In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from IPython import get_ipython
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
data = pd.read_csv("creditcard.csv")
```

In [3]:

```
data.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

In [4]:

```
data.tail()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006

5 rows × 31 columns

In [5]:

```
data.shape
```

Out[5]:

```
(284807, 31)
```

In [6]:

```
data.columns
```

Out[6]:

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V1
0',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V2
0',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

In [7]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   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  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [8]:

```
data.describe()
```

Out[8]:

	Time	V1	V2	V3	V4	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+
mean	94813.859575	1.759061e-12	-8.251130e-13	-9.654937e-13	8.321385e-13	1.649999e-
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+

8 rows × 31 columns

In [9]:

```
data.duplicated().sum()
```

Out[9]:

```
1081
```

In [10]:

```
data = data.drop_duplicates()
```

In [11]:

```
data.isnull().sum()
```

Out[11]:

```
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 [12]:

```
data['Class'].unique()
```

Out[12]:

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

In [13]:

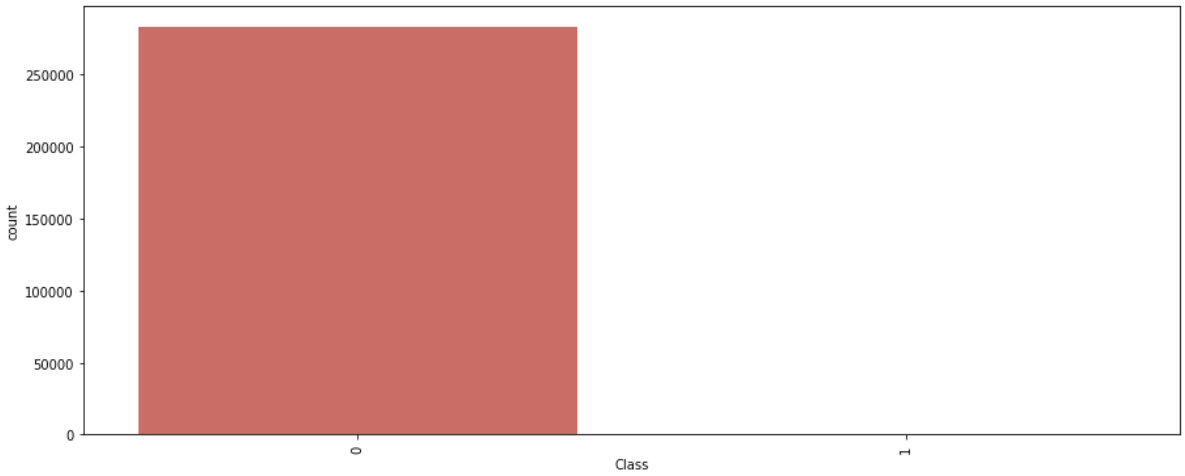
```
data['Class'].value_counts()
```

Out[13]:

```
0    283253
1       473
Name: Class, dtype: int64
```

In [14]:

```
plt.figure(figsize=(15,6))
sns.countplot(data['Class'], data = data,
                  palette='hls')
plt.xticks(rotation = 90)
plt.show()
```

