# Title : Credit Card Fraud Detection

## Description : Detecting credit card fraud transactions using machine learning techniques.

## Problem Statement: Building a model that accuratley identifies fraudlent trasctions.

## Desired Outcome: Minimizing the financial loss of both credit card holders and issuers by identifying fraud transactions. ¶

#### Importing required libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

### Load the data

In [2]:
```
df = pd.read_csv ("C:/Users/sande/Downloads/archive (6)/creditcard.csv")
```

In [3]:
```
df
```

Out[3]:

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 | 0.462388 | 0.239599 | 0 |
| **1** | 0.0 | 1.191857 | 0.266151 | 0.166480 | 0.448154 | 0.060018 | -0.082361 | -0.078803 | 0 |
| **2** | 1.0 | -1.358354 | -1.340163 | 1.773209 | 0.379780 | -0.503198 | 1.800499 | 0.791461 | 0 |
| **3** | 1.0 | -0.966272 | -0.185226 | 1.792993 | -0.863291 | -0.010309 | 1.247203 | 0.237609 | 0 |
| **4** | 2.0 | -1.158233 | 0.877737 | 1.548718 | 0.403034 | -0.407193 | 0.095921 | 0.592941 | -0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **284802** | 172786.0 | -11.881118 | 10.071785 | -9.834783 | -2.066656 | -5.364473 | -2.606837 | -4.918215 | 7 |
| **284803** | 172787.0 | -0.732789 | -0.055080 | 2.035030 | -0.738589 | 0.868229 | 1.058415 | 0.024330 | 0 |
| **284804** | 172788.0 | 1.919565 | -0.301254 | -3.249640 | -0.557828 | 2.630515 | 3.031260 | -0.296827 | 0 |
| **284805** | 172788.0 | -0.240440 | 0.530483 | 0.702510 | 0.689799 | -0.377961 | 0.623708 | -0.686180 | 0 |
| **284806** | 172792.0 | -0.533413 | -0.189733 | 0.703337 | -0.506271 | -0.012546 | -0.649617 | 1.577006 | -0 |

284807 rows × 31 columns

## Exploratory data analysis

In [4]:
```
df.columns
```

Out[4]:
```
Index(['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', 'Amount',
       'Class'],
      dtype='object')
```

In [5]:
```python
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
```

## Data Cleaning and Preprocessing

In [6]:
```
df.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
```

#### statistical summary

In [7]:
```
df.describe().T
```

Out[7]:

