This dataset was got from Kaggle website and the dataset contains transactions made by credit cards in September 2013 by European cardholders.

- This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.
- It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'.
- Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.
- Given the class imbalance ratio, we recommend measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC). Confusion matrix accuracy is not meaningful for unbalanced classification.
- Update (03/05/2021)

importing the library

In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

The code is showing all the numbers of clumus

In [2]:
```
pd.pandas.set_option('display.max_columns', None)
```

Importing and showing the head of data

In [3]:
```
crd=pd.read_csv('creditcard.csv')
crd.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.311
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.143
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.165
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.287
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.119

knowing the reo and columus

In [4]:
```
crd.shape
```

Out[4]:
```
(284806, 31)
```

Getting general information

In [5]:
```
crd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284806 entries, 0 to 284805
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284806 non-null  float64
 1   V1      284806 non-null  float64
 2   V2      284806 non-null  float64
 3   V3      284806 non-null  float64
 4   V4      284806 non-null  float64
```

```
 5   V5      284806 non-null  float64
 6   V6      284806 non-null  float64
 7   V7      284806 non-null  float64
 8   V8      284806 non-null  float64
 9   V9      284806 non-null  float64
 10  V10     284806 non-null  float64
 11  V11     284806 non-null  float64
 12  V12     284806 non-null  float64
 13  V13     284806 non-null  float64
 14  V14     284806 non-null  float64
 15  V15     284806 non-null  float64
 16  V16     284806 non-null  float64
 17  V17     284806 non-null  float64
 18  V18     284806 non-null  float64
 19  V19     284806 non-null  float64
 20  V20     284806 non-null  float64
 21  V21     284806 non-null  float64
 22  V22     284806 non-null  float64
 23  V23     284806 non-null  float64
 24  V24     284806 non-null  float64
 25  V25     284806 non-null  float64
 26  V26     284806 non-null  float64
 27  V27     284806 non-null  float64
 28  V28     284806 non-null  float64
 29  Amount  284806 non-null  float64
 30  Class   284806 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

checking for missing value

In [6]:

```
crd.isna().any()
```

Out[6]:

```
Time      False
V1        False
V2        False
V3        False
V4        False
V5        False
V6        False
V7        False
V8        False
```

```
V9        False
V10       False
V11       False
V12       False
V13       False
V14       False
V15       False
V16       False
V17       False
V18       False
V19       False
V20       False
V21       False
V22       False
V23       False
V24       False
V25       False
V26       False
V27       False
V28       False
Amount    False
Class     False
dtype: bool
```

knowing the data types

In [7]:
```
crd.dtypes.value_counts()
```

Out[7]:
```
float64    30
int64       1
dtype: int64
```

checking for Duplicate columus be habit to check for duplicate values

In [8]:
```
crd.duplicated(keep ='first').sum()
```

Out[8]:
```
1081
```

In [9]:
```
crd[crd.duplicated()]
```

Out[9]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13
33	26.0	-0.529912	0.873892	1.347247	0.145457	0.414209	0.100223	0.711206	0.176066	-0.286717	-0.484688	0.872490	0.851636	-0.571745
35	26.0	-0.535388	0.865268	1.351076	0.147575	0.433680	0.086983	0.693039	0.179742	-0.285642	-0.482474	0.871800	0.853447	-0.571822
113	74.0	1.038370	0.127486	0.184456	1.109950	0.441699	0.945283	-0.036715	0.350995	0.118950	-0.243289	0.578063	0.674730	-0.534231
114	74.0	1.038370	0.127486	0.184456	1.109950	0.441699	0.945283	-0.036715	0.350995	0.118950	-0.243289	0.578063	0.674730	-0.534231
115	74.0	1.038370	0.127486	0.184456	1.109950	0.441699	0.945283	-0.036715	0.350995	0.118950	-0.243289	0.578063	0.674730	-0.534231
...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
282986	171288.0	1.912550	-0.455240	-1.750654	0.454324	2.089130	4.160019	-0.881302	1.081750	1.022928	0.005356	-0.541998	0.745036	-0.375165
283482	171627.0	-1.464380	1.368119	0.815992	-0.601282	-0.689115	-0.487154	-0.303778	0.884953	0.054065	-0.828015	-1.192581	0.944989	1.372532
283484	171627.0	-1.457978	1.378203	0.811515	-0.603760	-0.711883	-0.471672	-0.282535	0.880654	0.052808	-0.830603	-1.191774	0.942870	1.372621
284190	172233.0	-2.667936	3.160505	-3.355984	1.007845	-0.377397	-0.109730	-0.667233	2.309700	-1.639306	-1.449823	-0.508930	0.600035	-0.627313
284192	172233.0	-2.691642	3.123168	-3.339407	1.017018	-0.293095	-0.167054	-0.745886	2.325616	-1.634651	-1.440241	-0.511918	0.607878	-0.627645

