Sardar Patel Institute of Technology, Mumbai
Department of Electronics and Telecommunication Engineering
B.E. Sem-VII

ISE-2 ML Minor

Name: Sruthi Shivaramakrishnan Batch:A UID:2019110059 Branch: ETRX

Objective: Dealing with imbalanced dataset (Given on Kaggle data repository)

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where you may observe 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

- **Understand the imbalanced dataset and will perform various approaches like undersampling/ oversampling, choosing the right metrics of ROC- AUC to deal with the imbalanced dataset.**
- **After trying different approaches and training different models (LR, KNN, SVM, Random Forest, XGboost, Naive Bayes) you will compare their results and decide the one which fits best for your application.**

Platform : Google Colab
Data Set : Credit Card Fraud Detection
Code and output:

```
# Importing libraries

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

import pandas as pd


# Reading the dataset

df=pd.read_csv('/content/drive/MyDrive/creditcard.csv')

df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267

5 rows × 31 columns

Checking null values

df.isnull().sum()

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

Data has no values

df.shape

```
(284807, 31)
```

df.describe()

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.84807
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.40633
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.09863
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.34340
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.43097
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.14287
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.97139
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.55949

8 rows × 31 columns

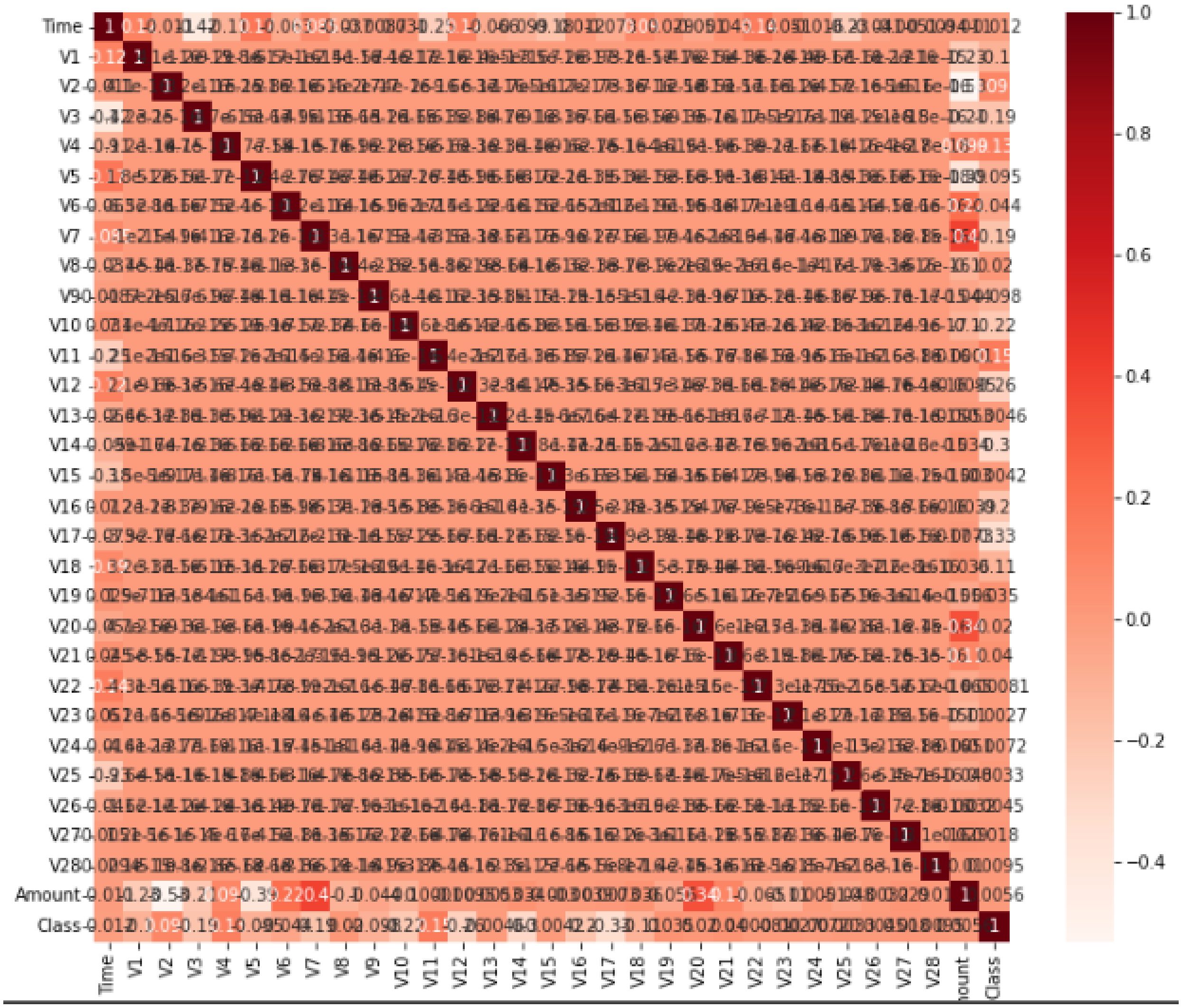
df.info()