| | count | mean | std | min | 25% | 50% | |
|---|---|---|---|---|---|---|---|
| **Time** | 284807.0 | 9.481386e+04 | 47488.145955 | 0.000000 | 54201.500000 | 84692.000000 | 139320. |
| **V1** | 284807.0 | 1.168375e-15 | 1.958696 | -56.407510 | -0.920373 | 0.018109 | 1. |
| **V2** | 284807.0 | 3.416908e-16 | 1.651309 | -72.715728 | -0.598550 | 0.065486 | 0. |
| **V3** | 284807.0 | -1.379537e-15 | 1.516255 | -48.325589 | -0.890365 | 0.179846 | 1. |
| **V4** | 284807.0 | 2.074095e-15 | 1.415869 | -5.683171 | -0.848640 | -0.019847 | 0. |
| **V5** | 284807.0 | 9.604066e-16 | 1.380247 | -113.743307 | -0.691597 | -0.054336 | 0 |
| **V6** | 284807.0 | 1.487313e-15 | 1.332271 | -26.160506 | -0.768296 | -0.274187 | 0. |
| **V7** | 284807.0 | -5.556467e-16 | 1.237094 | -43.557242 | -0.554076 | 0.040103 | 0. |
| **V8** | 284807.0 | 1.213481e-16 | 1.194353 | -73.216718 | -0.208630 | 0.022358 | 0. |
| **V9** | 284807.0 | -2.406331e-15 | 1.098632 | -13.434066 | -0.643098 | -0.051429 | 0. |
| **V10** | 284807.0 | 2.239053e-15 | 1.088850 | -24.588262 | -0.535426 | -0.092917 | 0. |
| **V11** | 284807.0 | 1.673327e-15 | 1.020713 | -4.797473 | -0.762494 | -0.032757 | 0. |
| **V12** | 284807.0 | -1.247012e-15 | 0.999201 | -18.683715 | -0.405571 | 0.140033 | 0. |
| **V13** | 284807.0 | 8.190001e-16 | 0.995274 | -5.791881 | -0.648539 | -0.013568 | 0. |
| **V14** | 284807.0 | 1.207294e-15 | 0.958596 | -19.214325 | -0.425574 | 0.050601 | 0. |
| **V15** | 284807.0 | 4.887456e-15 | 0.915316 | -4.498945 | -0.582884 | 0.048072 | 0. |
| **V16** | 284807.0 | 1.437716e-15 | 0.876253 | -14.129855 | -0.468037 | 0.066413 | 0. |
| **V17** | 284807.0 | -3.772171e-16 | 0.849337 | -25.162799 | -0.483748 | -0.065676 | 0. |
| **V18** | 284807.0 | 9.564149e-16 | 0.838176 | -9.498746 | -0.498850 | -0.003636 | 0. |
| **V19** | 284807.0 | 1.039917e-15 | 0.814041 | -7.213527 | -0.456299 | 0.003735 | 0. |
| **V20** | 284807.0 | 6.406204e-16 | 0.770925 | -54.497720 | -0.211721 | -0.062481 | 0. |
| **V21** | 284807.0 | 1.654067e-16 | 0.734524 | -34.830382 | -0.228395 | -0.029450 | 0. |
| **V22** | 284807.0 | -3.568593e-16 | 0.725702 | -10.933144 | -0.542350 | 0.006782 | 0. |
| **V23** | 284807.0 | 2.578648e-16 | 0.624460 | -44.807735 | -0.161846 | -0.011193 | 0. |
| **V24** | 284807.0 | 4.473266e-15 | 0.605647 | -2.836627 | -0.354586 | 0.040976 | 0. |
| **V25** | 284807.0 | 5.340915e-16 | 0.521278 | -10.295397 | -0.317145 | 0.016594 | 0. |
| **V26** | 284807.0 | 1.683437e-15 | 0.482227 | -2.604551 | -0.326984 | -0.052139 | 0. |
| **V27** | 284807.0 | -3.660091e-16 | 0.403632 | -22.565679 | -0.070840 | 0.001342 | 0. |
| **V28** | 284807.0 | -1.227390e-16 | 0.330083 | -15.430084 | -0.052960 | 0.011244 | 0. |
| **Amount** | 284807.0 | 8.834962e+01 | 250.120109 | 0.000000 | 5.600000 | 22.000000 | 77. |
| **Class** | 284807.0 | 1.727486e-03 | 0.041527 | 0.000000 | 0.000000 | 0.000000 | 0. |

In [8]:
```
df.drop(columns = "Time",inplace = True)
```

In [9]:
```
df
```

Out[9]:

| | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 | 0.462388 | 0.239599 | 0.098698 | 0 |
| **1** | 1.191857 | 0.266151 | 0.166480 | 0.448154 | 0.060018 | -0.082361 | -0.078803 | 0.085102 | -0 |
| **2** | -1.358354 | -1.340163 | 1.773209 | 0.379780 | -0.503198 | 1.800499 | 0.791461 | 0.247676 | -1 |
| **3** | -0.966272 | -0.185226 | 1.792993 | -0.863291 | -0.010309 | 1.247203 | 0.237609 | 0.377436 | -1 |
| **4** | -1.158233 | 0.877737 | 1.548718 | 0.403034 | -0.407193 | 0.095921 | 0.592941 | -0.270533 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **284802** | -11.881118 | 10.071785 | -9.834783 | -2.066656 | -5.364473 | -2.606837 | -4.918215 | 7.305334 | 1 |
| **284803** | -0.732789 | -0.055080 | 2.035030 | -0.738589 | 0.868229 | 1.058415 | 0.024330 | 0.294869 | 0 |
| **284804** | 1.919565 | -0.301254 | -3.249640 | -0.557828 | 2.630515 | 3.031260 | -0.296827 | 0.708417 | 0 |
| **284805** | -0.240440 | 0.530483 | 0.702510 | 0.689799 | -0.377961 | 0.623708 | -0.686180 | 0.679145 | 0 |
| **284806** | -0.533413 | -0.189733 | 0.703337 | -0.506271 | -0.012546 | -0.649617 | 1.577006 | -0.414650 | 0 |