1081 rows × 31 columns

slicing the duplicated number

In [10]:

```
crd[32:36]
```

Out[10]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14
32	26.0	-0.529912	0.873892	1.347247	0.145457	0.414209	0.100223	0.711206	0.176066	-0.286717	-0.484688	0.87249	0.851636	-0.571745	0.100974	-1.
33	26.0	-0.529912	0.873892	1.347247	0.145457	0.414209	0.100223	0.711206	0.176066	-0.286717	-0.484688	0.87249	0.851636	-0.571745	0.100974	-1.
34	26.0	-0.535388	0.865268	1.351076	0.147575	0.433680	0.086983	0.693039	0.179742	-0.285642	-0.482474	0.87180	0.853447	-0.571822	0.102252	-1.
35	26.0	-0.535388	0.865268	1.351076	0.147575	0.433680	0.086983	0.693039	0.179742	-0.285642	-0.482474	0.87180	0.853447	-0.571822	0.102252	-1.

Dropinng the duplicated number - keep first (mean keep the first number and drop the duplicate one)

In [11]:
```
crd.drop_duplicates(keep ='first' ,inplace = True)
```

Rechecking for dulicate number

In [12]:
```
crd[crd.duplicated()]
```

Out[12]:

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

In [13]:
```
crd.isnull().sum()
```

Out[13]:
```
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
```

Checking for the statistical value

In [14]:
```
crd.describe()
```

Out[14]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
count	283725.000000	283725.000000	283725.000000	283725.000000	283725.000000	283725.000000	283725.000000	283725.000000	283725.000000	283725.00
mean	94811.411324	0.005920	-0.004135	0.001607	-0.002969	0.001827	-0.001148	0.001800	-0.000857	-0.00
std	47480.798808	1.948029	1.646706	1.508681	1.414186	1.377011	1.331925	1.227666	1.179055	1.09
min	0.000000	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307	-26.160506	-43.557242	-73.216718	-13.43
25%	54207.000000	-0.915954	-0.600324	-0.889684	-0.850137	-0.689836	-0.769031	-0.552516	-0.208828	-0.64
50%	84693.000000	0.020386	0.063946	0.179958	-0.022256	-0.053469	-0.275168	0.040857	0.021897	-0.05
75%	139298.000000	1.316069	0.800283	1.026953	0.739625	0.612223	0.396785	0.570475	0.325691	0.59
max	172792.000000	2.454930	22.057729	9.382558	16.875344	34.801666	73.301626	120.589494	20.007208	15.59

checking the count of fraudulent 0 is not fraudulent

In [15]:
```
len(crd[crd['Class']==0])
```

Out[15]: 283252

checking the count of fraudulent 1 is fraudulent

In [16]:
```
len(crd[crd['Class']==1])
```

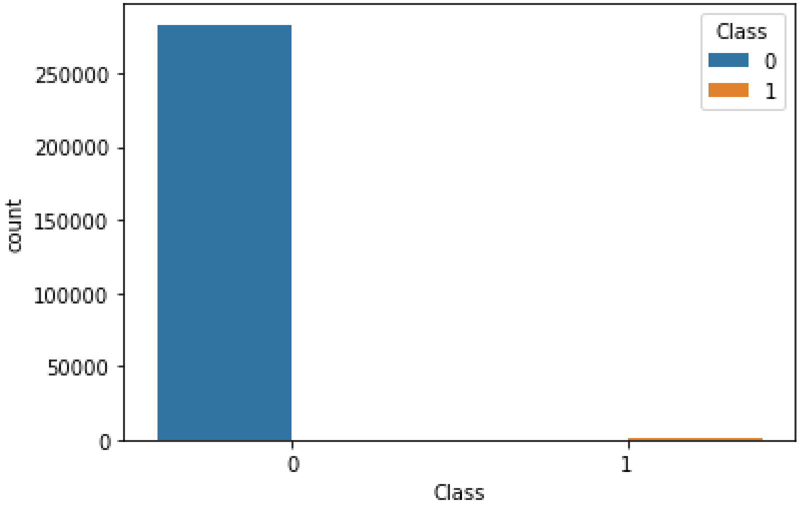
Out[16]: 473

In [17]:
```
len(crd)
```

Out[17]:
```
283725
```

In [18]:
```
sns.countplot(crd['Class'], hue =crd['Class']);
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



As we can see that the dataset is completely unbalance and if we do any modelling our model will be biasa & give an improper accuracy with these inbalances dataset

so we will use under samplling method

splitling our dataset into independent and dependent feature

In [19]:
```
x = crd.drop(['Class'], axis=1)
x.head
x.shape
```