```
 #   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  
```

import seaborn as sns

correlation=df.corr()
import matplotlib.pyplot as plt
plt.figure(figsize=(12,10))

sns.heatmap(correlation,annot=True,cmap=plt.cm.Reds) plt.show




```
correlation_mat = df.corr()
corr_pairs = correlation_mat.unstack()

sorted_pairs = corr_pairs.sort_values(kind="quicksort")
strong_pairs = sorted_pairs[abs(sorted_pairs) > 0.5]
print(strong_pairs)
```

```
V2      Amount   -0.531409
Amount  V2       -0.531409
Time    Time      1.000000
V15     V15       1.000000
V28     V28       1.000000
V1      V1        1.000000
V2      V2        1.000000
V3      V3        1.000000
V4      V4        1.000000
V5      V5        1.000000
V6      V6        1.000000
V7      V7        1.000000
V8      V8        1.000000
V9      V9        1.000000
V10     V10       1.000000
V11     V11       1.000000
V12     V12       1.000000
Amount  Amount    1.000000
V13     V13       1.000000
V16     V16       1.000000
V17     V17       1.000000
V18     V18       1.000000
V19     V19       1.000000
V20     V20       1.000000
V21     V21       1.000000
V22     V22       1.000000
V23     V23       1.000000
V24     V24       1.000000
V25     V25       1.000000
V26     V26       1.000000
V27     V27       1.000000
V14     V14       1.000000
Class   Class     1.000000
dtype: float64
```

df.columns

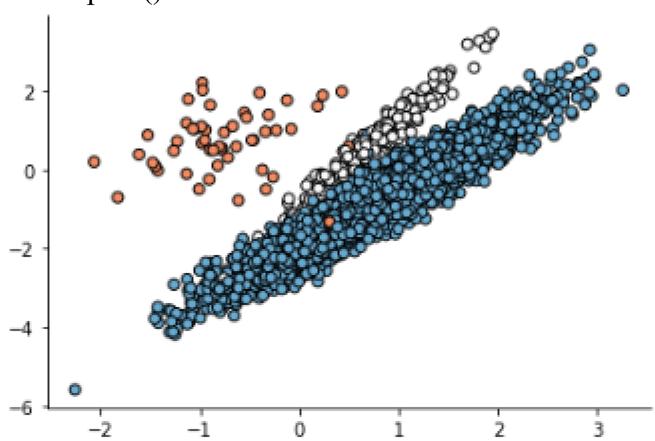
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

x=df.iloc[:, 0:30].values
y=df['Class']

#Splitting into train and test set

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)

