Double-click (or enter) to edit

```
Abhishek Patwardhan

D17A - 57

ADS Experiment 5
```

Abhishek Patwardhan

D17A - 57

ADS Experiment 5

==> Importing Dependencies

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
from google.colab import files
import io
```

==> Loading the Dataset

```
# uploaded = files.upload()
df = pd.read_csv('creditcard.csv')
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098
[illegible]	[illegible]	[illegible]	[illegible]	.166480	0.448154	0.060018	-0.082361	-0.078803	0.085
[illegible]	[illegible]	[illegible]	[illegible]	.773209	0.379780	-0.503198	1.800499	0.791461	0.247
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270

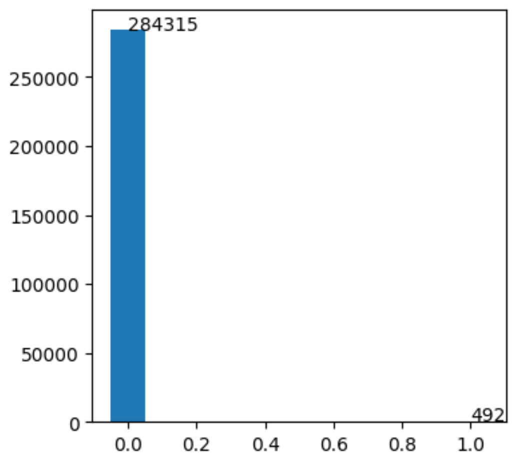
5 rows × 31 columns

Saved successfully!

Exploratory Data Analysis

```
plt.figure(figsize=(4,4))
plt.bar(df['Class'].unique(), df['Class'].value_counts(), width = 0.1)
for i, v in enumerate(df['Class'].value_counts()):
  plt.text(i, v, str(v), ha='left')
plt.figure(figsize=(16,5))
```

```
<Figure size 1600x500 with 0 Axes>
```




```
<Figure size 1600x500 with 0 Axes>
```

Implementing Oversampling using SMOTE

```
from imblearn.over_sampling import SMOTE
```

```
# check version number
import imblearn
print(imblearn.__version__)
```

```
0.10.1
```

```
X = df.iloc[:,:-1]
y = df.iloc[:,30:]
```

Oversampling

SMOTE works by selecting examples that are close in the feature space, drawing a linebetween the examples in the feature space and drawing a new sample at a pointalong that line.

```
oversample = SMOTE()
X, y = oversample.fit_resample(X, y)
```

Balanced Value Counts

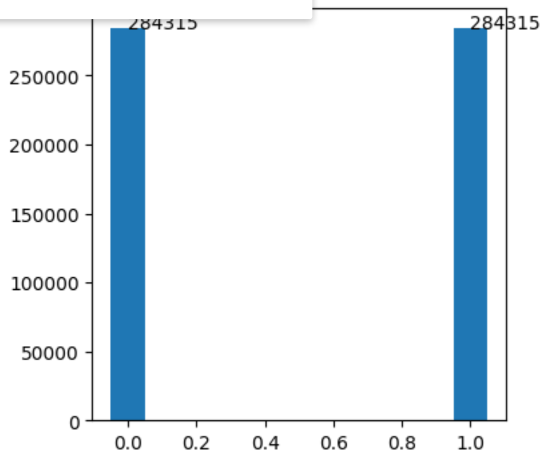
The count of Class jumps from 492 to 284315

```
y.value_counts()
```

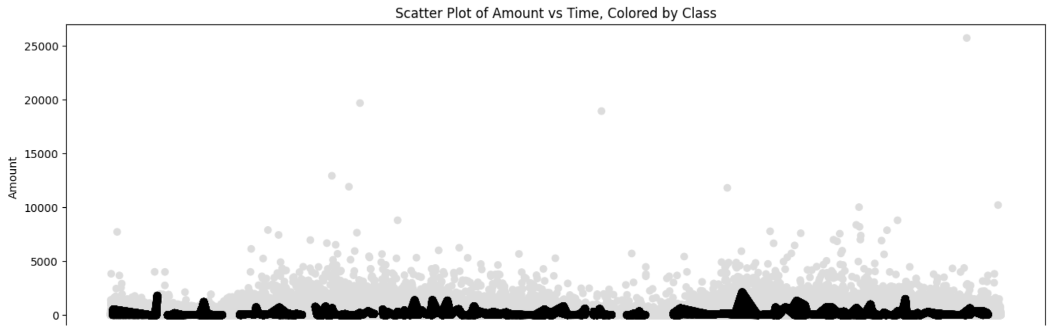
```
Class
0        284315
1        284315
dtype: int64
```

```
plt.figure(figsize=(4,4))
plt.bar(y['Class'].unique(), y['Class'].value_counts(), width = 0.1)
for i, v in enumerate(y['Class'].value_counts()):
```