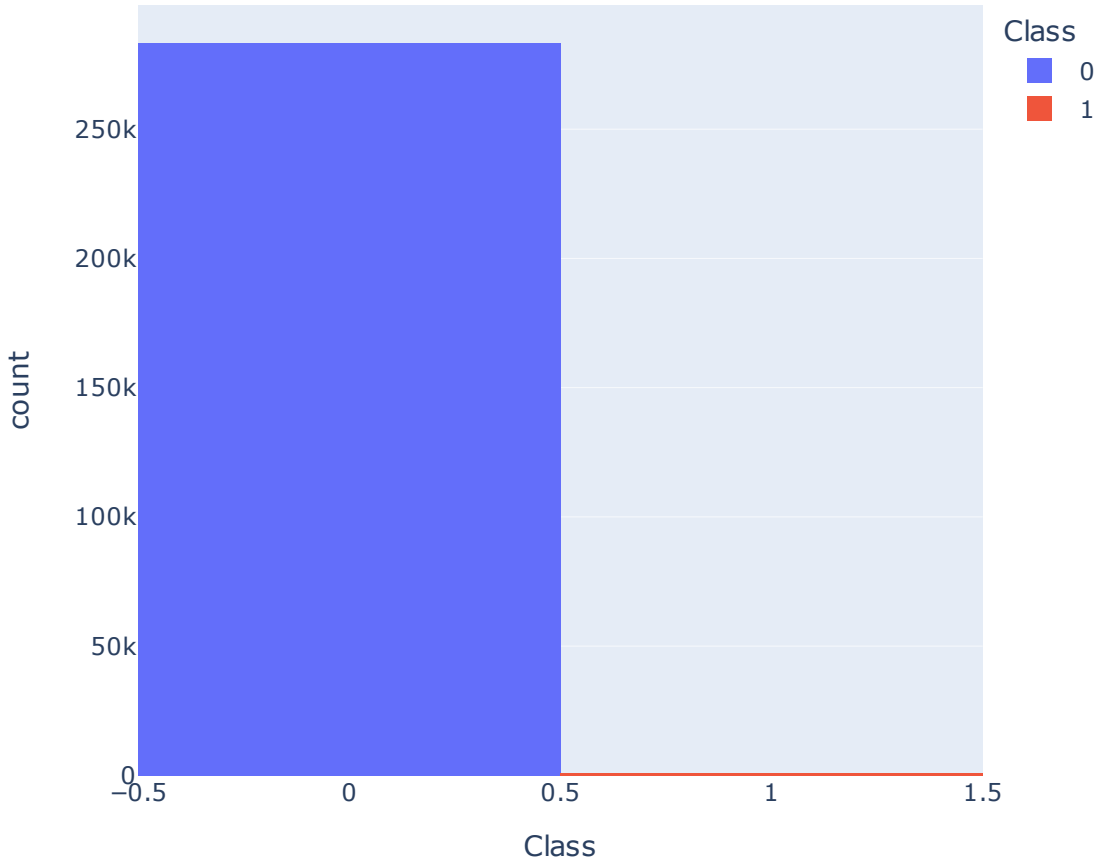


In [15]:

```
import plotly.express as px
```

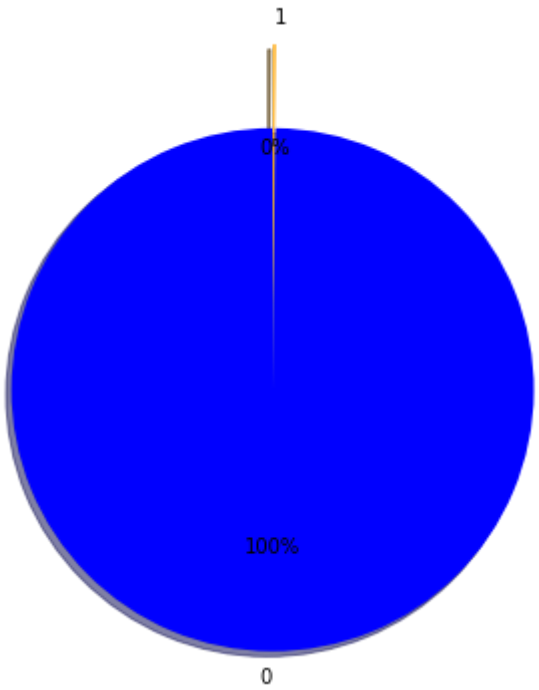
In [16]:

```
fig1 = px.histogram(data, x = 'Class', color = 'Class')
fig1.show()
```



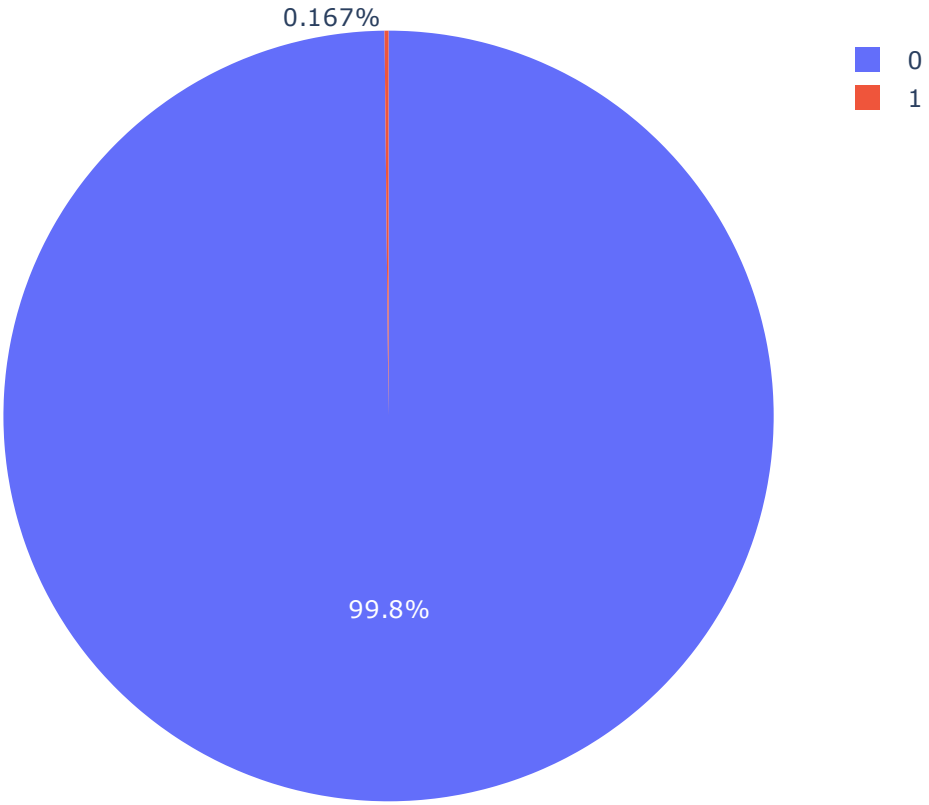
In [18]:

```
plt.figure(figsize=(15,6))
colors = sns.color_palette('bright')
explode = [0.3, 0.02]
plt.pie(data['Class'].value_counts(), colors = ['blue', 'orange'],
        labels = [0,1],
        explode = explode, autopct = '%0.0f%%', shadow = 'True',
        startangle = 90)
plt.show()
```



