BINARY CLASS CLASSIFICATION MODEL

- A binary class classification model is a type of machine learning model used to classify input data into one of two possible classes or categories.
- The term "binary" refers to the fact that there are only two distinct classes, often denoted as 0 and 1, or sometimes as positive and negative classes, or true and false classes.
- Binary classification models aim to learn patterns and relationships from labeled training data.

COMMON ALGORITHMS FOR BINARY CLASS

- Some common algorithms which we have used here :-

(1) Logistic Regression (2) Random Forest (3) Decision Trees (4) K-Nearest Neighbors

POINTS TO REMEMBER TO PREDICT A GOOD CLASSIFICATION MODEL

- Quality of data
- Feature Selection
- Train-Test Split
- Cross Validation
- Choose appropriate algorithm
- Hyper-parameter tuning
- Handle class imbalance
- Ensemble methods
- Evaluation Metrics
- Avoid overfit & underfit
- Feature scaling
- Performance on test set

TOPIC NAME

CREDIT CARD FRAUDULENT DATA

- This dataset contains a large number of credit card transactions, some of which are fraudulent, and others that are legitimate or non-fraudulent.
- TARGET VARIABLE - The dataset contains a binary target variable indicating whether each transaction is fraudulent (class 1) or non-fraudulent (class 0).

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

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
df=pd.read_csv(r"C:\Users\lenovo\Downloads\creditcard.csv")
df
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.27
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.63
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.77
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.00
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.79
...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.11
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.92
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.57
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.80
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.64

284807 rows × 31 columns

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

Out[4]:
```
(284807, 31)
```

In [5]:
```
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
```

In [6]:
```
df.describe()
```

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.8480
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.1943
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.3216
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.0007

8 rows × 31 columns

In [7]:
```
duplicates=df.duplicated().sum()
duplicates
```

Out[7]:
```
1081
```

In [8]:
```
df=df.drop_duplicates()
df
```

Out[8]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.27
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.63
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.77
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.00
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.79
...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.11
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.92
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.57
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.80
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.64