284807 rows × 30 columns

In [10]:
```
df.dtypes
```

Out[10]:
```
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 [11]:
```
df.duplicated()
```

Out[11]:
```
0         False
1         False
2         False
3         False
4         False
          ...
284802    False
284803    False
284804    False
284805    False
284806    False
Length: 284807, dtype: bool
```

In [12]:
```
df.drop_duplicates()
```

Out[12]:

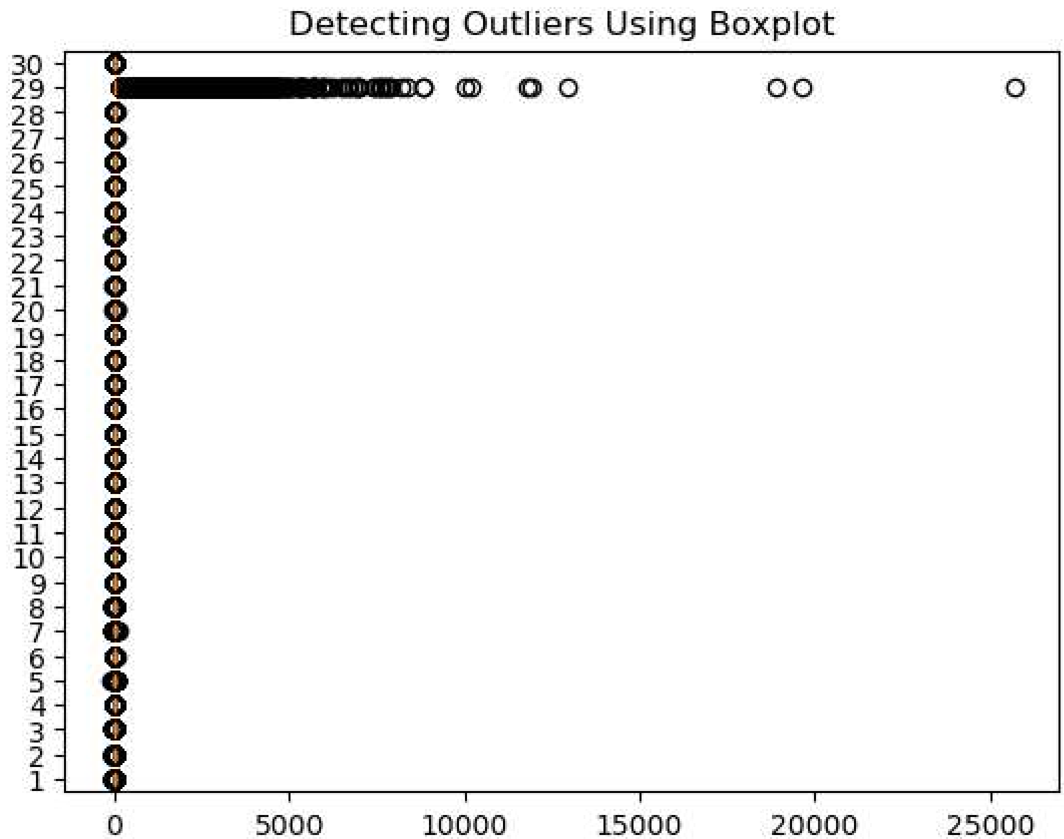
| | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 |
|---|---|---|---|---|---|---|---|---|
| **0** | -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 | 0.462388 | 0.239599 | 0.098698 |
| **1** | 1.191857 | 0.266151 | 0.166480 | 0.448154 | 0.060018 | -0.082361 | -0.078803 | 0.085102 |
| **2** | -1.358354 | -1.340163 | 1.773209 | 0.379780 | -0.503198 | 1.800499 | 0.791461 | 0.247676 |
| **3** | -0.966272 | -0.185226 | 1.792993 | -0.863291 | -0.010309 | 1.247203 | 0.237609 | 0.377436 |
| **4** | -1.158233 | 0.877737 | 1.548718 | 0.403034 | -0.407193 | 0.095921 | 0.592941 | -0.270533 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **284802** | -11.881118 | 10.071785 | -9.834783 | -2.066656 | -5.364473 | -2.606837 | -4.918215 | 7.305334 |
| **284803** | -0.732789 | -0.055080 | 2.035030 | -0.738589 | 0.868229 | 1.058415 | 0.024330 | 0.294869 |
| **284804** | 1.919565 | -0.301254 | -3.249640 | -0.557828 | 2.630515 | 3.031260 | -0.296827 | 0.708417 |
| **284805** | -0.240440 | 0.530483 | 0.702510 | 0.689799 | -0.377961 | 0.623708 | -0.686180 | 0.679145 |
| **284806** | -0.533413 | -0.189733 | 0.703337 | -0.506271 | -0.012546 | -0.649617 | 1.577006 | -0.414650 |

275663 rows × 30 columns

## outliers removing

using boxplot detecting outliers

In [13]:
```python
plt.boxplot(df,vert= False)
plt.title("Detecting Outliers Using Boxplot")
plt.show()
```




removing outliers

In [14]:
```python
# Function to remove outliers using the IQR method
def remove_outliers_iqr(df):
    # Select only numerical columns
    df_num = df.select_dtypes(include=['float64', 'int64'])