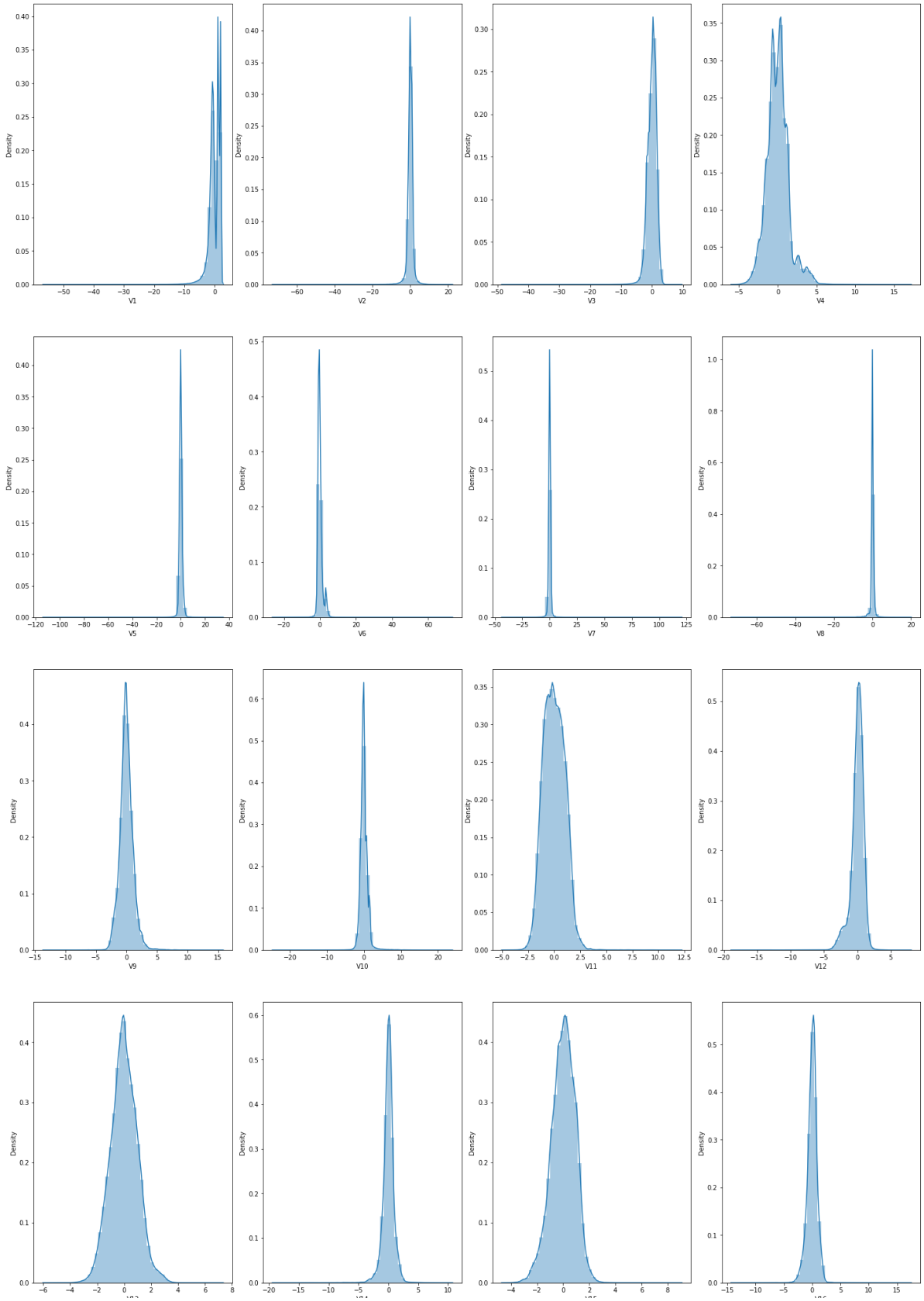
In [19]:

