```
#Imported several Python libraries and machine learning-related modules :
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import PowerTransformer
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import StackingClassifier


# Load the creditcard.csv dataset
df = pd.read_csv('/content/drive/MyDrive/project/Credit card fraud/Project 2.csv')


df.shape
```

```
(284807, 31)
```

```
df.head(n=10)
```

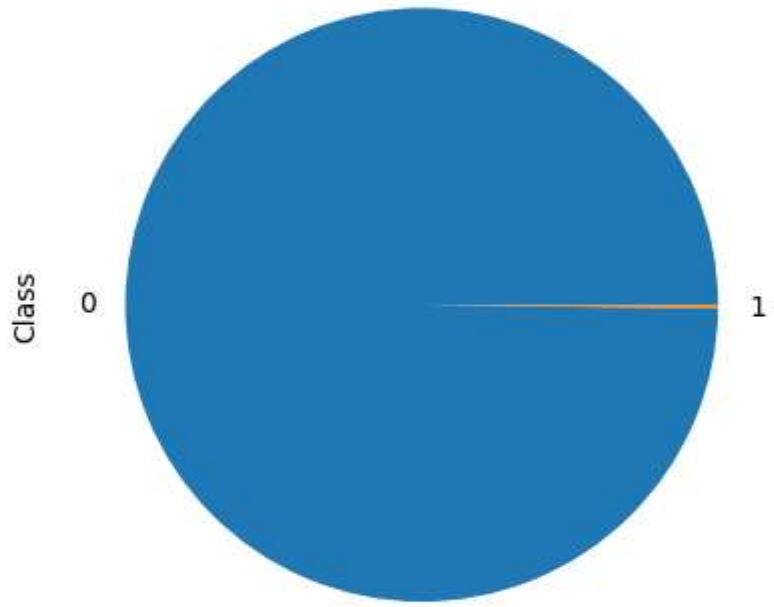
V2	V3	V4	V5	V6	V7	V8	V9	...	V
072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.0183
266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.2257
340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.2479
185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.1083
877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.0094
960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.2082
141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.1677
417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.9434
286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.0734
119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.2469

```
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
```

```
df['Class'].value_counts().plot(kind='pie')
```

```
<Axes: ylabel='Class'>
```




```
df.duplicated().sum()
```

```
1081
```

```
df.drop_duplicates(keep=False,inplace=True)
```

```
# Correcting imbalanced data by oversampling
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

```
print(x.shape,y.shape)
```

```
(282953, 30) (282953,)
```

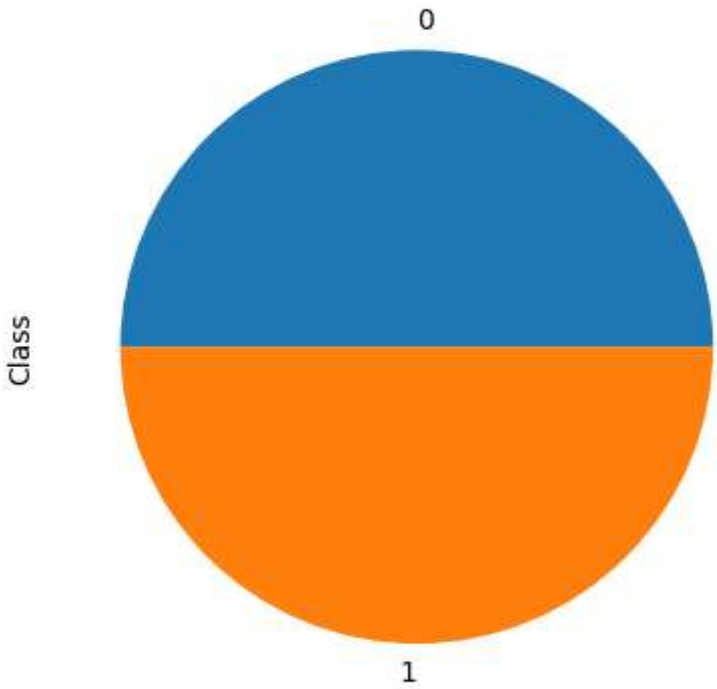
```
smote=SMOTE()
x,y=smote.fit_resample(x,y)
```