283726 rows × 31 columns

In [9]:
```
df.duplicated().sum()
```

Out[9]:
```
0
```

DATA PRE~PROCESSING

(1) NULL Values Treatment

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
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
```

- No NULL values detected here

(2) OUTLIERS TREATMENT

Checking for Outliers

In [11]:
```
df.head()
```

Out[11]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
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	-0.11(
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	0.10
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	0.90
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	-0.19(
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	-0.13

5 rows × 31 columns

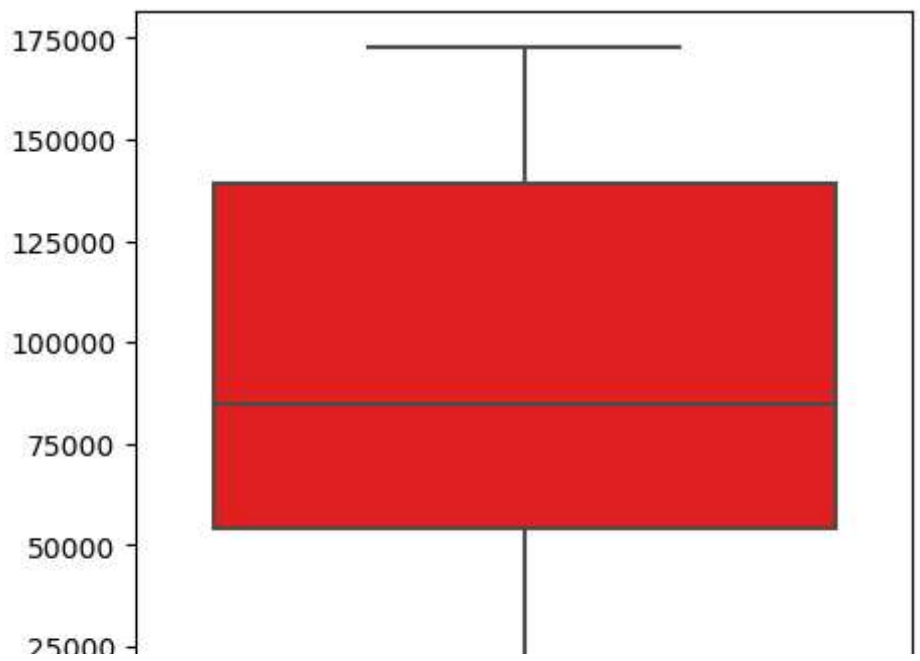
In [12]:
```
df.describe()
```

Out[12]:

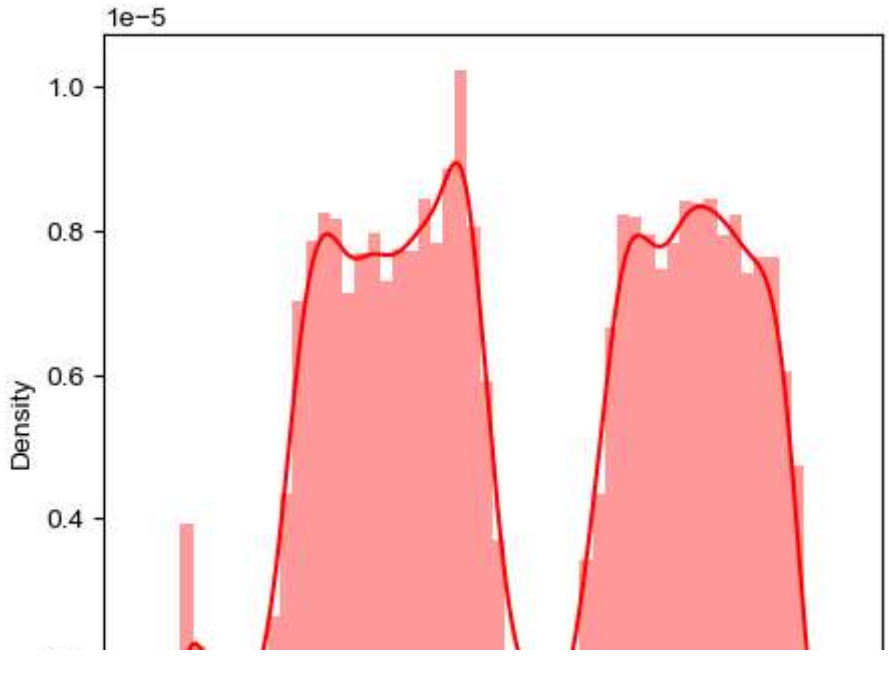
	Time	V1	V2	V3	V4	V5	V6	V7	
count	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000	2837
mean	94811.077600	0.005917	-0.004135	0.001613	-0.002966	0.001828	-0.001139	0.001801	
std	47481.047891	1.948026	1.646703	1.508682	1.414184	1.377008	1.331931	1.227664	
min	0.000000	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307	-26.160506	-43.557242	
25%	54204.750000	-0.915951	-0.600321	-0.889682	-0.850134	-0.689830	-0.769031	-0.552509	
50%	84692.500000	0.020384	0.063949	0.179963	-0.022248	-0.053468	-0.275168	0.040859	
75%	139298.000000	1.316068	0.800283	1.026960	0.739647	0.612218	0.396792	0.570474	
max	172792.000000	2.454930	22.057729	9.382558	16.875344	34.801666	73.301626	120.589494	

8 rows × 31 columns

In [13]:
```
def boxplots(col):
    plt.figure(figsize=(5,5))
    sns.boxplot(df[col],color='red')
    plt.show()

for i in list(df.columns):
    boxplots(i)
```



In [14]:
```
def distplots(col):
    plt.figure(figsize=(5,5))
    sns.distplot(df[col],color='red')
    sns.set(style='dark')
    plt.show()

for i in list(df.columns):
    distplots(i)
```



Treating the Outliers

In [15]:
```
new_df=df.copy()
```

In [16]:
```
new_df.head()
```

Out[16]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
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	-0.11(
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	0.10
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	0.90
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	-0.19(
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	-0.13

5 rows × 31 columns

In [17]:
```
Q1=df['V1'].quantile(0.25)
Q3=df['V1'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [18]:
```
new_df['V1']=np.where(new_df['V1']>Higher_limit,Higher_limit,
                      np.where(new_df['V1']<Lower_limit,Lower_limit,new_df['V1']))
```

In [19]:
```
Q1=df['V2'].quantile(0.25)
Q3=df['V2'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [20]:
```
new_df['V2']=np.where(new_df['V2']>Higher_limit,Higher_limit,
                      np.where(new_df['V2']<Lower_limit,Lower_limit,new_df['V2']))
```

In [21]:
```
Q1=df['V3'].quantile(0.25)
Q3=df['V3'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [22]:
```
new_df['V3']=np.where(new_df['V3']>Higher_limit,Higher_limit,
                      np.where(new_df['V3']<Lower_limit,Lower_limit,new_df['V3']))
```