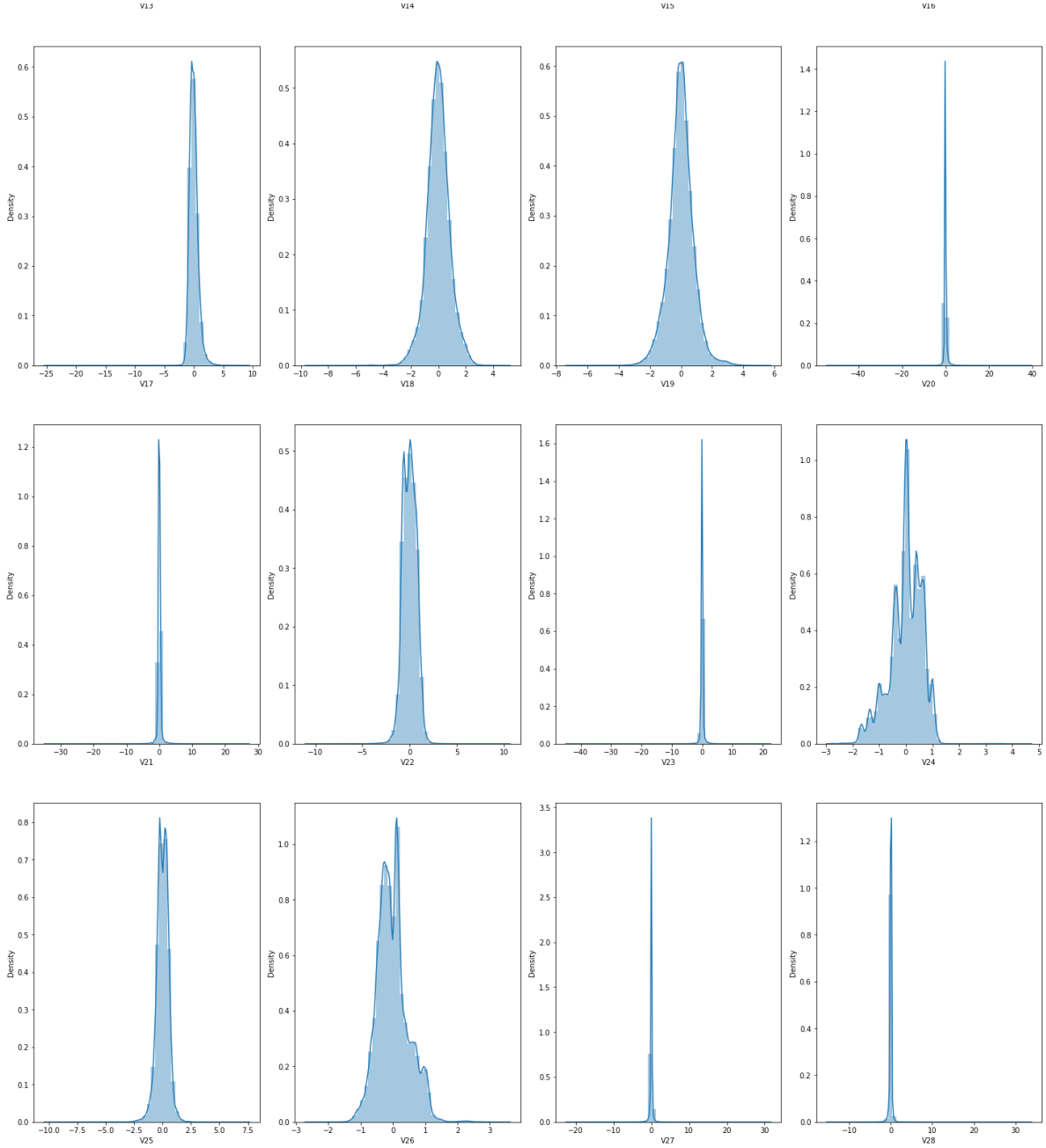
```
fig2 = px.pie(values=data['Class'].value_counts(), names= [0,1])
fig2.show()
```