```
import seaborn as sns
from sklearn.datasets import make_classification

x, y = make_classification(n_samples=5000, n_features=2, n_informative=2,
                           n_redundant=0, n_repeated=0, n_classes=3,
                           n_clusters_per_class=1,
                           weights=[0.01, 0.05],
                           class_sep=0.8, random_state=0)

import matplotlib.pyplot as plt
colors = ['#ef8a62' if v == 0 else '#f7f7f7' if v == 1 else '#67a9cf' for v in y]
kwarg_params = {'linewidth': 1, 'edgecolor': 'black'}
fig = plt.Figure(figsize=(12,6))
plt.scatter(x[:, 0], x[:, 1], c=colors, **kwarg_params)
sns.despine()
```



The above plot shows the bias and imbalance of the data points. The fraud class accound for only a small portion of the data

```
#Applying the XCG algorithm on the given data

# import linrary
from xgboost import XGBClassifier

xgb_model = XGBClassifier().fit(x_train, y_train)

# predict
xgb_y_predict = xgb_model.predict(x_test)
```

```
from sklearn.metrics import accuracy_score,precision_score,recall_score

xgb_score = accuracy_score(xgb_y_predict, y_test)

print('Accuracy score is:', xgb_score)
```

```
Accuracy score is: 0.9904
```

The above code shows high accuracy. The reason for this is high imbalance in the data. Though the accuracy is high, because of the imbalance it might not be able to perform well in world cases of fraud detection.

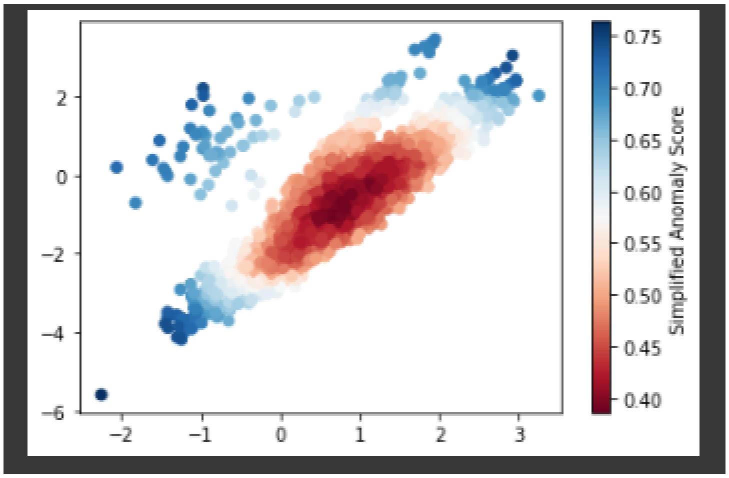
```
class_count_0, class_count_1 = df['Class'].value_counts()

# Separate class
class_0 = df[df['Class'] == 0]
class_1 = df[df['Class'] == 1]# print the shape of the class
print('class 0:', class_0.shape)
print('class 1:', class_1.shape)
```

```
class 0: (284315, 33)
class 1: (492, 33)
```

The above snippet shows the number of data points in each class.

```
#Isolation forest to detect anomalies

import numpy as np
from sklearn.ensemble import IsolationForest

random_state = np.random.RandomState(42)
model=IsolationForest(n_estimators=100,max_samples='auto',contamination=float(0.2),random_
state=random_state)

model.fit(x)
pred = model.predict(x)

pred_scores = -1*model.score_samples(x)

print(model.get_params())

plt.scatter(x[:, 0], x[:, 1], c=pred_scores, cmap='RdBu')
```

plt.colorbar(label='Simplified Anomaly Score')
plt.show()



The above snippet shows other anomalies in the data. The large anomalies may be due to the imbalance.

Over sampling using SMOTE

# import library
from imblearn.over_sampling import SMOTE
from collections import Counter

smote = SMOTE()

fit predictor and target variable
x_smote, y_smote = smote.fit_resample(x, y)

print('Original dataset shape', Counter(y))
print('Resample dataset shape', Counter(y_smote))