    # Calculate Q1 (25th percentile) and Q3 (75th percentile)
    Q1 = df_num.quantile(0.25)
    Q3 = df_num.quantile(0.75)

    # Calculate the IQR
    IQR = Q3 - Q1

    # Filter the dataframe
    df_no_outliers = df[~((df_num < (Q1 - 1.5 * IQR)) | (df_num > (Q3 + 1.5 *

    return df_no_outliers

# Applying the function
df_no_outliers = remove_outliers_iqr(df)
```

In [15]:
```
df_no_outliers
```

Out[15]:

| | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 | 0.462388 | 0.239599 | 0.098698 | 0. |
| **1** | 1.191857 | 0.266151 | 0.166480 | 0.448154 | 0.060018 | -0.082361 | -0.078803 | 0.085102 | -0. |
| **3** | -0.966272 | -0.185226 | 1.792993 | -0.863291 | -0.010309 | 1.247203 | 0.237609 | 0.377436 | -1. |
| **4** | -1.158233 | 0.877737 | 1.548718 | 0.403034 | -0.407193 | 0.095921 | 0.592941 | -0.270533 | 0. |
| **5** | -0.425966 | 0.960523 | 1.141109 | -0.168252 | 0.420987 | -0.029728 | 0.476201 | 0.260314 | -0. |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **284796** | 1.884849 | -0.143540 | -0.999943 | 1.506772 | -0.035300 | -0.613638 | 0.190241 | -0.249058 | 0. |
| **284797** | -0.241923 | 0.712247 | 0.399806 | -0.463406 | 0.244531 | -1.343668 | 0.929369 | -0.206210 | 0. |
| **284800** | 2.039560 | -0.175233 | -1.196825 | 0.234580 | -0.008713 | -0.726571 | 0.017050 | -0.118228 | 0. |
| **284801** | 0.120316 | 0.931005 | -0.546012 | -0.745097 | 1.130314 | -0.235973 | 0.812722 | 0.115093 | -0. |
| **284803** | -0.732789 | -0.055080 | 2.035030 | -0.738589 | 0.868229 | 1.058415 | 0.024330 | 0.294869 | 0. |