```
y=crd['Class']
y.head
y.shape
```

Out[19]:
```
(283725,)
```

pip install u imbalanced-learn and import sampling

In [20]:
```
#from imblearn import under_sampling
#from imblearn.under_sampling import NearMiss
```

In [21]:
```
#pip install imbalanced-learn==0.6.0
#pip install scikit-learn==0.22.1
```

In [22]:
```
from imblearn import under_sampling
from imblearn.under_sampling import NearMiss
```

Implementing under sampling to handle mbalaced in the dataset

In [23]:
```
nm=NearMiss()
x_1, y_1,=nm.fit_sample(x,y)
x_1.shape
y_1.shape
```

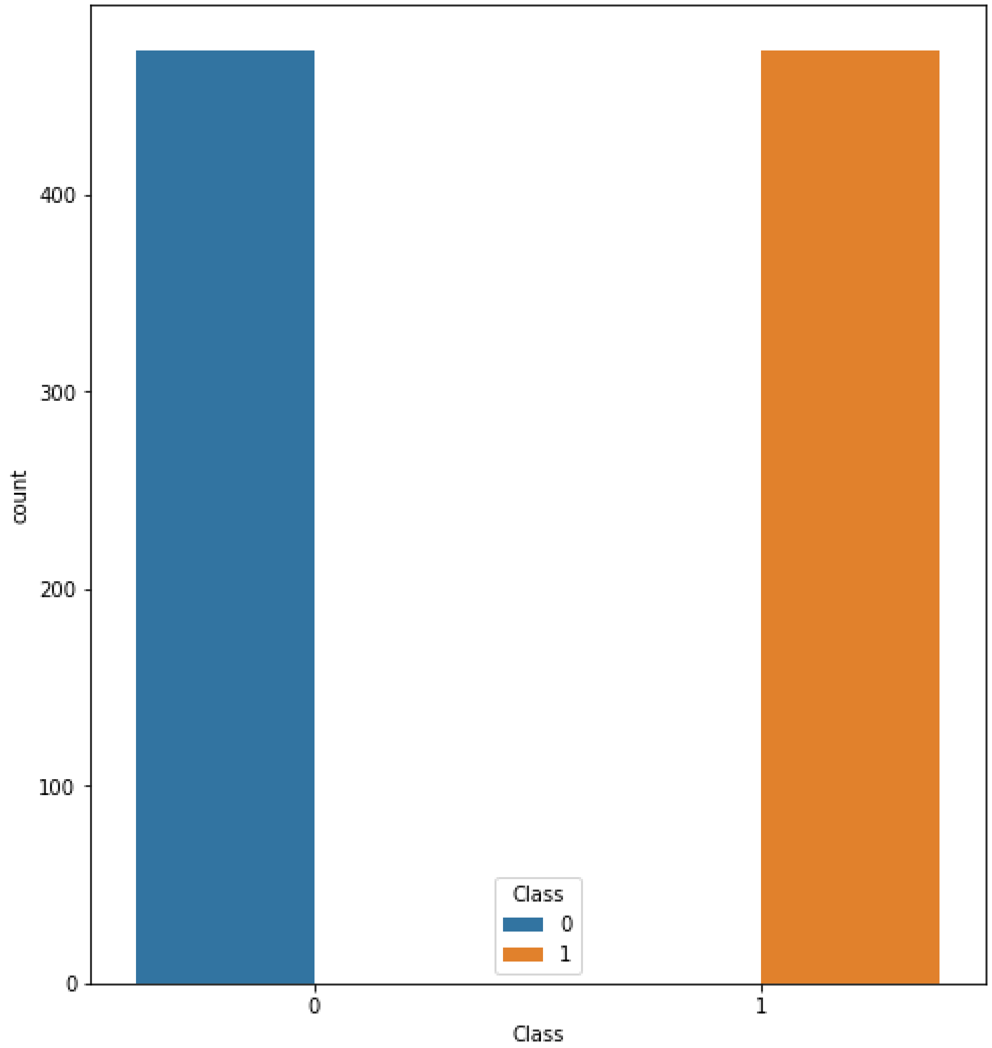
Out[23]:
```
(946,)
```

In [24]:
```
x_1.head()
y_1.head()
```

Out[24]:
```
0    0
1    0
2    0
3    0
4    0
Name: Class, dtype: int64
```

In [25]:
```
plt.figure(figsize=(8,9))
sns.countplot(y_1, hue= y_1);
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg:
x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword w
ill result in an error or misinterpretation.
  warnings.warn(
```



the dataset is now balanced so we can use any classification algorithm we will be using logistic regression

In [37]:
```
from sklearn.model_selection import train_test_split