```
print(x.shape,y.shape)
```

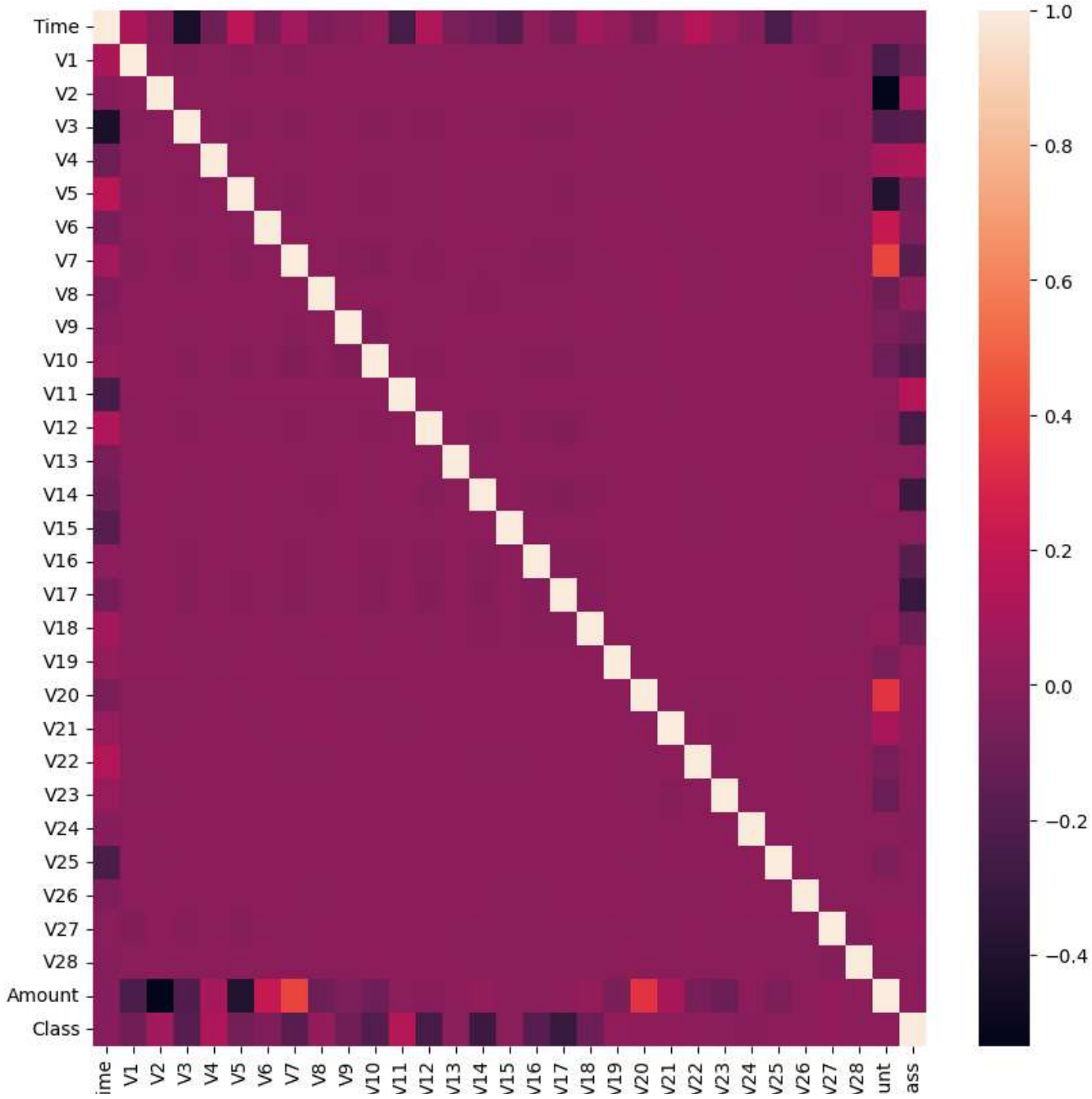
```
(564986, 30) (564986,)
```