146319 rows × 30 columns

## Standardization

In [16]:
```
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
scaled_df = scale.fit_transform(df)
```

In [17]:
```
scaled_df
```

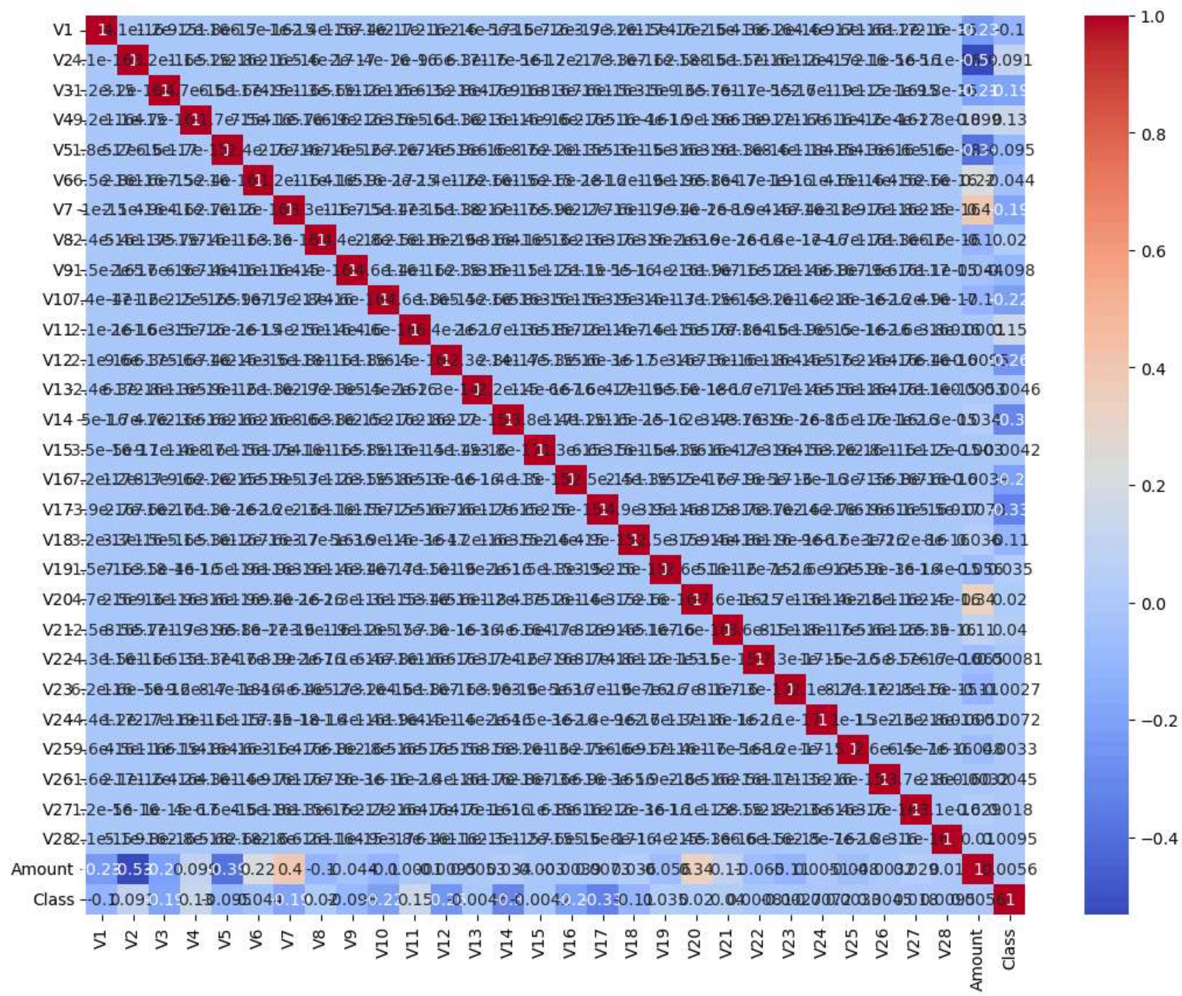
Out[17]:
```
array([[-0.69424232, -0.04407492,  1.6727735 , ..., -0.06378115,
         0.24496426, -0.04159898],
       [ 0.60849633,  0.16117592,  0.1097971 , ...,  0.04460752,
        -0.34247454, -0.04159898],
       [-0.69350046, -0.81157783,  1.16946849, ..., -0.18102083,
         1.16068593, -0.04159898],
       ...,
       [ 0.98002374, -0.18243372, -2.14320514, ..., -0.0804672 ,
        -0.0818393 , -0.04159898],
       [-0.12275539,  0.32125034,  0.46332013, ...,  0.31668678,
        -0.31324853, -0.04159898],
       [-0.27233093, -0.11489898,  0.46386564, ...,  0.04134999,
         0.51435531, -0.04159898]])
```

In [18]:
```
df['Class'].value_counts()
```

Out[18]:
```
0    284315
1       492
Name: Class, dtype: int64
```

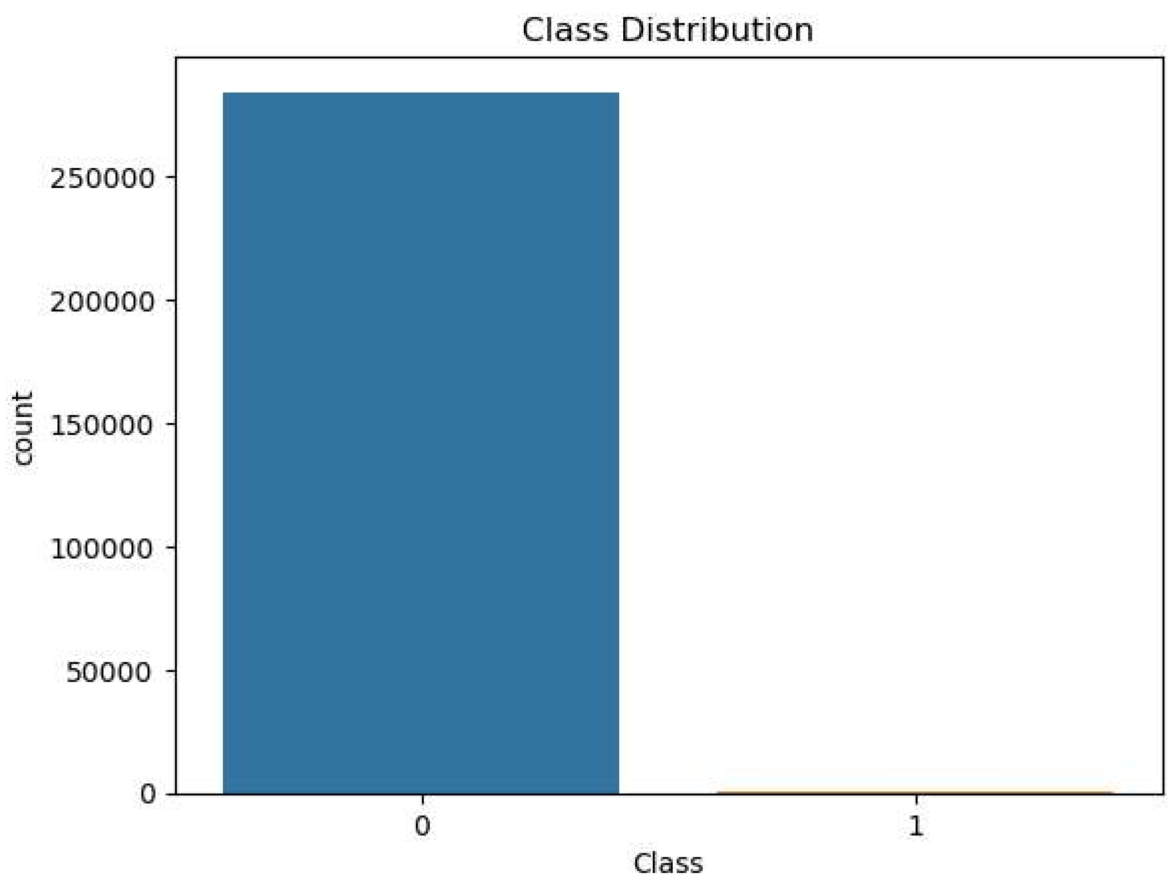
In [19]:
```python
plt.figure(figsize=(13,10))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
```

Out[19]: <Axes: >



## Visualization of (Fraud vs Non Fruad)

In [20]:
```
sns.countplot(x='Class', data=df)
plt.title('Class Distribution')
plt.show()
```




## Data balancing

In [21]:
```
X=df.iloc[:,:-1] # selects all rows (:) and all columns except the last one (:
Y=df.iloc[:,-1]  # selects all rows (:) and only the last column (-1).
```

In [22]:
```
print(X.shape)
print(Y.shape)
```