In [17]:

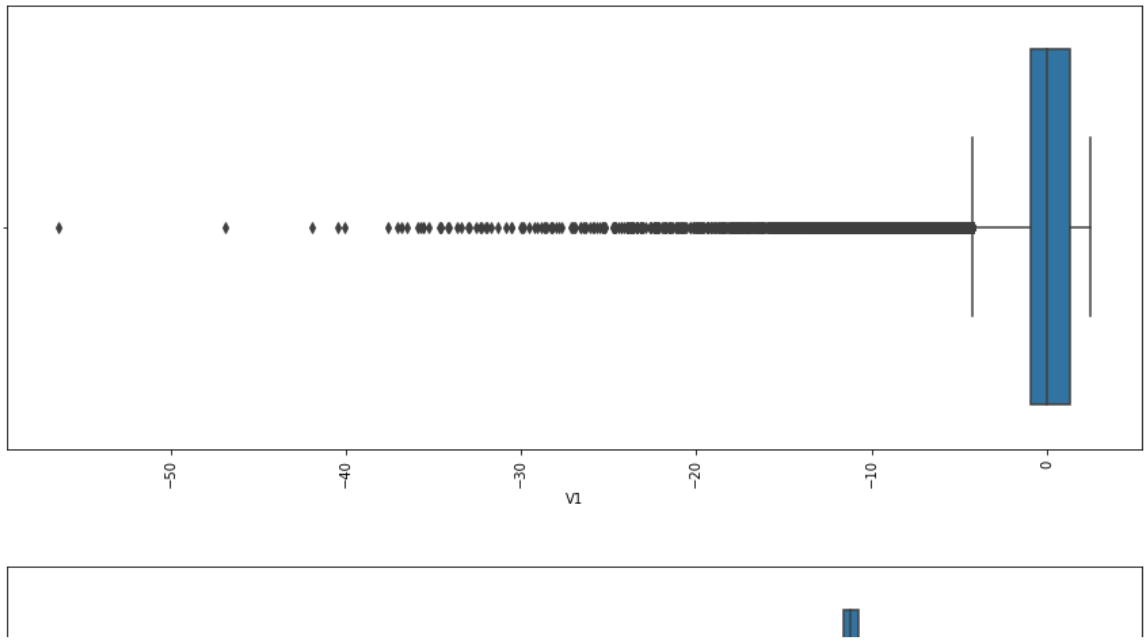
```
data1 = data.drop(columns=['Time', 'Amount', 'Class'], axis=1)
fig, ax = plt.subplots(ncols=4, nrows=7, figsize=(20, 50))
index = 0
ax = ax.flatten()
for col in data1.columns:
    sns.distplot(data1[col], ax=ax[index])
    index += 1
plt.tight_layout(pad=0.5, w_pad=0.5, h_pad=5)
```




V13
V14
V15
V16
Density
V17
V18
V19
V20
V21
V22
V23
V24
V25
V26
V27
V28

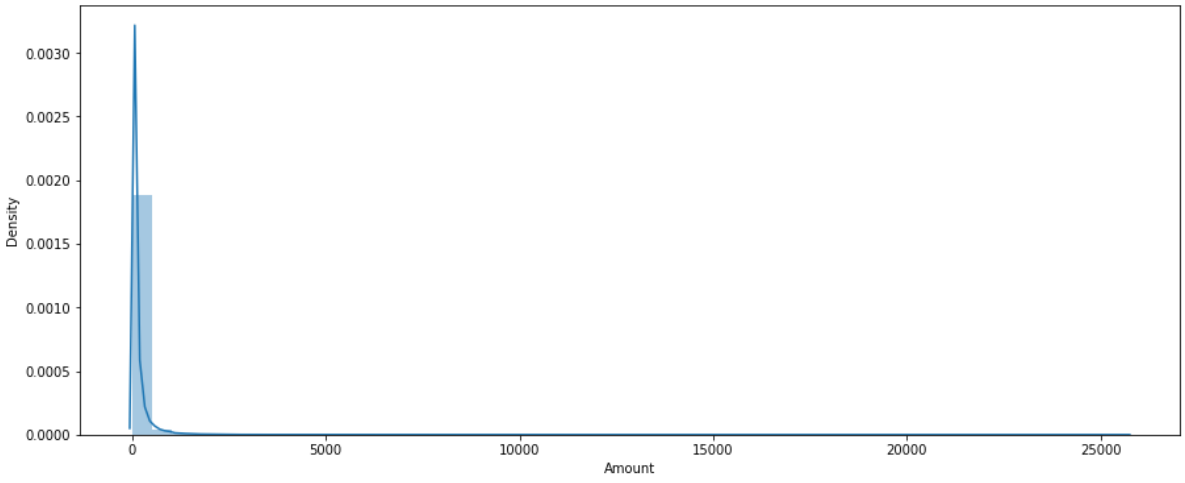
In [20]:

```
for i in data1.columns:
    plt.figure(figsize=(15,6))
    sns.boxplot(data1[i])
    plt.xticks(rotation = 90)
    plt.show()
```