Saved successfully!




```
plt.figure(figsize=(16,5))
# Create a color map for the Class values
color_map = {0: '#dcdcdc', 1: 'black'}
# Create a scatter plot of Amount vs Time, with color based on Class
plt.scatter(X['Time'], X['Amount'], c=y['Class'].map(color_map))
# Set the axis labels and title
plt.xlabel('Time')
plt.ylabel('Amount')
plt.title('Scatter Plot of Amount vs Time, Colored by Class')
# Display the plot
plt.show()
```



Issue

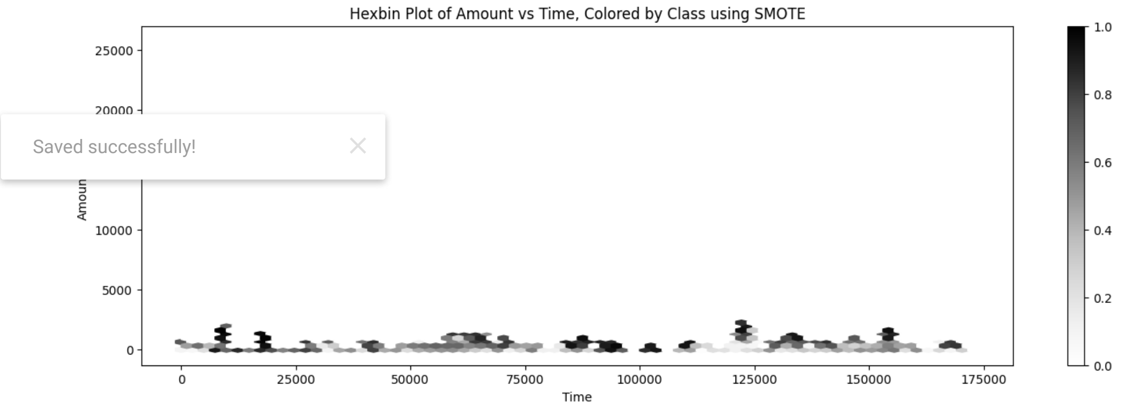
Here, two tuples may have overlapping results, in the sense, for the same attributes, the outcome would be both 0 and 1

Hypothesis testing

1. Visualising the overlapping of Oversampled points using a Hexbin Plot

```
plt.figure(figsize=(16,5))
plt.hexbin(x=X['Time'], y=X['Amount'], C=y['Class'], gridsize=70, cmap='Greys')
plt.xlabel('Time')
plt.ylabel('Amount')
plt.title('Hexbin Plot of Amount vs Time, Colored by Class using SMOTE')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f32c3f468e0>
```



Solution

User Borderline-SMOTE SVM

Implementing Borderline-SMOTE SVM

```
from imblearn.over_sampling import SVMSMOTE
```

Hien Nguyen, et al. suggest using an alternative of Borderline-SMOTE where an SVM algorithm is used instead of a KNN to identify misclassified examples on the decision boundary. An SVM is used to locate the decision boundary defined by the support vectors and examples in the minority class that close to the support vectors become the focus for generating synthetic examples

Instance of SVMSMOTE()

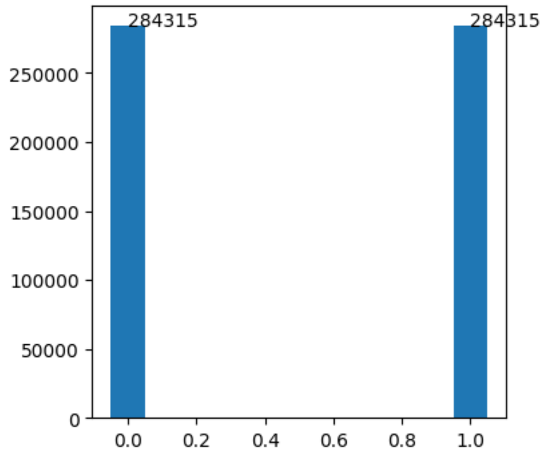
```
# transform the dataset
svmoversample = SVMSMOTE()
```

```
X2 = df.iloc[:,:-1]
y2 = df.iloc[:,30:]
```