```
(284807, 29)
(284807,)
```

In [24]:
```
from imblearn.under_sampling import RandomUnderSampler
ros = RandomUnderSampler(random_state=0)
X_res,Y_res = ros.fit_resample(X,Y)
```

In [25]:
```
X.shape
```
Out[25]:
```
(284807, 29)
```
In [26]:
```
X_res.shape
```
Out[26]:
```
(984, 29)
```
In [27]:
```
Y.value_counts()
```
Out[27]:
```
0    284315
1       492
Name: Class, dtype: int64
```
In [28]:
```
Y_res.shape
```
Out[28]:
```
(984,)
```
In [29]:
```
Y_res.value_counts()
```
Out[29]:
```
0    492
1    492
Name: Class, dtype: int64
```

## Train_Test_Split

In [30]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X_res,Y_res,test_size=0.2, ra
```
In [31]:
```
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```
```
(787, 29)
(787,)
(197, 29)
(197,)
```

## Data modeling

y variable is categorical data then we implement logistic regression

In [32]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

In [33]:
```
model
```

Out[33]:
```
LogisticRegression()
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [34]:
```
model = model.fit(X_train,Y_train)
```

```
C:\Users\sande\AppData\Roaming\Python\Python311\site-packages\sklearn\linear_
model\_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=
1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sciki
t-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(
```