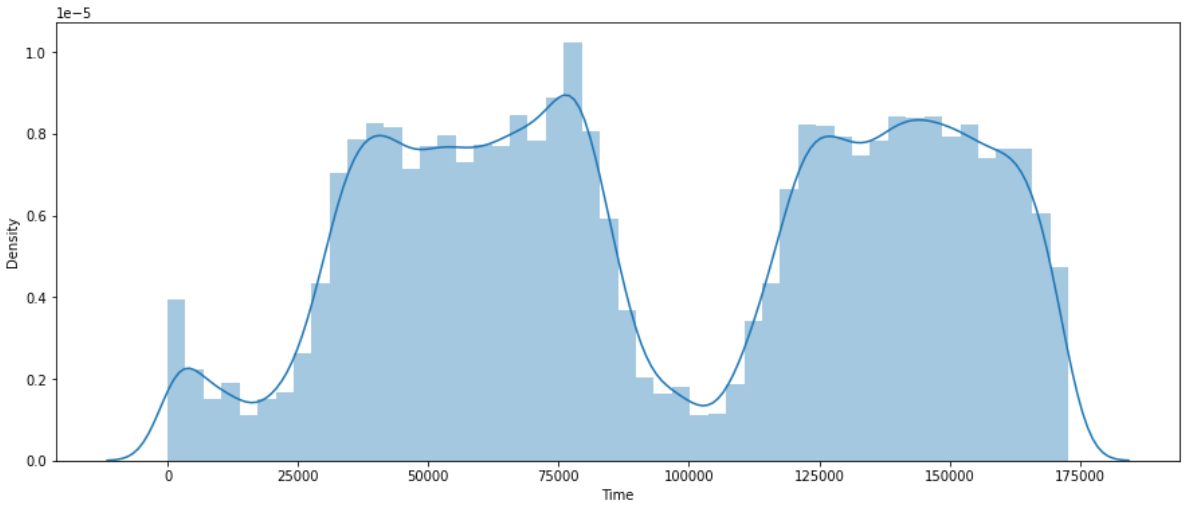
In [21]:

```
plt.figure(figsize=(15,6))
sns.distplot(data['Amount'])
plt.show()
```



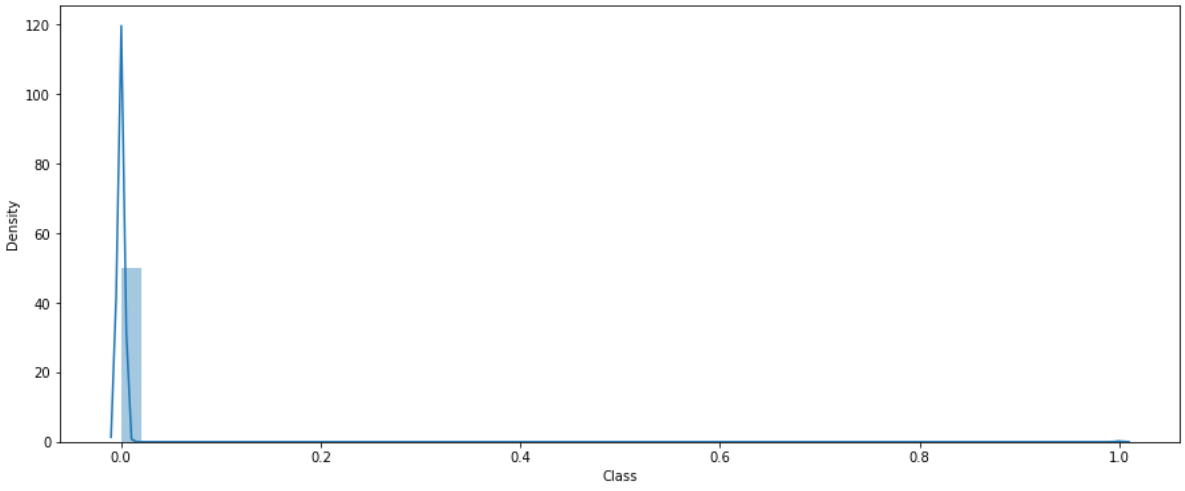
In [22]:

```
plt.figure(figsize=(15,6))
sns.distplot(data['Time'])
plt.show()
```



