Credit Card fraud detection

The dataset has credit card transactions, and its features are the result of PCA analysis. It has ‘Amount’, ‘Time’, and ‘Class’ features where ‘Amount’ shows the monetary value of every transaction, ‘Time’ shows the seconds elapsed between the first and the respective transaction, and ‘Class’ shows whether a transaction is legit or not.

In ‘Class’, value 1 represents a fraud transaction, and value 0 represents a valid transaction.

Credit card fraud detection is a critical application of machine learning in finance. It involves a series of steps starting from data collection and preprocessing to model building, evaluation, deployment, and continuous monitoring to safeguard against fraudulent activities.

Overview of the Dataset:

Dataset Source: The dataset consists of credit card transactions made in September 2013 by European cardholders. It contains numerical input variables that are the result of PCA transformations due to privacy concerns. The features include time, amount, and anonymized numerical features (V1-V28) that are a result of PCA transformation.

Objective: The primary objective is to detect fraudulent transactions among a vast number of legitimate ones.

Imbalanced Data: Typically, the dataset is highly imbalanced, where fraudulent transactions are a tiny fraction of the total transactions, making it challenging to train models effectively.

In [ ]:

```

```

In [ ]:

```
Data Preprocessing: Explore and understand the data. Handle missing values, outliers, and scale/normalize numer

Feature Engineering: Create or extract new features that might aid in fraud detection.

Handling Imbalanced Data: Techniques like oversampling (SMOTE), undersampling, or using algorithms robust to cl

Model Selection: Commonly used models include Logistic Regression, Decision Trees, Random Forests, Gradient Boo

Model Evaluation: Metrics like precision, recall, F1-score, and area under the ROC curve (AUC-ROC) are used to

Hyperparameter Tuning: Optimize model parameters to enhance performance.

Deployment and Monitoring: Deploy the model in a production environment and continually monitor its performance
```

Import Models

In [1]:

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [2]:

```python
df = pd.read_csv('creditcard.csv')
df.head()
```

Out[2]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458

5 rows × 31 columns

In [3]:

```python
# statistical info
df.describe()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15	2.040130e-15	-1.698953e-15	-1.893285e-16
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01

8 rows × 31 columns

In [4]:

```
# datatype info
df.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
```

Preprocessing the Data

In [5]:

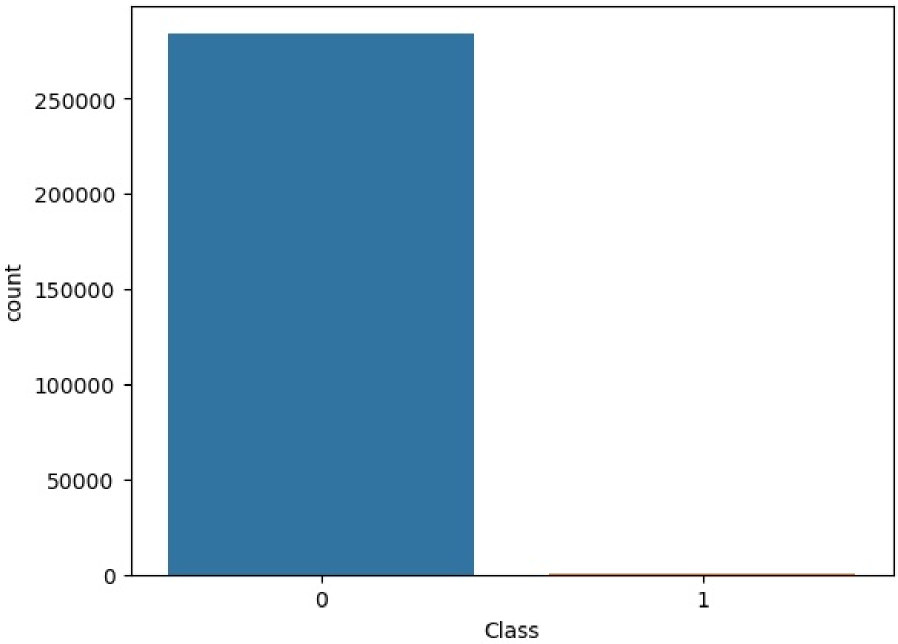
```
# check for null values
df.isnull().sum()
```

Out[5]:
```
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
```

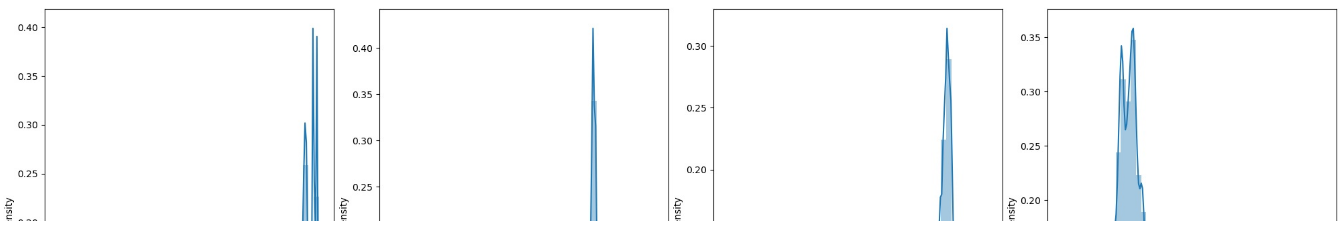