```
print(y.value_counts().plot(kind='pie'))
```

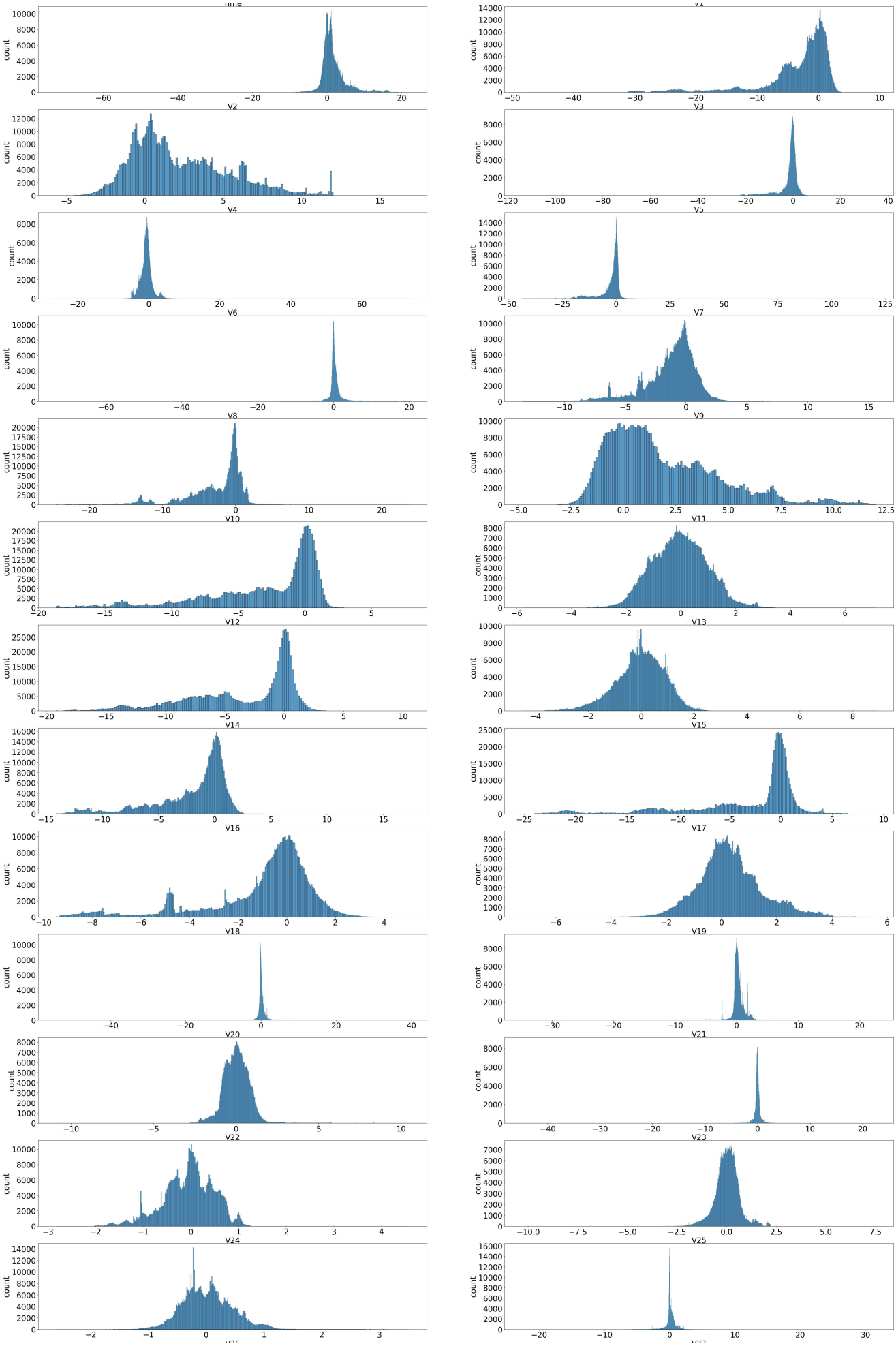
```
Axes(0.22375,0.11;0.5775x0.77)
```




```
plt.figure(figsize=(10,10))
dataplot=sns.heatmap(df.corr())
plt.show()
```




```
# Fixing multi co-linearity
plt.figure(figsize=(50,90))
i=1
for col in x.columns:
    plt.subplot(15,2,i)
    sns.histplot(x[col])
    plt.xticks(fontsize=25)
    plt.yticks(fontsize=25)
    plt.xlabel(col,fontsize=25)
    plt.ylabel("count",fontsize=25)
    i+=1
plt.show()
```


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
count


```
def check_skewness(x):
    skew_limit=0.75
    skew_value=df[x.columns].skew()
    #print(skew_value)
    skew_col=skew_value[abs(skew_value)>skew_limit]
    cols=skew_col.index
    return cols


skewed_col=check_skewness(x)
print(skewed_col)
```

```
     Index(['V1', 'V2', 'V3', 'V5', 'V6', 'V7', 'V8', 'V10', 'V12', 'V14', 'V16',
            'V17', 'V20', 'V21', 'V23', 'V28', 'Amount'],
           dtype='object')
```

```
pt=PowerTransformer(standardize=False)
x[skewed_col]=pt.fit_transform(x[skewed_col])


x.duplicated().sum()
```

```
     0
```

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=1)

print(xtrain.shape)
print(ytrain.shape)
print(xtest.shape)
print(ytest.shape)
```

```
     (451988, 30)
     (451988,)
     (112998, 30)
     (112998,)
```

```
sc=StandardScaler()
xtrain=sc.fit_transform(xtrain)
xtest=sc.transform(xtest)
```

```
def model_evaluate(model):
    model.fit(xtrain,ytrain)
    acc=model.score(xtest,ytest)

    print("Model Name",model)
    print("Acuuracy",acc)


lr=LogisticRegression()
svm=SVC()
dt=DecisionTreeClassifier(max_depth=6)
rf=RandomForestClassifier(max_samples=0.9)
knn=KNeighborsClassifier(n_neighbors=5)

models=[lr,dt,rf,knn]

for model in models:
    model_evaluate(model)
```

```
 Model Name LogisticRegression()
 Acuuracy 0.9797430043009611
 Model Name DecisionTreeClassifier(max_depth=6)
 Acuuracy 0.9747075169472026
 Model Name RandomForestClassifier(max_samples=0.9)
 Acuuracy 0.9999115028584576
 Model Name KNeighborsClassifier()
 Acuuracy 0.9992477742968902
```

```
base_models=[('RF',RandomForestClassifier(max_samples=0.9)),('knn',KNeighborsClassifier(n_ne
meta_model = LogisticRegression()
stacking_model = StackingClassifier(estimators=base_models, final_estimator=meta_model, pass


stacking_model.fit(xtrain, ytrain)
acc=stacking_model.score(xtest,ytest)


from sklearn.metrics import confusion_matrix
y_pred = stacking_model.predict(xtest)
conf_matrix = confusion_matrix(ytest, y_pred)
sns.heatmap(conf_matrix, annot = True, fmt='g')
```

```
<Axes: >
```