```
Original dataset shape Counter({0: 284315, 1: 492})
Resample dataset shape Counter({0: 284315, 1: 284298})
```

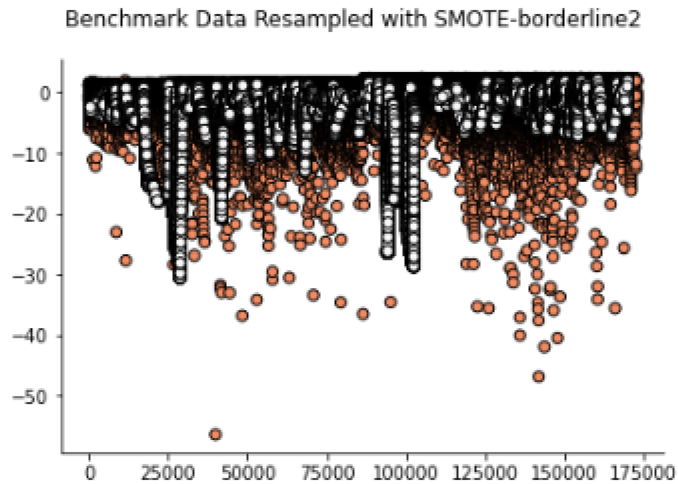
To increase the number of fraud cases we are using SMOTE to increase the number of samples of the imbalanced data.

Over sampling using Adasyn

from imblearn.over_sampling import ADASYN

ada = ADASYN(random_state=42)

```
X_res, y_res = ada.fit_resample(x, y)
kwarg_params = {'linewidth': 1, 'edgecolor': 'black'}
colors = ['#ef8a62' if v == 0 else '#f7f7f7' if v == 1 else '#67a9cf' for v in y_res]
plt.scatter(X_res[:, 0], X_res[:, 1], c=colors, **kwarg_params)
sns.despine()
plt.suptitle("Benchmark Data Resampled with SMOTE-borderline2")
pass
```



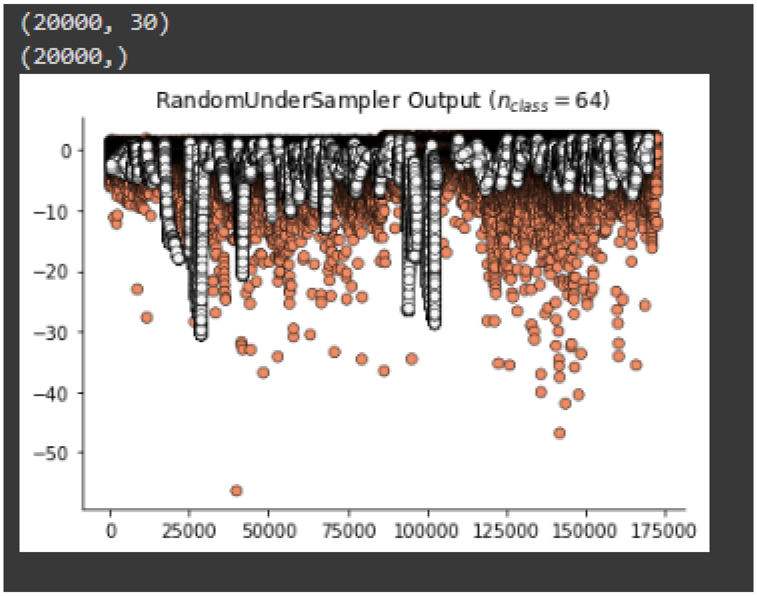

```
print('Original dataset shape', Counter(y))
print('Resample dataset shape', Counter(y_res))
```

```
Original dataset shape Counter({0: 284315, 1: 492})
Resample dataset shape Counter({0: 284315, 1: 284298})
```