Exploratory Data Analysis

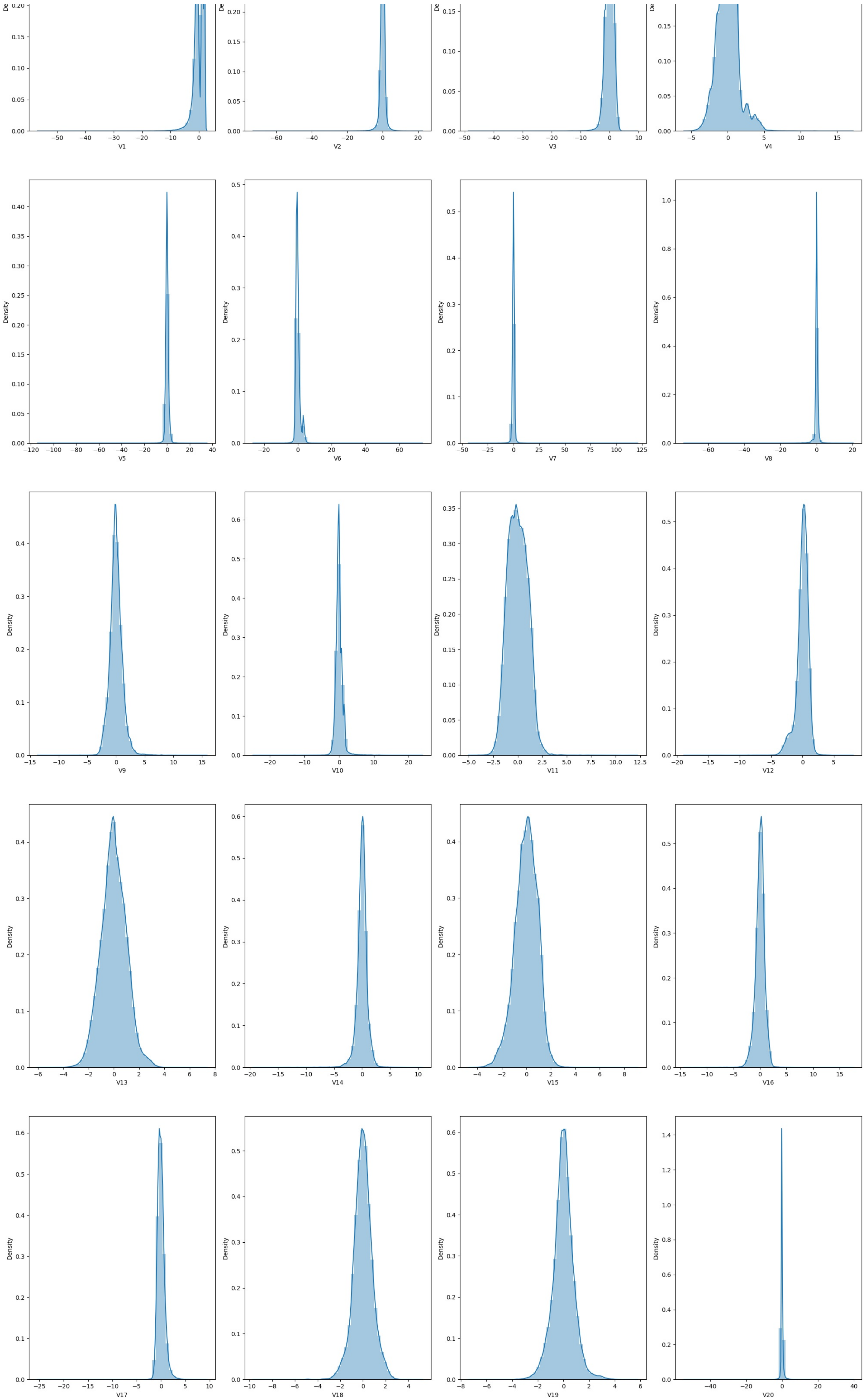
In [6]:
```
sns.countplot(df['Class'])
```

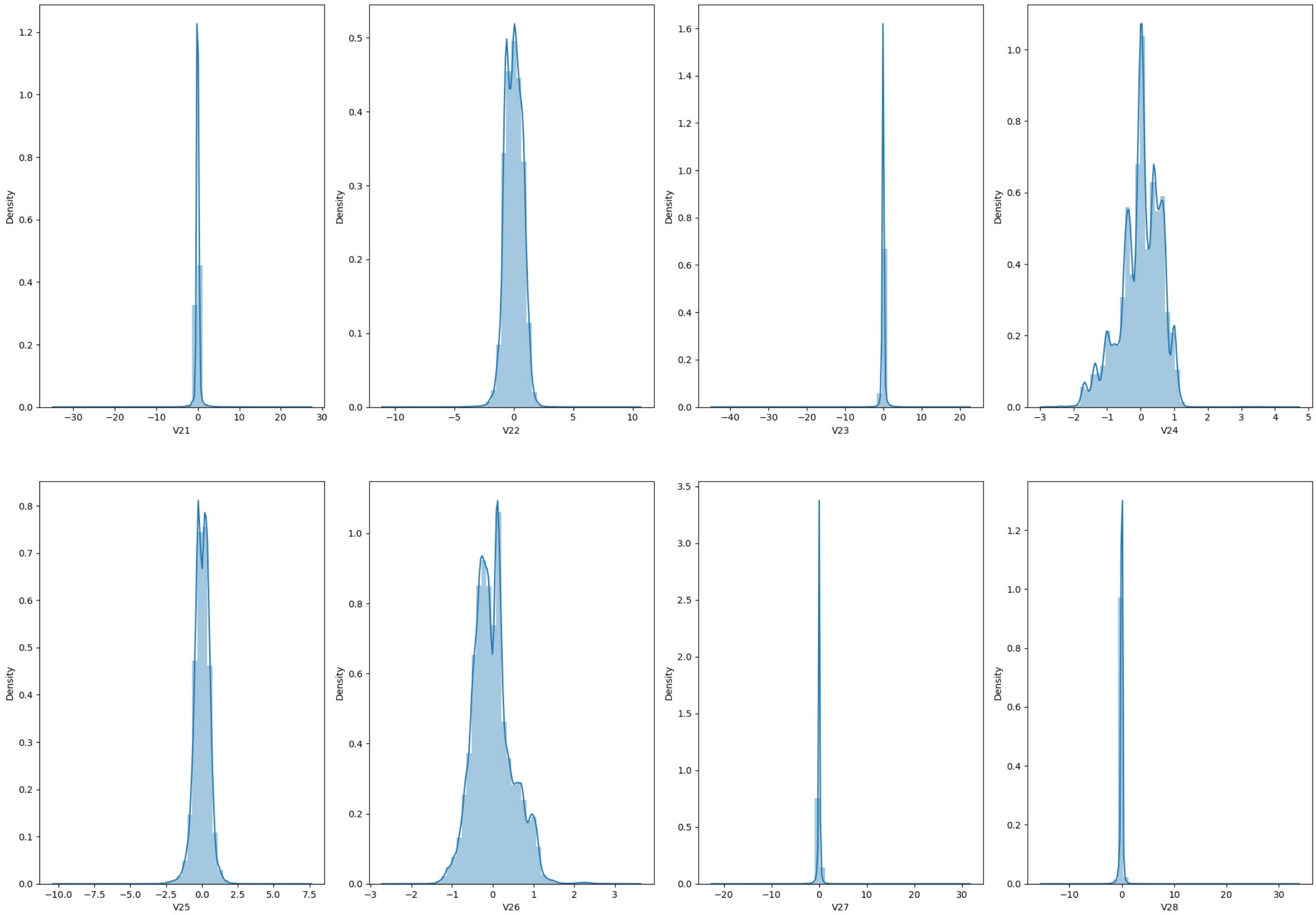
Out[6]:
```
<AxesSubplot:xlabel='Class', ylabel='count'>
```



In [7]:
```
df_temp = df.drop(columns=['Time', 'Amount', 'Class'], axis=1)