In [23]:
```
Q1=df['V4'].quantile(0.25)
Q3=df['V4'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [24]:
```
new_df['V4']=np.where(new_df['V4']>Higher_limit,Higher_limit,
                                np.where(new_df['V4']<Lower_limit,Lower_limit,new_df['V4']))
```

In [25]:
```
Q1=df['V5'].quantile(0.25)
Q3=df['V5'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [26]:
```
new_df['V5']=np.where(new_df['V5']>Higher_limit,Higher_limit,
                                np.where(new_df['V5']<Lower_limit,Lower_limit,new_df['V5']))
```

In [27]:
```
Q1=df['V6'].quantile(0.25)
Q3=df['V6'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [28]:
```
new_df['V6']=np.where(new_df['V6']>Higher_limit,Higher_limit,
                                np.where(new_df['V6']<Lower_limit,Lower_limit,new_df['V6']))
```

In [29]:
```
Q1=df['V7'].quantile(0.25)
Q3=df['V7'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [30]:
```
new_df['V7']=np.where(new_df['V7']>Higher_limit,Higher_limit,
                                np.where(new_df['V7']<Lower_limit,Lower_limit,new_df['V7']))
```

In [31]:
```
Q1=df['V8'].quantile(0.25)
Q3=df['V8'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [32]:
```
new_df['V8']=np.where(new_df['V8']>Higher_limit,Higher_limit,
                                np.where(new_df['V8']<Lower_limit,Lower_limit,new_df['V8']))
```

In [33]:
```
Q1=df['V9'].quantile(0.25)
Q3=df['V9'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [34]:
```
new_df['V9']=np.where(new_df['V9']>Higher_limit,Higher_limit,
                                np.where(new_df['V9']<Lower_limit,Lower_limit,new_df['V9']))
```

In [35]:
```
Q1=df['V10'].quantile(0.25)
Q3=df['V10'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [36]:
```
new_df['V10']=np.where(new_df['V10']>Higher_limit,Higher_limit,
                                np.where(new_df['V10']<Lower_limit,Lower_limit,new_df['V10']))
```

In [37]:
```
Q1=df['V11'].quantile(0.25)
Q3=df['V11'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [38]:
```
new_df['V11']=np.where(new_df['V11']>Higher_limit,Higher_limit,
                                np.where(new_df['V11']<Lower_limit,Lower_limit,new_df['V11']))
```

In [39]:
```
Q1=df['V12'].quantile(0.25)
Q3=df['V12'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [40]:
```
new_df['V12']=np.where(new_df['V12']>Higher_limit,Higher_limit,
                                np.where(new_df['V12']<Lower_limit,Lower_limit,new_df['V12']))
```

In [41]:
```
Q1=df['V13'].quantile(0.25)
Q3=df['V13'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [42]:
```
new_df['V13']=np.where(new_df['V13']>Higher_limit,Higher_limit,
                       np.where(new_df['V13']<Lower_limit,Lower_limit,new_df['V13']))
```

In [43]:
```
Q1=df['V14'].quantile(0.25)
Q3=df['V14'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [44]:
```
new_df['V14']=np.where(new_df['V14']>Higher_limit,Higher_limit,
                       np.where(new_df['V14']<Lower_limit,Lower_limit,new_df['V14']))
```

In [45]:
```
Q1=df['V15'].quantile(0.25)
Q3=df['V15'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [46]:
```
new_df['V15']=np.where(new_df['V15']>Higher_limit,Higher_limit,
                       np.where(new_df['V15']<Lower_limit,Lower_limit,new_df['V15']))
```

In [47]:
```
Q1=df['V16'].quantile(0.25)
Q3=df['V16'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [48]:
```
new_df['V16']=np.where(new_df['V16']>Higher_limit,Higher_limit,
                       np.where(new_df['V16']<Lower_limit,Lower_limit,new_df['V16']))
```

In [49]:
```
Q1=df['V17'].quantile(0.25)
Q3=df['V17'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [50]:
```
new_df['V17']=np.where(new_df['V17']>Higher_limit,Higher_limit,
                       np.where(new_df['V17']<Lower_limit,Lower_limit,new_df['V17']))
```