In [29]:

```
plt.figure(figsize=(15,6))
sns.distplot(data['Class'])
plt.show()
```



In [23]:

```
x = data.drop(columns=['Class'], axis=1)
y = data['Class']
```

In [24]:

```
from sklearn.preprocessing import StandardScaler
```

In [25]:

```
sc = StandardScaler()
x_scaler = sc.fit_transform(x)
```

In [26]:

```
x_scaler [-1]
```

Out[26]:

```
array([ 1.64236181, -0.27686005, -0.1127094 ,  0.46512487, -0.35589839,
       -0.01043804, -0.48687097,  1.28309413, -0.35095639,  0.44525779,
       -0.84910935, -1.02153857, -0.03096322, -0.18956298, -0.08881285,
        0.04403866, -0.34769819, -0.78402639,  0.19813651, -0.31455594,
        0.49710317,  0.36113414,  0.88757737,  0.60378078,  0.01417241,
       -0.90828607, -1.69777619, -0.01055821,  0.03994074,  0.51329005])
```

In [27]:

```
from sklearn.model_selection import train_test_split
```

In [28]:

```
x_train, x_test, y_train, y_test = train_test_split(x_scaler, y,
                                                    test_size=0.20,
                                                    random_state=42,
                                                    stratify=y)
```

In [30]:

```
from sklearn.linear_model import LogisticRegression
```

In [31]:

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

Out[31]:

```
LogisticRegression()
```

In [32]:

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

In [33]:

```
print("Training Accuracy :", model.score(x_train, y_train))
print("Testing Accuracy :", model.score(x_test, y_test))
```

```
Training Accuracy : 0.999242223984492
Testing Accuracy : 0.9991188806259472
```

In [34]:

```
from sklearn.metrics import classification_report, f1_score
```

In [35]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56651
           1       0.85      0.58      0.69        95

    accuracy                           1.00     56746
   macro avg       0.92      0.79      0.84     56746
weighted avg       1.00      1.00      1.00     56746
```

In [36]:

```
print("F1 Score:",f1_score(y_test, y_pred))
```

```
F1 Score: 0.6875
```

In [37]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [38]:

```
model1 = RandomForestClassifier()
model1.fit(x_train, y_train)
```

Out[38]:

```
RandomForestClassifier()
```

In [39]:

```
y_pred = model1.predict(x_test)
```

In [40]:

```
print("Training Accuracy :", model1.score(x_train, y_train))
print("Testing Accuracy :", model1.score(x_test, y_test))
```

```
Training Accuracy : 1.0
Testing Accuracy : 0.9995241955380115
```

In [41]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56651
           1       0.97      0.74      0.84        95

    accuracy                           1.00     56746
   macro avg       0.99      0.87      0.92     56746
weighted avg       1.00      1.00      1.00     56746
```

In [42]:

```
print("F1 Score:",f1_score(y_test, y_pred))
```

```
F1 Score: 0.8383233532934131
```

In [43]:

```
from xgboost import XGBClassifier
```

In [44]:

```
model2 = XGBClassifier(n_jobs=-1)
model2.fit(x_train, y_train)
```

Out[44]:

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwis
e',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weigh
t=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=-1, num_parallel_tree=1, predictor='auto', random_sta
te=0,
              reg_alpha=0, reg_lambda=1, ...)
```

In [45]:

```
y_pred = model2.predict(x_test)
```

In [46]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56651
           1       0.97      0.75      0.85        95

    accuracy                           1.00     56746
   macro avg       0.99      0.87      0.92     56746
weighted avg       1.00      1.00      1.00     56746
```