# create dist plots
fig, ax = plt.subplots(ncols=4, nrows=7, figsize=(20, 50))
index = 0
ax = ax.flatten()

for col in df_temp.columns:
    sns.distplot(df_temp[col], ax=ax[index])
    index += 1
plt.tight_layout(pad=0.5, w_pad=0.5, h_pad=5)
```

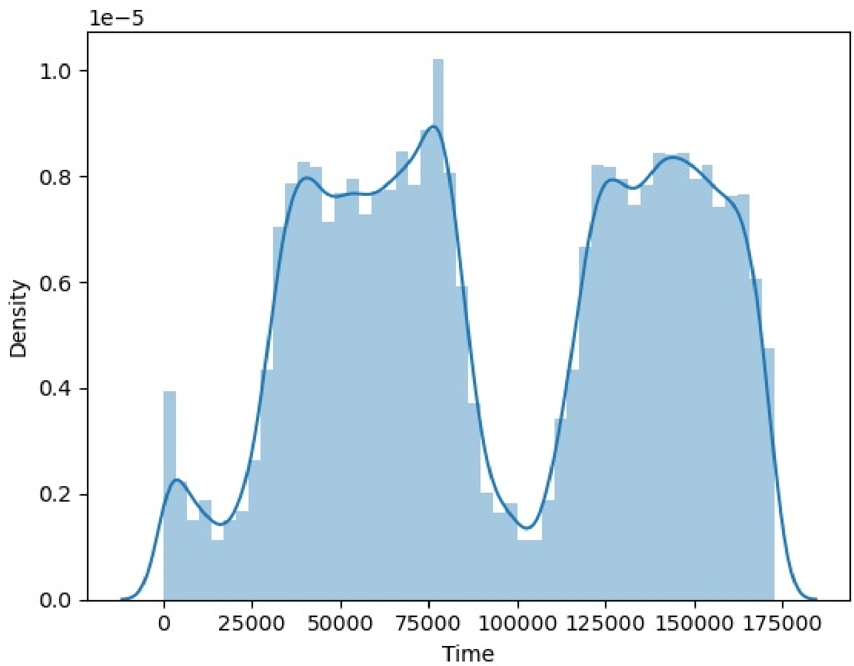



Density
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V20



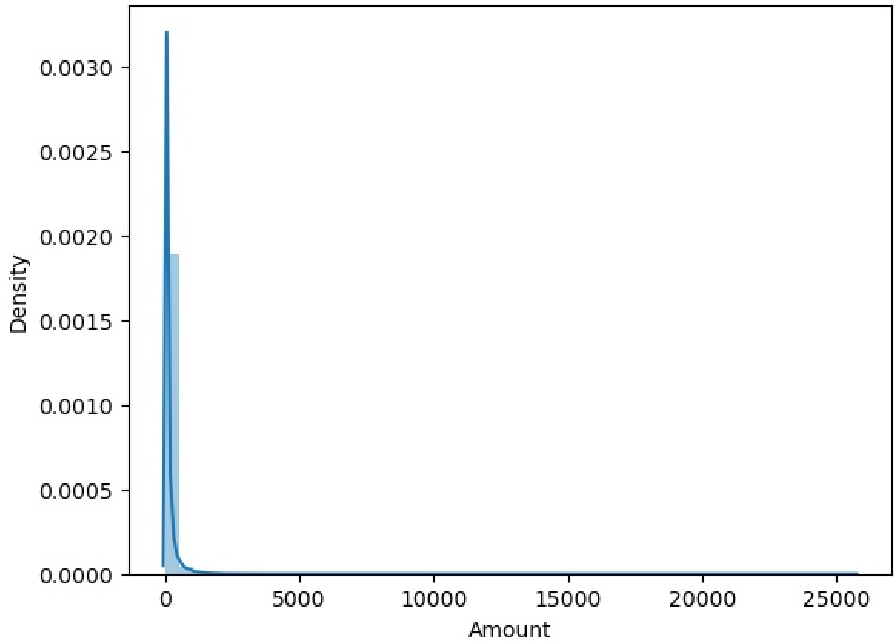
In [8]:
```
sns.distplot(df['Time'])
```

Out[8]:
```
<AxesSubplot:xlabel='Time', ylabel='Density'>
```



In [9]:
```
sns.distplot(df['Amount'])
```

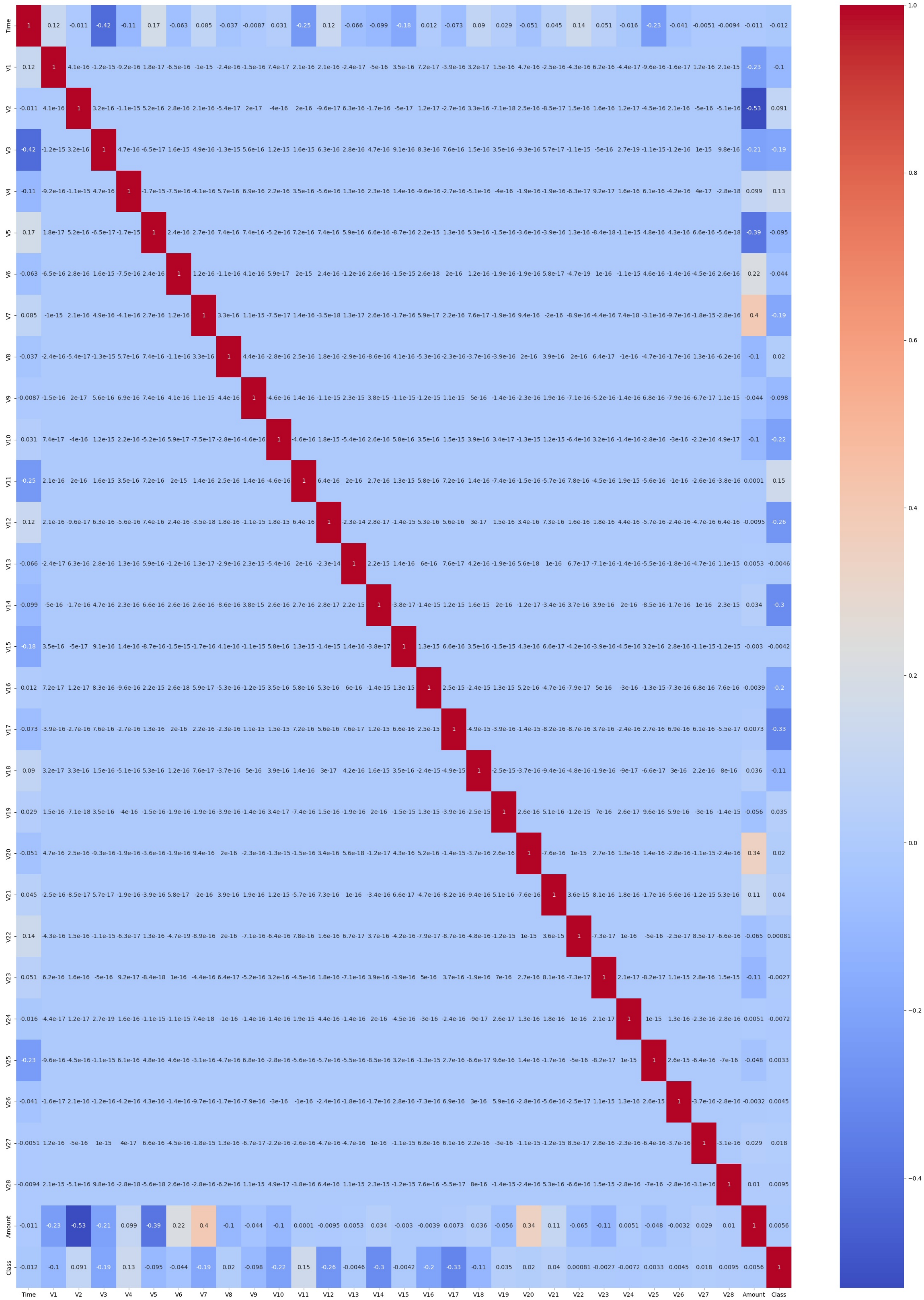
Out[9]:
```
<AxesSubplot:xlabel='Amount', ylabel='Density'>
```



Coorelation Matrix

In [10]:
```
corr = df.corr()
plt.figure(figsize=(30,40))
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Out[10]: <AxesSubplot:>