In [51]:
```
Q1=df['V18'].quantile(0.25)
Q3=df['V18'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [52]:
```
new_df['V18']=np.where(new_df['V18']>Higher_limit,Higher_limit,
                       np.where(new_df['V18']<Lower_limit,Lower_limit,new_df['V18']))
```

In [53]:
```
Q1=df['V19'].quantile(0.25)
Q3=df['V19'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [54]:
```
new_df['V19']=np.where(new_df['V19']>Higher_limit,Higher_limit,
                       np.where(new_df['V19']<Lower_limit,Lower_limit,new_df['V19']))
```

In [55]:
```
Q1=df['V20'].quantile(0.25)
Q3=df['V20'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [56]:
```
new_df['V20']=np.where(new_df['V20']>Higher_limit,Higher_limit,
                       np.where(new_df['V20']<Lower_limit,Lower_limit,new_df['V20']))
```

In [57]:
```
Q1=df['V21'].quantile(0.25)
Q3=df['V21'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [58]:
```
new_df['V21']=np.where(new_df['V21']>Higher_limit,Higher_limit,
                       np.where(new_df['V21']<Lower_limit,Lower_limit,new_df['V21']))
```

In [59]:
```
Q1=df['V22'].quantile(0.25)
Q3=df['V22'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [60]:
```
new_df['V22']=np.where(new_df['V22']>Higher_limit,Higher_limit,
                       np.where(new_df['V22']<Lower_limit,Lower_limit,new_df['V22']))
```

In [61]:
```
Q1=df['V23'].quantile(0.25)
Q3=df['V23'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [62]:
```
new_df['V23']=np.where(new_df['V23']>Higher_limit,Higher_limit,
                       np.where(new_df['V23']<Lower_limit,Lower_limit,new_df['V23']))
```

In [63]:
```
Q1=df['V24'].quantile(0.25)
Q3=df['V24'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [64]:
```
new_df['V24']=np.where(new_df['V24']>Higher_limit,Higher_limit,
                       np.where(new_df['V24']<Lower_limit,Lower_limit,new_df['V24']))
```

In [65]:
```
Q1=df['V25'].quantile(0.25)
Q3=df['V25'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [66]:
```
new_df['V25']=np.where(new_df['V25']>Higher_limit,Higher_limit,
                       np.where(new_df['V25']<Lower_limit,Lower_limit,new_df['V25']))
```

In [67]:
```
Q1=df['V26'].quantile(0.25)
Q3=df['V26'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [68]:
```
new_df['V26']=np.where(new_df['V26']>Higher_limit,Higher_limit,
                       np.where(new_df['V26']<Lower_limit,Lower_limit,new_df['V26']))
```

In [69]:
```
Q1=df['V27'].quantile(0.25)
Q3=df['V27'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [70]:
```
new_df['V27']=np.where(new_df['V27']>Higher_limit,Higher_limit,
                       np.where(new_df['V27']<Lower_limit,Lower_limit,new_df['V27']))
```

In [71]:
```
Q1=df['V28'].quantile(0.25)
Q3=df['V28'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [72]:
```
new_df['V28']=np.where(new_df['V28']>Higher_limit,Higher_limit,
                       np.where(new_df['V28']<Lower_limit,Lower_limit,new_df['V28']))
```

In [73]:
```
Q1=df['Amount'].quantile(0.25)
Q3=df['Amount'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [74]:
```
new_df['Amount']=np.where(new_df['Amount']>Higher_limit,Higher_limit,
                       np.where(new_df['Amount']<Lower_limit,Lower_limit,new_df['Amount']))
```