In [35]:
```
Y_predict = model.predict(X_test)
```

In [36]:
```
Y_true = Y_test
```

**Model Evalution**

In [37]:
```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(Y_predict,Y_true))
```

```
              precision    recall  f1-score   support

           0       0.96      0.90      0.93       106
           1       0.89      0.96      0.92        91

    accuracy                           0.92       197
   macro avg       0.92      0.93      0.92       197
weighted avg       0.93      0.92      0.92       197
```

In [38]:
```
print(confusion_matrix(Y_true,Y_predict))
```

```
[[95  4]
 [11 87]]
```

In [39]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [40]:
```
model = RandomForestClassifier()
```

In [41]:
```
model = model.fit(X_train,Y_train)
```

In [42]:
```
Y_predict = model.predict(X_test)
```

In [43]:
```
Y_true = Y_test
```

In [44]:
```
from sklearn.metrics import classification_report
print(classification_report(Y_predict,Y_true))
```

```
              precision    recall  f1-score   support

           0       0.99      0.89      0.94       110
           1       0.88      0.99      0.93        87

    accuracy                           0.93       197
   macro avg       0.93      0.94      0.93       197
weighted avg       0.94      0.93      0.93       197
```

In [45]:
```
print(confusion_matrix(Y_true,Y_predict))
```

```
[[98  1]
 [12 86]]
```

# Final Result

The Random Forest model successfully detects fraudulent transactions with high accuracy. The most influential features in fraud detection are V14, V10, V17, V12, and V4. Balancing the dataset significantly improved model performance by reducing bias towards non-fraudulent transactions, leading to better fraud detection.