Input split

In [11]:
```
X = df.drop(columns=['Class'], axis=1)
y = df['Class']
```

Standard scaling

In [12]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_scaler = sc.fit_transform(X)
```

In [13]:
```
x_scaler[-1]
```

Out[13]:
```
array([ 1.64205773, -0.27233093, -0.11489898,  0.46386564, -0.35757    ,
       -0.00908946, -0.48760183,  1.27476937, -0.3471764 ,  0.44253246,
       -0.84072963, -1.01934641, -0.0315383 , -0.18898634, -0.08795849,
        0.04515766, -0.34535763, -0.77752147,  0.1997554 , -0.31462479,
        0.49673933,  0.35541083,  0.8861488 ,  0.6033653 ,  0.01452561,
       -0.90863123, -1.69685342, -0.00598394,  0.04134999,  0.51435531])
```

Model Training

In [14]:
```
# train test split
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, f1_score
x_train, x_test, y_train, y_test = train_test_split(x_scaler, y, test_size=0.25, random_state=42, stratify=y)
```

In [15]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
# training
model.fit(x_train, y_train)
# testing
y_pred = model.predict(x_test)
print(classification_report(y_test, y_pred))
print("F1 Score:",f1_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     71079
           1       0.85      0.63      0.72       123

    accuracy                           1.00     71202
   macro avg       0.92      0.81      0.86     71202
weighted avg       1.00      1.00      1.00     71202

F1 Score: 0.719626168224299
```

In [20]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
# training
model.fit(x_train, y_train)
# testing
y_pred = model.predict(x_test)
print(classification_report(y_test, y_pred))
print("F1 Score:",f1_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     71079
           1       0.96      0.78      0.86       123

    accuracy                           1.00     71202
   macro avg       0.98      0.89      0.93     71202
weighted avg       1.00      1.00      1.00     71202

F1 Score: 0.8609865470852018
```

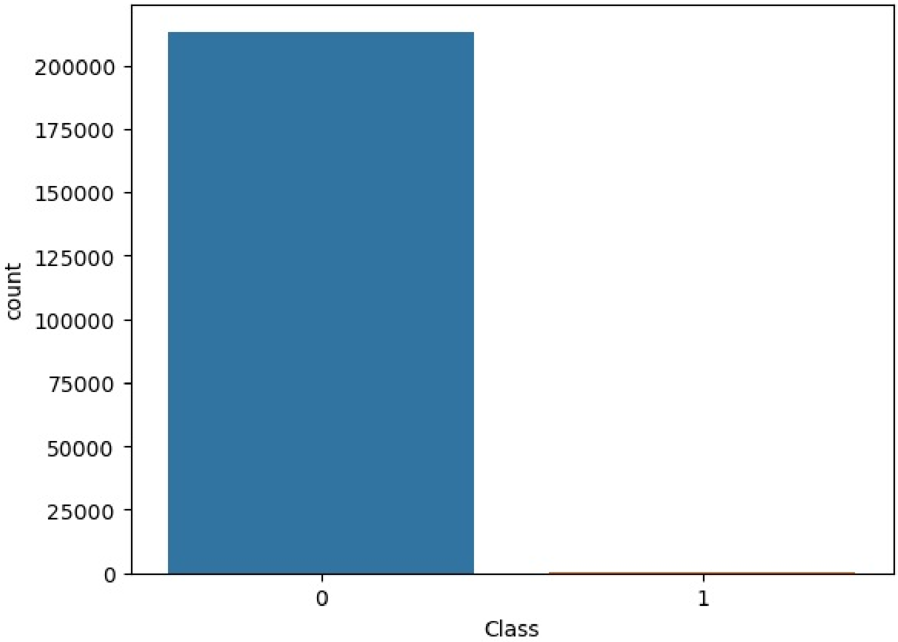
In [17]:
```
from xgboost import XGBClassifier
model = XGBClassifier(n_jobs=-1)
# training
model.fit(x_train, y_train)
# testing
y_pred = model.predict(x_test)
print(classification_report(y_test, y_pred))
print("F1 Score:",f1_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     71079
           1       0.94      0.79      0.86       123