x_train,x_test, y_train, y_test = train_test_split(x_1, y_1, test_size=0.20, random_state=40)



print('x_train : ',x_train.shape)
print('x_test : ' ,x_test.shape)
print('y_train : ',y_train.shape)
print('y_test : ' ,x_test.shape)
```

```
x_train :  (756, 30)
x_test :  (190, 30)
y_train :  (756,)
y_test :  (190, 30)
```

In [38]:
```
#from sklearn.linear_model import LogisticRegression

from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver='lbfgs', max_iter=1000)
model.fit(x_train, y_train)

#log_model = LogisticRegression(solver='lbfgs', max_iter=1000)
#model.fit(x_train, y_train)
```

Out[38]:
```
▾        LogisticRegression
LogisticRegression(max_iter=1000)
```

In [39]:
```
y_predicated = model.predict(x_test)
y_predicated
```

Out[39]:
```
array([0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0,
       0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
```

```
       0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1], dtype=int64)
```

In [40]:
```python
model.score(x_test, y_test)
```

Out[40]:
```
0.9473684210526315
```

In [41]:
```python
model.coef_
```

Out[41]:
```
array([[-3.59435915e-05,  1.78586091e-01, -2.11482339e-01,
        -6.23490989e-01,  7.55530572e-01,  1.65272884e-01,
        -3.93463724e-02, -3.63124201e-01, -7.69274121e-02,
        -4.27134112e-01, -5.16555330e-01,  9.41193555e-02,
        -4.72119610e-01, -2.07755177e-01, -9.23700851e-01,
        -1.15421446e-01, -2.65809703e-01, -5.03723019e-01,
        -6.29623885e-02,  4.43054454e-02, -1.55357586e-01,
         2.37400427e-01,  6.52428730e-02, -1.49852934e-01,
        -8.77113887e-02, -8.21672810e-02,  4.38439515e-02,
        -7.09979384e-03, -3.42961148e-04,  3.60888393e-02]])
```

In [42]:
```python
model.intercept_
```

Out[42]:
```
array([-0.21601374])
```

In [54]:
```python
from sklearn.metrics import confusion_matrix, accuracy_score ,classification_report,f1_score

cm = confusion_matrix(y_test,y_predicated)
print (cm)
print('accuracy_score :', accuracy_score(y_test, y_predicated))
print (classification_report(y_test,y_predicated))
```

```
[[93  3]
 [ 6 88]]
accuracy_score : 0.9526315789473684
              precision    recall  f1-score   support

           0       0.94      0.97      0.95        96
           1       0.97      0.94      0.95        94
```

```
    accuracy                           0.95       190
   macro avg       0.95      0.95      0.95       190
weighted avg       0.95      0.95      0.95       190
```

In [ ]:

Training a Decision Tree Model¶

Let's start by Training a single decision tree first!

Import DecisionTreeClassifier

In [61]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [62]:

```
dtree=DecisionTreeClassifier()
```

In [63]:

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

Out[63]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [64]:

```
from sklearn.metrics import classification_report,confusion_matrix
```

In [69]:

```
print (classification_report(y_test,y_predicated))
```

```
              precision    recall  f1-score   support

           0       0.94      0.97      0.95        96
           1       0.97      0.94      0.95        94

    accuracy                           0.95       190
   macro avg       0.95      0.95      0.95       190
```

```
weighted avg       0.95      0.95      0.95       190
```

In [ ]:

Using RandomForest

In [51]:

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
model.fit(x_train, y_train)

y_predicated = model.predict(x_test)
print(classification_report(y_test,y_predicated))
```

```
              precision    recall  f1-score   support

           0       0.94      0.97      0.95        96
           1       0.97      0.94      0.95        94

    accuracy                           0.95       190
   macro avg       0.95      0.95      0.95       190
weighted avg       0.95      0.95      0.95       190
```

In [60]:

```
print(confusion_matrix(y_test,y_predicated))
```

```
[[93  3]
 [ 6 88]]
```

In [ ]:

In [ ]:

In [ ]: