In [1]:
```
import pandas as pd
import numpy as np
```

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

In [4]:
```
data.head()
```

Out[4]:

V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

In [5]:
```
data.describe()
```

Out[5]:

V21	V22	V23	V24	V25	V26	V27	V28	Amount	Cla
2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	284807.000000	284807.0000
1.537294e-16	7.959909e-16	5.367590e-16	4.458112e-15	1.453003e-15	1.699104e-15	-3.660161e-16	-1.206049e-16	88.349619	0.0017
7.345240e-01	7.257016e-01	6.244603e-01	6.056471e-01	5.212781e-01	4.822270e-01	4.036325e-01	3.300833e-01	250.120109	0.0415
3.483038e+01	-1.093314e+01	-4.480774e+01	-2.836627e+00	-1.029540e+01	-2.604551e+00	-2.256568e+01	-1.543008e+01	0.000000	0.0000
2.283949e-01	-5.423504e-01	-1.618463e-01	-3.545861e-01	-3.171451e-01	-3.269839e-01	-7.083953e-02	-5.295979e-02	5.600000	0.0000
2.945017e-02	6.781943e-03	-1.119293e-02	4.097606e-02	1.659350e-02	-5.213911e-02	1.342146e-03	1.124383e-02	22.000000	0.0000
1.863772e-01	5.285536e-01	1.476421e-01	4.395266e-01	3.507156e-01	2.409522e-01	9.104512e-02	7.827995e-02	77.165000	0.0000
2.720284e+01	1.050309e+01	2.252841e+01	4.584549e+00	7.519589e+00	3.517346e+00	3.161220e+01	3.384781e+01	25691.160000	1.0000

In [6]:
```
data.isna().sum()
```

Out[6]:
```
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
```

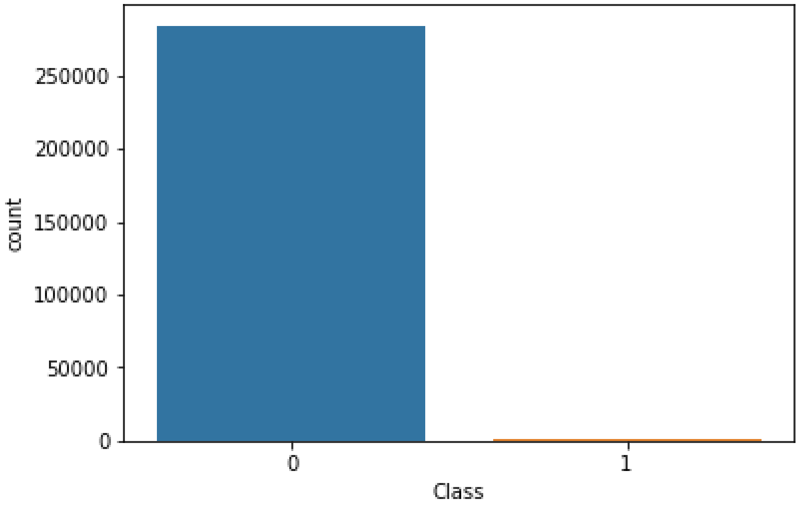
In [7]:
```
data.dtypes
```

Out[7]:
```
Time      float64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

In [9]:

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot('Class',data = data)
plt.show()
```



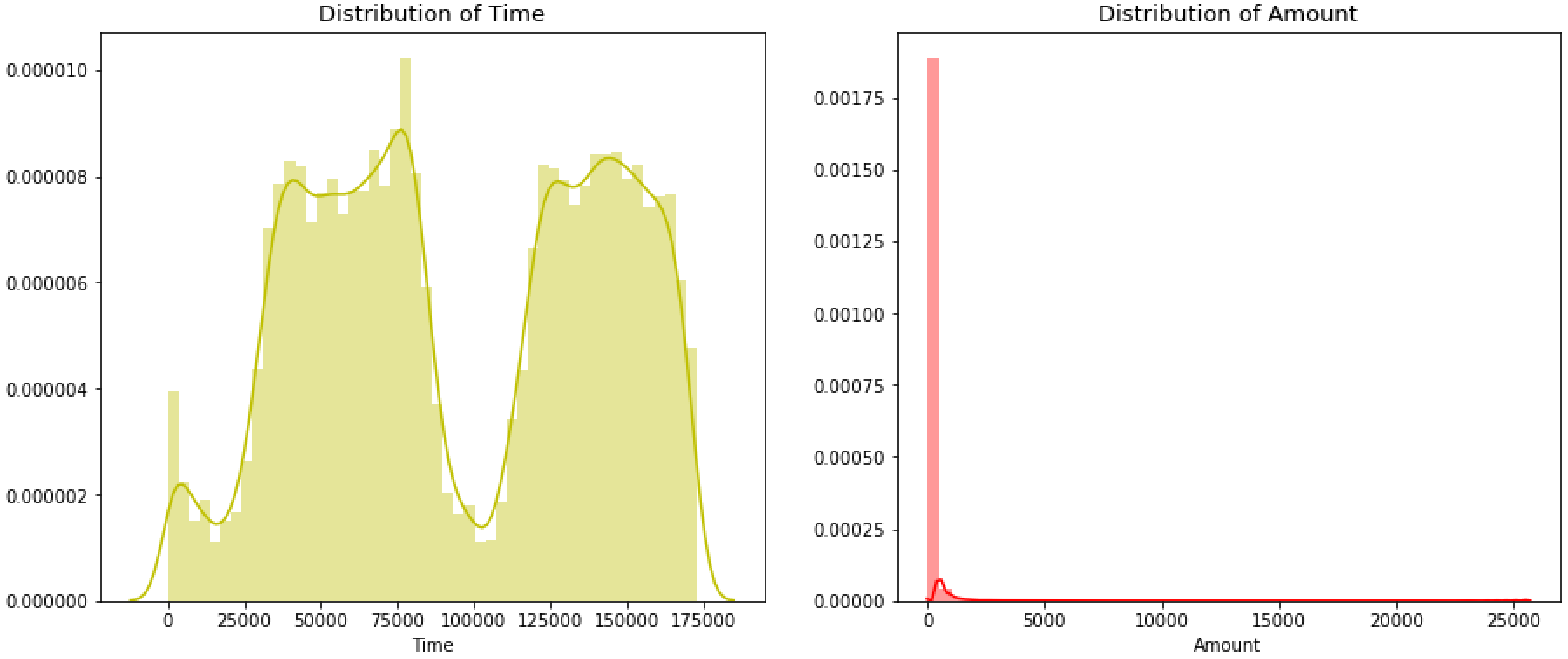
In [11]:

```
print(f"The percentage of data in class 0 : {100*data['Class'].value_counts()[0]/data.shape[0]}")
print(f"The percentage of data in class 1 : {100*data['Class'].value_counts()[1]/data.shape[0]}")
```

```
The percentage of data in class 0 : 99.827251436938
The percentage of data in class 1 : 0.1727485630620034
```

In [13]:
```
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))
ax1 = sns.distplot(data['Time'], ax=ax1, color='y')
ax2 = sns.distplot(data['Amount'], ax=ax2, color='r')
ax1.set_title('Distribution of Time', fontsize=13)
ax2.set_title('Distribution of Amount', fontsize=13)
```

Out[13]: Text(0.5,1,'Distribution of Amount')



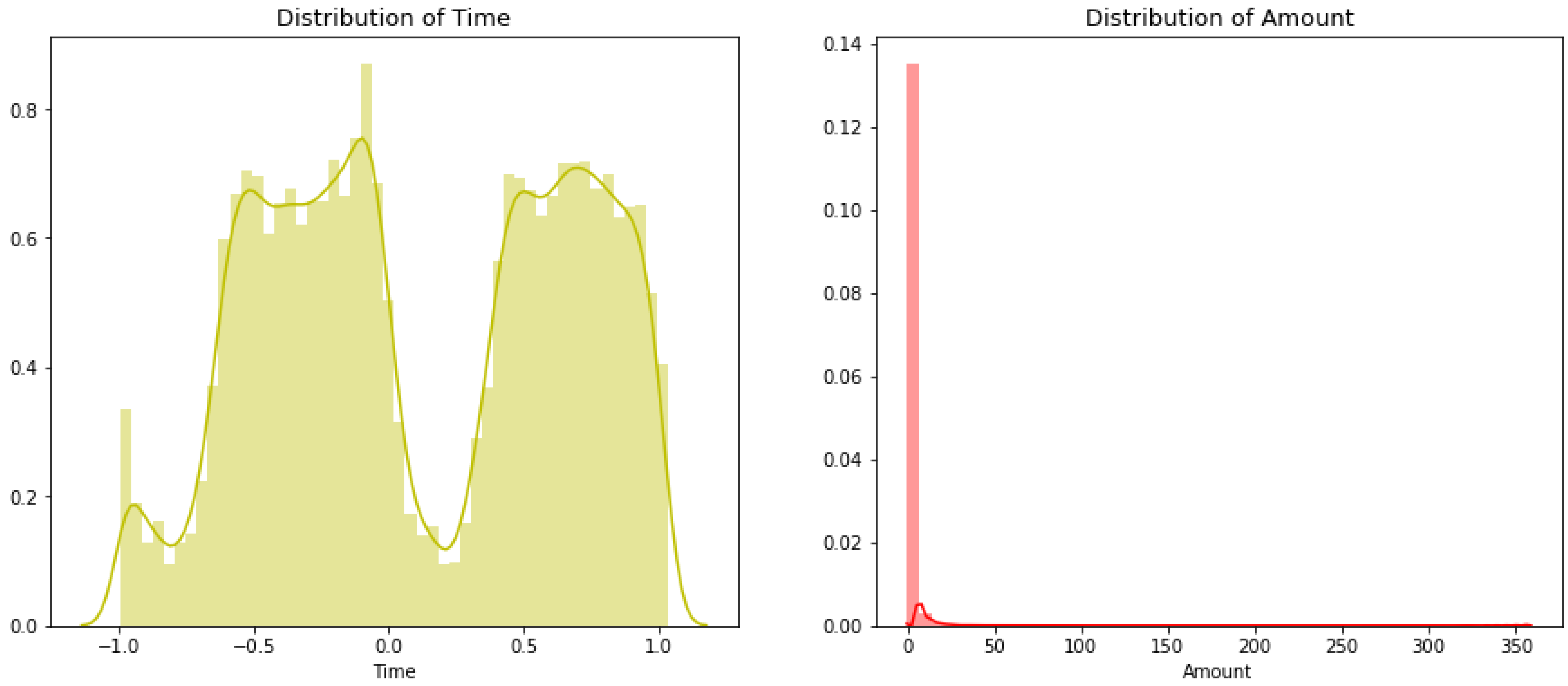
In [14]:
```
from sklearn.preprocessing import RobustScaler
```

In [15]:
```
scaler = RobustScaler()
```

In [18]:
```
data['Amount'] = scaler.fit_transform(data['Amount'].values.reshape(-1,1))
data['Time'] = scaler.fit_transform(data['Time'].values.reshape(-1,1))
```

In [19]:
```
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))
ax1 = sns.distplot(data['Time'], ax=ax1, color='y')
ax2 = sns.distplot(data['Amount'], ax=ax2, color='r')
ax1.set_title('Distribution of Time', fontsize=13)
ax2.set_title('Distribution of Amount', fontsize=13)
```

Out[19]:
```
Text(0.5,1,'Distribution of Amount')
```



In [20]:
```
from sklearn.model_selection import train_test_split
```

In [21]:
```
X_train,X_test,y_train,y_test= train_test_split(data.drop('Class',1),data['Class'],stratify=data['Class'],random_sta
```

In [32]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
```

In [33]:
```
rf = LogisticRegression(n_jobs = -1,verbose  = 2)
```

In [34]:
```
rf.fit(X_train,y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   13.6s finished
```

Out[34]:
```
LogisticRegression(n_jobs=-1, verbose=2)
```

In [35]:
```
predictions = rf.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import classification_report

print(classification_report(predictions,y_test))
```

Dealing with Class Imbalance

Random OverSampling

Randomly oversample the minority class to even out the distribution

Pros:

1. Easier than smote

Cons:

1. Random oversampling repeatation of the same data(redundant learning)

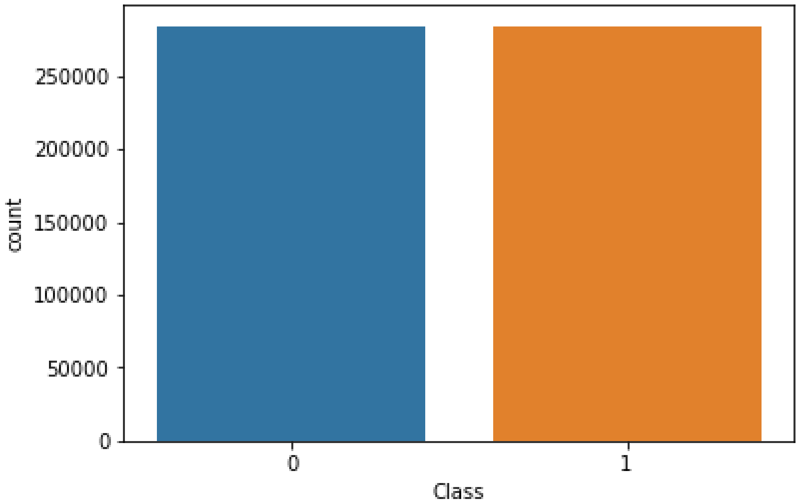
In [37]:
```
from imblearn.over_sampling import RandomOverSampler
```

In [38]:
```
ros = RandomOverSampler(random_state=42,sampling_strategy = 'auto')
```

In [39]:
```
X,y = data.drop('Class',1),data['Class']