```
# Under sampling

# Random Under Sampling for Adasyn data

from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(random_state=0, sampling_strategy={0: 10000, 1: 10000})
rus.fit(X_res, y_res)
X_rus, y_rus = rus.fit_resample(X_res, y_res)
print(X_rus.shape)
print(y_rus.shape)
```

```
colors = ['#ef8a62' if v == 0 else '#f7f7f7' if v == 1 else '#67a9cf' for v in y_res]
plt.scatter(X_res[:, 0], X_res[:, 1], c=colors, linewidth=0.5, edgecolor='black')
```

sns.despine()
plt.title("RandomUnderSampler Output ($n_{class}=64)$")
pass

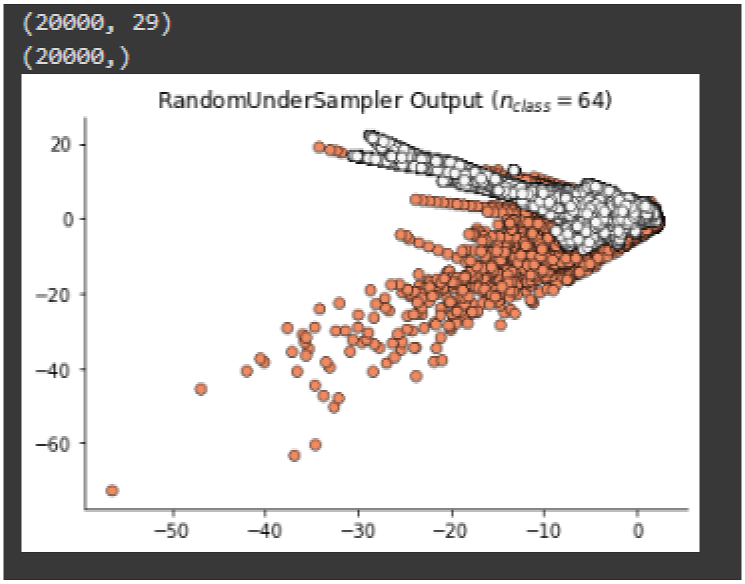


Random Under sampling of SMOTE data

from imblearn.under_sampling import RandomUnderSampler

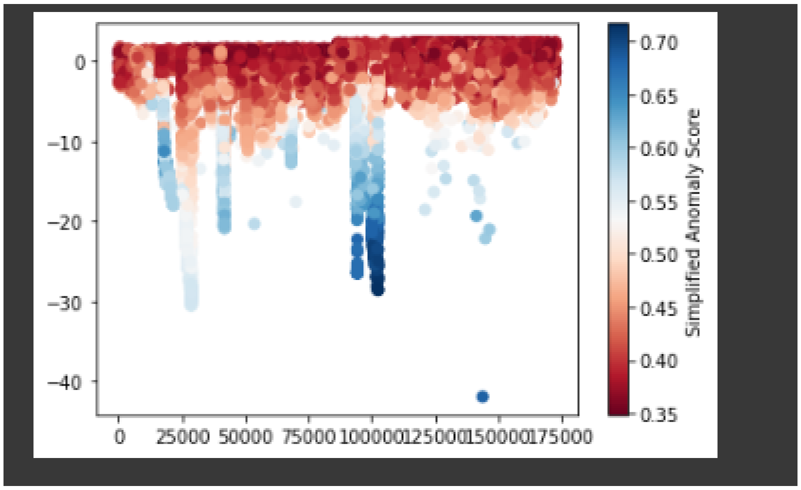
rus = RandomUnderSampler(random_state=0, sampling_strategy={0: 10000, 1: 10000})
rus.fit(x_smote, y_smote)
x_rus1, y_rus1 = rus.fit_resample(x_smote, y_smote)
print(x_rus1.shape)
print(y_rus1.shape)

colors = ['#ef8a62' if v == 0 else '#f7f7f7' if v == 1 else '#67a9cf' for v in y_smote]
plt.scatter(x_smote[:, 0], x_smote[:, 1], c=colors, linewidth=0.5, edgecolor='black')
sns.despine()
plt.title("RandomUnderSampler Output ($n_{class}=64)$")
pass



Comparing the under sampled data for both the algorithms we can see that the Adasyn undersampled data has more balance compared to the SMOTE undersampled data.

```
# Isolation Forest for anomaly detection

import numpy as np
from sklearn.ensemble import IsolationForest

random_state = np.random.RandomState(42)
model=IsolationForest(n_estimators=100,max_samples='auto',contamination=float(0.179),random_state=random_state)

model.fit(X_rus)
pred1 = model.predict(X_rus)

pred_scores1 = -1*model.score_samples(X_rus)

print(model.get_params())

plt.scatter(X_rus[:, 0], X_rus[:, 1], c=pred_scores1, cmap='RdBu')
plt.colorbar(label='Simplified Anomaly Score')
plt.show()
```



The above plot shows lesser anomalies for the sampled data compared to the original data. Thus improving the distribution and balance of the data points.

```
class1_count_0, class1_count_1 = y_rus.value_counts()

# Separate class
class1_0 = y_rus[y_rus == 0]
class1_1 = y_rus[y_rus == 1]# print the shape of the class
print('class 0:', class1_0.shape)
print('class 1:', class1_1.shape)
```

```
class 0: (10000,)
class 1: (10000,)
```

```
# Splitting into train and test for Adasyn data

from sklearn.model_selection import train_test_split
x_train1, x_test1, y_train1, y_test1= train_test_split(X_rus, y_rus, test_size= 0.25,
random_state=0)


# Splitting into train and test for Smote data

from sklearn.model_selection import train_test_split
x_train2, x_test2, y_train2, y_test2= train_test_split(x_rus1, y_rus1, test_size= 0.25,
random_state=0)

# Random Forest

from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
```

```
classifier1= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_train1, y_train1)
classifier1.fit(x_train2, y_train2)

y_pred= classifier.predict(x_test1)
y_pred_smote=classifier1.predict(x_test2)

#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test1, y_pred)
print(cm)
```

```
[[2473    4]
 [  31 2492]]
```

```
from sklearn.metrics import precision_score, recall_score,accuracy_score
print(accuracy_score(y_test1, y_pred))
print(precision_score(y_test1, y_pred))
print(recall_score(y_test1, y_pred))
```

```
0.993
0.9983974358974359
0.9877130400317082
```

The above snippet shows accuracy for Adasyn data

```
from sklearn.metrics import precision_score, recall_score,accuracy_score
print(accuracy_score(y_test2, y_pred_smote))
print(precision_score(y_test2, y_pred_smote))
print(recall_score(y_test2, y_pred_smote))
```

```
0.9912
0.9963956748097718
0.9861276258422513
```

The above shows accuracy for smote data

The above two plots show adasyn has a higher accuracy compared to smote. Hence adasyn performs better compared to smote. We will be using Adasyn for all algorithms henceforth.

```
# KNN

#Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=24, metric='minkowski', p=2 )
classifier.fit(x_train1, y_train1)
```

```
y_pred1= classifier.predict(x_test1)

from sklearn.metrics import precision_score, recall_score,accuracy_score
print(accuracy_score(y_test1, y_pred1))
print(precision_score(y_test1, y_pred1))
print(recall_score(y_test1, y_pred1))
```

```
0.872
0.9906201146430432
0.7534680935394372
```

```
# Naive Bayes

from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(x_train1, y_train1)

# making predictions on the testing set
y_pred2 = clf.predict(x_test1)


from sklearn.metrics import precision_score, recall_score,accuracy_score
print(accuracy_score(y_test1, y_pred2))
print(precision_score(y_test1, y_pred2))
print(recall_score(y_test1, y_pred2))

# SVM

from sklearn.svm import SVC
svclassifier = SVC(kernel='linear')
clf=svclassifier.fit(x_train1, y_train1)

y_pred3 = svclassifier.predict(x_test1)

from sklearn.metrics import precision_score, recall_score,accuracy_score
print(accuracy_score(y_test1, y_pred3))
print(precision_score(y_test1, y_pred3))
print(recall_score(y_test1, y_pred3))
```

```
0.9452
0.9870073624945864
0.9032897344431232
```

```
# Logistic regression
```

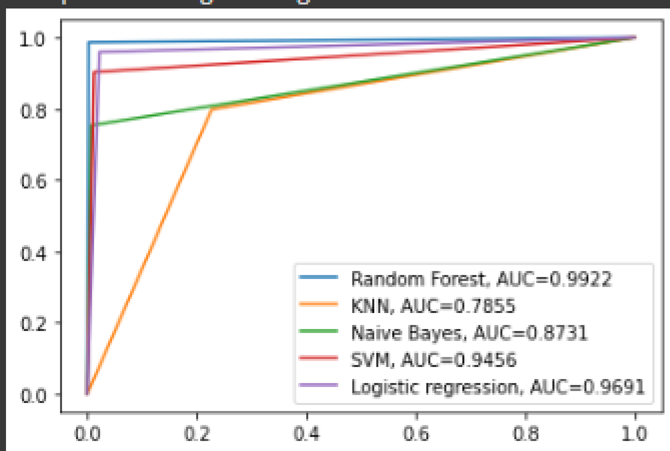
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression


classifier = LogisticRegression(random_state=0)
classifier.fit(x_train1, y_train1)
y_pred4 = classifier.predict(x_test1)
confusion_matrix = confusion_matrix(y_test1, y_pred4)

print(confusion_matrix)

print('Score: {:.2f}'.format(classifier.score(x_test1,y_test1)))
```

```
[[2423   54]
 [ 101 2422]]
Score: 0.97
```

```
# AUC ROC plots with threshhold 0.5

from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
fpr, tpr, _ = metrics.roc_curve(y_test1,  y_pred)
auc = round(metrics.roc_auc_score(y_test1,  y_pred), 4)
plt.plot(fpr,tpr,label="Random Forest, AUC="+str(auc))

fpr, tpr, _ = metrics.roc_curve(y_test1,  y_pred1)
auc = round(metrics.roc_auc_score(y_test1,  y_pred1), 4)
plt.plot(fpr,tpr,label="KNN, AUC="+str(auc))


fpr, tpr, _ = metrics.roc_curve(y_test1,  y_pred2)
auc = round(metrics.roc_auc_score(y_test1,  y_pred2), 4)
plt.plot(fpr,tpr,label="Naive Bayes, AUC="+str(auc))

fpr, tpr, _ = metrics.roc_curve(y_test1,  y_pred3)
auc = round(metrics.roc_auc_score(y_test1,  y_pred3), 4)
plt.plot(fpr,tpr,label="SVM, AUC="+str(auc))

fpr, tpr, _ = metrics.roc_curve(y_test1,  y_pred4)
```

```
auc = round(metrics.roc_auc_score(y_test1,  y_pred4), 4)
plt.plot(fpr,tpr,label="Logistic regression, AUC="+str(auc))
#add legend
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f47b2b52610>
```



Inference:

1. The accuracy for different models are as follows

Random Forest- 99.3%
KNN- 78.56%
Naive Bayes - 87.2%
SVM - 94.52%
Logistic regression- 97%

2. Thus the highest accuracy occurs for random forest algorithm.
3. Looking at the AUC ROC plot for each graphs we can see that the data shows best results for Random Forest followed by Logistic regression and SVM
4. The above results for Adasyn and Smote up sampling data shows Adasyn works better for this dataset.
5. The imbalanced data is up sampled to balance and down sampled to reduce the size and computation power.
The down sampling process reduces the accuracy by a very marginal value but provides higher computer power, hence it is preferred when the data size is huge.