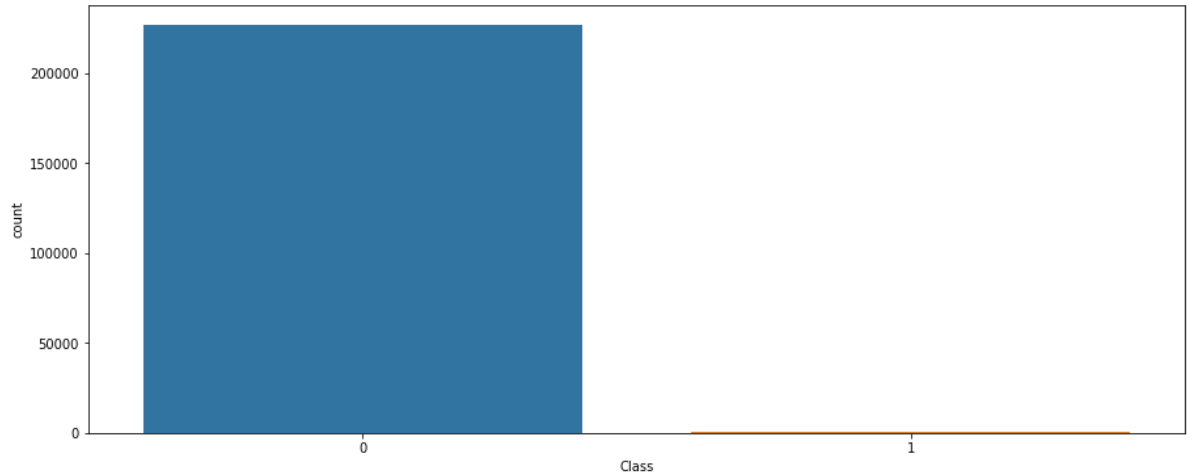
In [47]:

```
print("F1 Score:",f1_score(y_test, y_pred))
```

```
F1 Score: 0.8452380952380952
```

In [48]:

```
plt.figure(figsize=(15,6))
sns.countplot(y_train)
plt.show()
```

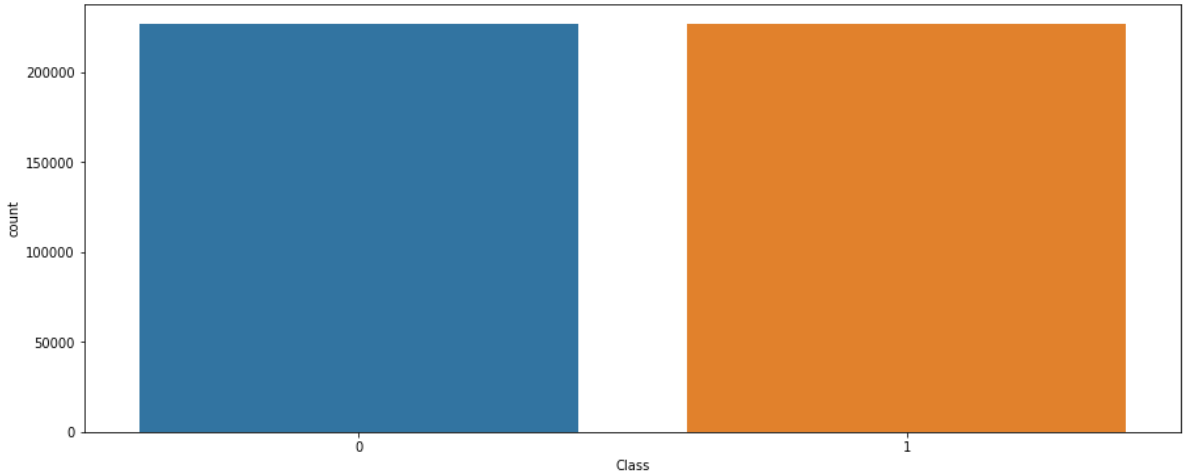


In [49]:

```
from imblearn.over_sampling import SMOTE
over_sample = SMOTE()
x_smote, y_smote = over_sample.fit_resample(x_train, y_train)
```

In [50]:

```
plt.figure(figsize=(15,6))
sns.countplot(y_smote)
plt.show()
```



In [51]:

```
from xgboost import XGBClassifier
```

In [52]:

```
model3 = XGBClassifier(n_jobs=-1)
model3.fit(x_smote, y_smote)
```

Out[52]:

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwis
e',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weigh
t=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=-1, num_parallel_tree=1, predictor='auto', random_sta
te=0,
              reg_alpha=0, reg_lambda=1, ...)
```

In [53]:

```
y_pred = model3.predict(x_test)
```

In [54]:

```
print("Training Accuracy :", model3.score(x_train, y_train))
print("Testing Accuracy :", model3.score(x_test, y_test))
```

```
Training Accuracy : 1.0
Testing Accuracy : 0.9992246149508336
```

In [55]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56651
           1       0.76      0.79      0.77        95

    accuracy                           1.00     56746
   macro avg       0.88      0.89      0.89     56746
weighted avg       1.00      1.00      1.00     56746
```

In [56]:

```
print("F1 Score:",f1_score(y_test, y_pred))
```

```
F1 Score: 0.7731958762886598
```