ros_x,ros_y = ros.fit_resample(X,y)
```

In [40]:
```
sns.countplot(ros_y)
plt.show()
```



Random Undersampling

Removing over-represented/majority class data points till the distribution is fixed

Pros:

1. Easier than smote
2. No redundant learning

Cons:

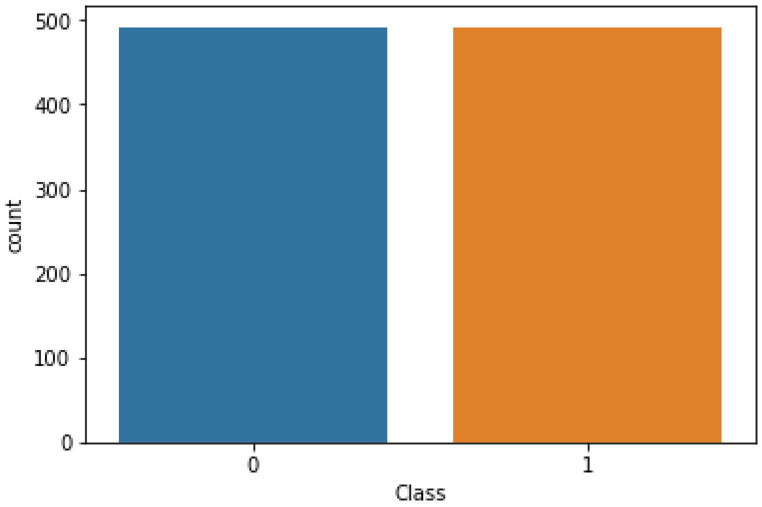
1. loss of information(Should not be used alone for fixing class imbalance)

In [49]:
```
from imblearn.under_sampling import RandomUnderSampler
```

In [50]:
```
rus = RandomUnderSampler(random_state=42,sampling_strategy = 'auto')
```

In [51]:
```
rus_x,rus_y = ros.fit_resample(X,y)
```

In [52]:
```
sns.countplot(ros_y)
plt.show()
```



Smote

Synthetic Minority over sampling technique

Pros:

1. No redundant learning
2. Better assumptions

Cons:

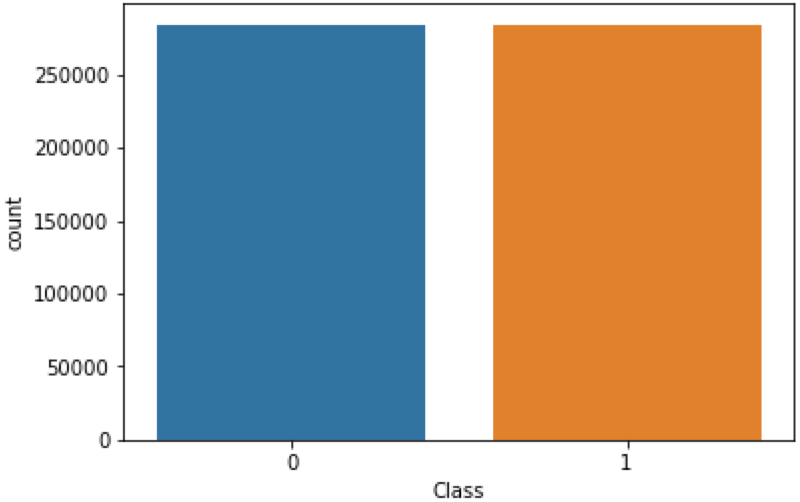
1. Synthetic data and hence not proper representation
2. Very sensitive to outliers
3. Fails for datasets with higher dimensions

In [48]:
```
from imblearn.over_sampling import SMOTE
```

In [53]:
```
sm = SMOTE()

ros_x,ros_y = sm.fit_resample(X,y)
```

In [54]:
```
sns.countplot(ros_y)
plt.show()
```



In [ ]:
```

```