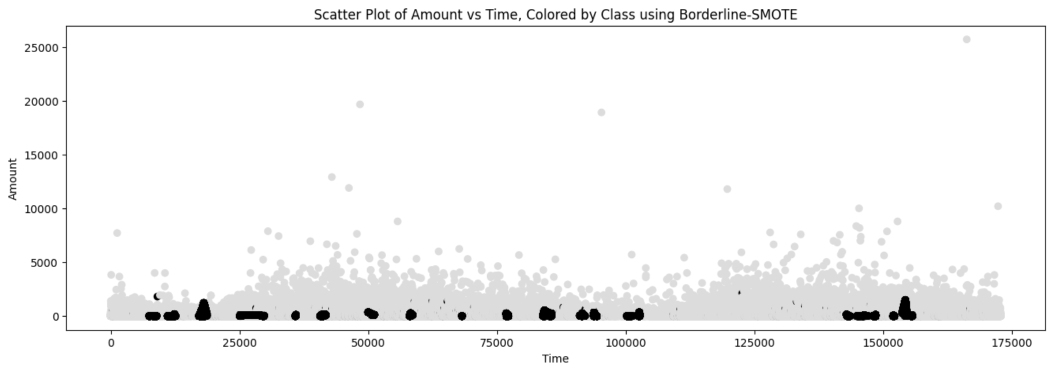
```
%%time
X2, y2 = svmoversample.fit_resample(X2, y2)
```

```
CPU times: user 25.6 s, sys: 770 ms, total: 26.3 s
Wall time: 25.5 s
```

```
plt.figure(figsize=(4,4))
plt.bar(y2['Class'].unique(), y2['Class'].value_counts(), width = 0.1)
for i, v in enumerate(y['Class'].value_counts()):
  plt.text(i, v, str(v), ha='left')
```




```
plt.figure(figsize=(16,5))
# Create a color map for the Class values
color_map = {0: '#dcdcdc', 1: 'black'}
Saved successfully!                     Time, with color based on Class
                                       , c=y2['Class'].map(color_map))
# Set the axis labels and title
plt.xlabel('Time')
plt.ylabel('Amount')
plt.title('Scatter Plot of Amount vs Time, Colored by Class using Borderline-SMOTE')
# Display the plot
plt.show()
```



Visualising the overlapping of Oversampled points using a Hexbin Plot

The resulting plot will show the hexagonal bins, with the color of each bin indicating the density of points in that region. The overlap between the classes will be highlighted by the color of the overlapping bins.

plt.figure(figsize=(16,5)) plt.hexbin(x=X2['Time'], y=X2['Amount'], C=y2['Class'], gridsize=70, cmap='Grey') plt.xlabel('Time') plt.ylabel('Amount') plt.title('Hexbin Plot of Amount vs Time, Colored by Class using Borderline-SMOTE SVM') plt.colorbar()

ROC-AUC Evaluation

```
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.tree import DecisionTreeClassifier
```

```
X_base = df.iloc[:,:-1]
y_base = df.iloc[:,30:]
```

```
%%time
model = DecisionTreeClassifier() # define model
model.fit(X_base, y_base)
base_scores = cross_val_score(model, X_base, y_base, cv=10, scoring='roc_auc')
print('Mean ROC AUC for Unbalanced Data: %.3f' % mean(base_scores))
```

```
Mean ROC AUC for Unbalanced Data: 0.798
CPU times: user 4min 49s, sys: 358 ms, total: 4min 49s
Wall time: 4min 50s
```

Saved successfully!

```
%%time
model = DecisionTreeClassifier() # define model
model.fit(X, y)
smote_scores = cross_val_score(model, X, y, cv=10, scoring='roc_auc')
print('Mean ROC AUC for Balanced data using SMOTE: %.3f' % mean(smote_scores))
```

```
Mean ROC AUC for Balanced data using SMOTE: 0.948
CPU times: user 12min 21s, sys: 1.05 s, total: 12min 22s
Wall time: 12min 22s
```

Balanced Data Using Balanced SMOTE

```
%%time
model = DecisionTreeClassifier() # define model
model.fit(X2, y2)
border_smote_scores = cross_val_score(model, X2, y2, scoring='roc_auc', cv=10)
print('Mean ROC AUC for Balanced data using Borderline-SMOTE SVM: %.3f' % mean(border_smote_scores))
```

```
Mean ROC AUC for Balanced data using Borderline-SMOTE SVM: 0.915
CPU times: user 9min 46s, sys: 639 ms, total: 9min 47s
Wall time: 9min 48s
```