UnderSampling and OverSampling

While applying classification algorithms, if one of the class of target variable has much more observations than the other classes then, our model is to be biased for the class with majority samples. We use sampling techniques to deal with the imbalanced data.

Oversampling and undersampling in data analysis are techniques used to adjust the class distribution of a data set (i.e. the ratio between the different classes/categories represented).

Random Under Sampling is removing observations of the majority class until it is same as classes with minor observations.

Oversampling can be defined as adding more copies to the minority class. Oversampling can be a good choice when you don't have a ton of data to work with.

Decision Trees.

Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

1. **Import required libraries**
2. **Importing the data set into DataFrame.**
3. **Analyzing the data,its columns and types.**
4. **Converting data into target and feature variables.**
5. **Undersample the data.**
6. **Dividing data into training and testing.**
7. **Applying Decision Trees Algorithm to train madel on training data.**
8. **Conclusion.**

Importing Libraries

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import imblearn
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
%matplotlib inline
```

Reading Data in Data Frame

In [2]:

```
Data = pd.read_csv('biddings.csv')
```

Data Analysis

In [3]:

```
Data.head()
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	...	79	80	81	82	83	84	85	86	87	convert
0	-0.01	-0.43	2.22	-0.59	0.80	0.21	-0.19	0.30	-0.25	0.42	...	-0.07	0.30	-0.19	0.61	-0.04	0.36	-0.18	-0.24	0.07	0
1	0.00	-4.11	1.48	0.92	-7.37	0.60	0.37	1.95	0.19	0.85	...	-0.57	-0.27	1.17	-0.52	-1.43	-0.24	-0.41	0.71	-0.22	0
2	0.01	-5.03	-2.78	-0.83	0.92	0.46	0.10	1.36	-0.74	1.62	...	-0.13	0.33	0.38	0.41	-0.61	0.24	-0.10	-1.19	0.37	0
3	-0.02	1.98	2.30	0.87	-7.09	0.36	0.16	1.79	0.23	0.26	...	0.07	0.41	-0.81	-0.49	0.07	-0.20	-0.31	-0.55	0.18	0
4	-0.01	0.14	-2.43	-0.68	0.73	-1.47	0.68	1.93	-0.35	3.12	...	-0.04	0.03	0.07	-0.05	-0.02	-0.01	0.00	-0.20	0.07	0

5 rows × 89 columns

Analysing Data info

In [4]:

```
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 89 columns):
0          1000000 non-null float64
1          1000000 non-null float64
2          1000000 non-null float64
3          1000000 non-null float64
4          1000000 non-null float64
5          1000000 non-null float64
```

```
5           1000000 non-null float64
6           1000000 non-null float64
7           1000000 non-null float64
8           1000000 non-null float64
9           1000000 non-null float64
10          1000000 non-null float64
11          1000000 non-null float64
12          1000000 non-null float64
13          1000000 non-null float64
14          1000000 non-null float64
15          1000000 non-null float64
16          1000000 non-null float64
17          1000000 non-null float64
18          1000000 non-null float64
19          1000000 non-null float64
20          1000000 non-null float64
21          1000000 non-null float64
22          1000000 non-null float64
23          1000000 non-null float64
24          1000000 non-null float64
25          1000000 non-null float64
26          1000000 non-null float64
27          1000000 non-null float64
28          1000000 non-null float64
29          1000000 non-null float64
30          1000000 non-null float64
31          1000000 non-null float64
32          1000000 non-null float64
33          1000000 non-null float64
34          1000000 non-null float64
35          1000000 non-null float64
36          1000000 non-null float64
37          1000000 non-null float64
38          1000000 non-null float64
39          1000000 non-null float64
40          1000000 non-null float64
41          1000000 non-null float64
42          1000000 non-null float64
43          1000000 non-null float64
44          1000000 non-null float64
45          1000000 non-null float64
46          1000000 non-null float64
47          1000000 non-null float64
48          1000000 non-null float64
49          1000000 non-null float64
50          1000000 non-null float64
51          1000000 non-null float64
52          1000000 non-null float64
53          1000000 non-null float64
54          1000000 non-null float64
55          1000000 non-null float64
56          1000000 non-null float64
57          1000000 non-null float64
58          1000000 non-null float64
59          1000000 non-null float64
60          1000000 non-null float64
61          1000000 non-null float64
62          1000000 non-null float64
63          1000000 non-null float64
64          1000000 non-null float64
65          1000000 non-null float64
66          1000000 non-null float64
67          1000000 non-null float64
68          1000000 non-null float64
69          1000000 non-null float64
70          1000000 non-null float64
71          1000000 non-null float64
72          1000000 non-null float64
73          1000000 non-null float64
74          1000000 non-null float64
75          1000000 non-null float64
76          1000000 non-null float64
77          1000000 non-null float64
78          1000000 non-null float64
79          1000000 non-null float64
80          1000000 non-null float64
81          1000000 non-null float64
82          1000000 non-null float64
83          1000000 non-null float64
84          1000000 non-null float64
85          1000000 non-null float64
86          1000000 non-null float64
87          1000000 non-null float64
convert     1000000 non-null int64
dtypes: float64(88), int64(1)
memory usage: 679.0 MB
```

Converting Data into features and target variable.

In [5]:

```
X = Data.drop('convert', axis = 1)
Y = Data.iloc[:,88]
```

Analysing Target variable

In [6]:

```
Data.loc[:,'convert'].value_counts()
```

Out[6]:

```
0    998092
1      1908
Name: convert, dtype: int64
```

The '0' class has 998092 samples while '1' class has only 1908 samples.

Since we have a lot of data, undersampling is a good choice in this situation.

In [7]:

```
Sampler = RandomUnderSampler(random_state = 0, replacement = False)
```

In [8]:

```
X, Y = Sampler.fit_resample(X,Y)
```

Since there are 87 number of columns and we don't know what each column is base on, we cannot visualize the data this big properly, and there is no need to visualize the data.

Now dividing the balanced data into training and testing data.

In [9]:

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)
```

Now training the decision tree model using trianing data.

In [10]:

```
model = DecisionTreeClassifier(random_state = 0)
```

Fitting the model with training data.

In [11]:

```
model.fit(x_train,y_train)
```

Out[11]:

```
DecisionTreeClassifier(random_state=0)
```

Predicting values of the test data.

In [12]:

```
y_pred = model.predict(x_test)
y_pred
```

Out[12]:

```
array([1, 1, 1, ..., 1, 1, 0], dtype=int64)
```

Calculating accuracy of model on test data.

In [13]:

```
model.score(x_test, y_test)
```

Out[13]:

```
0.5606986899563319
```

Model has the accuracy of 56.07% on the testing data.

In [ ]:

```

```