  f1-score   support
```

```
              0       1.00      0.98      0.99     56633
              1       0.10      0.87      0.18       113

       accuracy                           0.98     56746
      macro avg       0.55      0.93      0.59     56746
   weighted avg       1.00      0.98      0.99     56746
```

```
draw_roc(y_test, y_pred_smote_svm)
```

```
auc_score: 0.9261062025033404
```




```
```