    accuracy                           1.00     71202
   macro avg       0.97      0.89      0.93     71202
weighted avg       1.00      1.00      1.00     71202

F1 Score: 0.8584070796460177
```

Class Imbalancement

In [18]:
```
sns.countplot(y_train)
```

Out[18]:
```
<AxesSubplot:xlabel='Class', ylabel='count'>
```



In [21]:

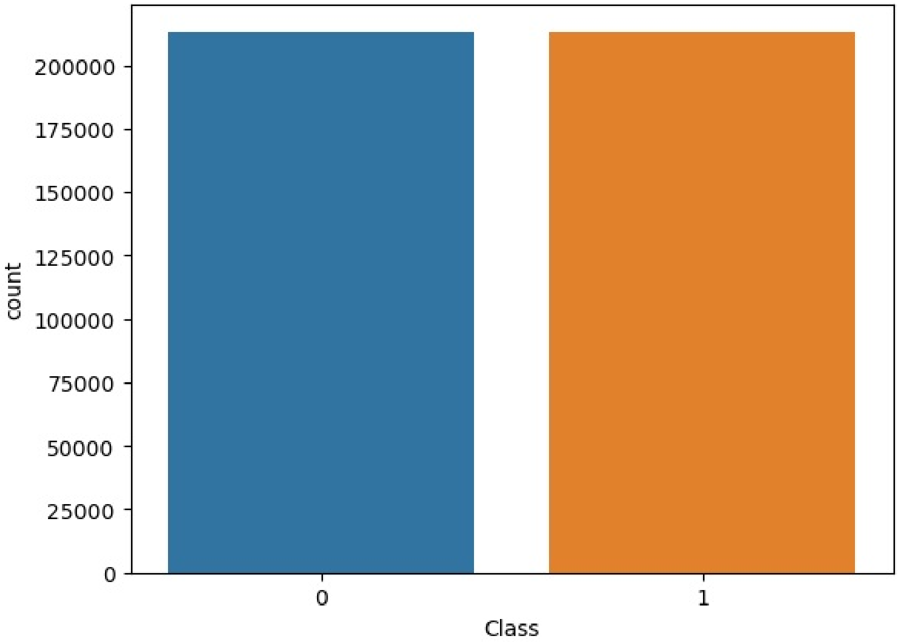
```
# balance the class with equal distribution
from imblearn.over_sampling import SMOTE
over_sample = SMOTE()
x_smote, y_smote = over_sample.fit_resample(x_train, y_train)
```

In [22]:

```
sns.countplot(y_smote)
```

Out[22]:

```
<AxesSubplot:xlabel='Class', ylabel='count'>
```



In [23]:

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
# training
model.fit(x_smote, y_smote)
# testing
y_pred = model.predict(x_test)
print(classification_report(y_test, y_pred))
print("F1 Score:",f1_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     71079
           1       0.06      0.89      0.11       123

    accuracy                           0.98     71202
   macro avg       0.53      0.93      0.55     71202
weighted avg       1.00      0.98      0.99     71202

F1 Score: 0.11219763252702007
```

In [24]:

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_jobs=-1)
# training
```

```python
model.fit(x_smote, y_smote)
# testing
y_pred = model.predict(x_test)
print(classification_report(y_test, y_pred))
print("F1 Score:",f1_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     71079
           1       0.88      0.77      0.82       123

    accuracy                           1.00     71202
   macro avg       0.94      0.89      0.91     71202
weighted avg       1.00      1.00      1.00     71202

F1 Score: 0.8225108225108225
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js