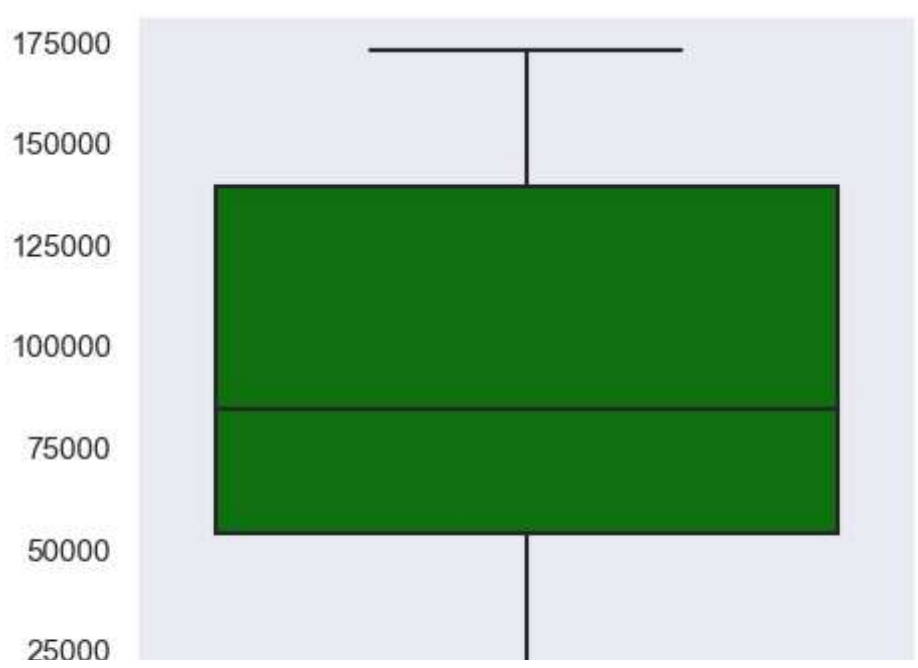
In [75]:
```
new_df.describe()
```

Out[75]:

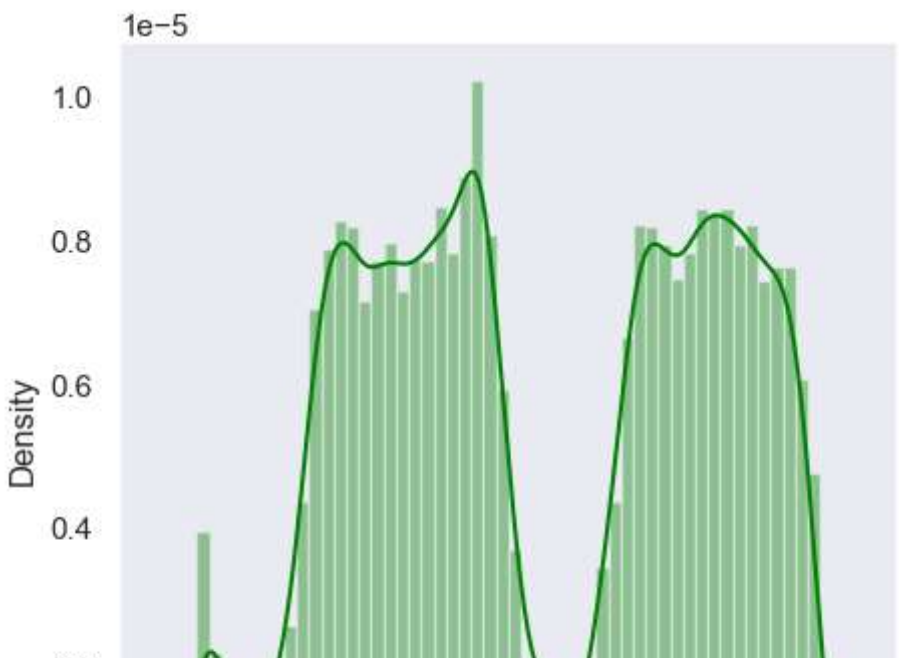
	Time	V1	V2	V3	V4	V5	V6	V7	
count	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000	2837
mean	94811.077600	0.079902	0.046775	0.027858	-0.030700	-0.001586	-0.092059	0.009550	
std	47481.047891	1.590245	1.119493	1.368971	1.309182	1.075933	1.001754	0.863909	
min	0.000000	-4.263980	-2.701226	-3.764645	-3.234807	-2.642901	-2.517765	-2.236984	
25%	54204.750000	-0.915951	-0.600321	-0.889682	-0.850134	-0.689830	-0.769031	-0.552509	
50%	84692.500000	0.020384	0.063949	0.179963	-0.022248	-0.053468	-0.275168	0.040859	
75%	139298.000000	1.316068	0.800283	1.026960	0.739647	0.612218	0.396792	0.570474	
max	172792.000000	2.454930	2.901188	3.901923	3.124319	2.565290	2.145527	2.254949	

8 rows × 31 columns

In [76]:
```
def boxplots(col):
    plt.figure(figsize=(5,5))
    sns.boxplot(new_df[col],color='green')
    plt.show()

for i in list(new_df.columns):
    boxplots(i)
```



In [77]:
```
def distplots(col):
    plt.figure(figsize=(5,5))
    sns.distplot(new_df[col],color='green')
    sns.set(style='dark')
    plt.show()

for i in list(new_df.columns):
    distplots(i)
```



- All Outliers are treated

In [78]:
```
duplicates=df.duplicated().sum()
duplicates
```

Out[78]: 0

(3) ENCODING CONCEPT

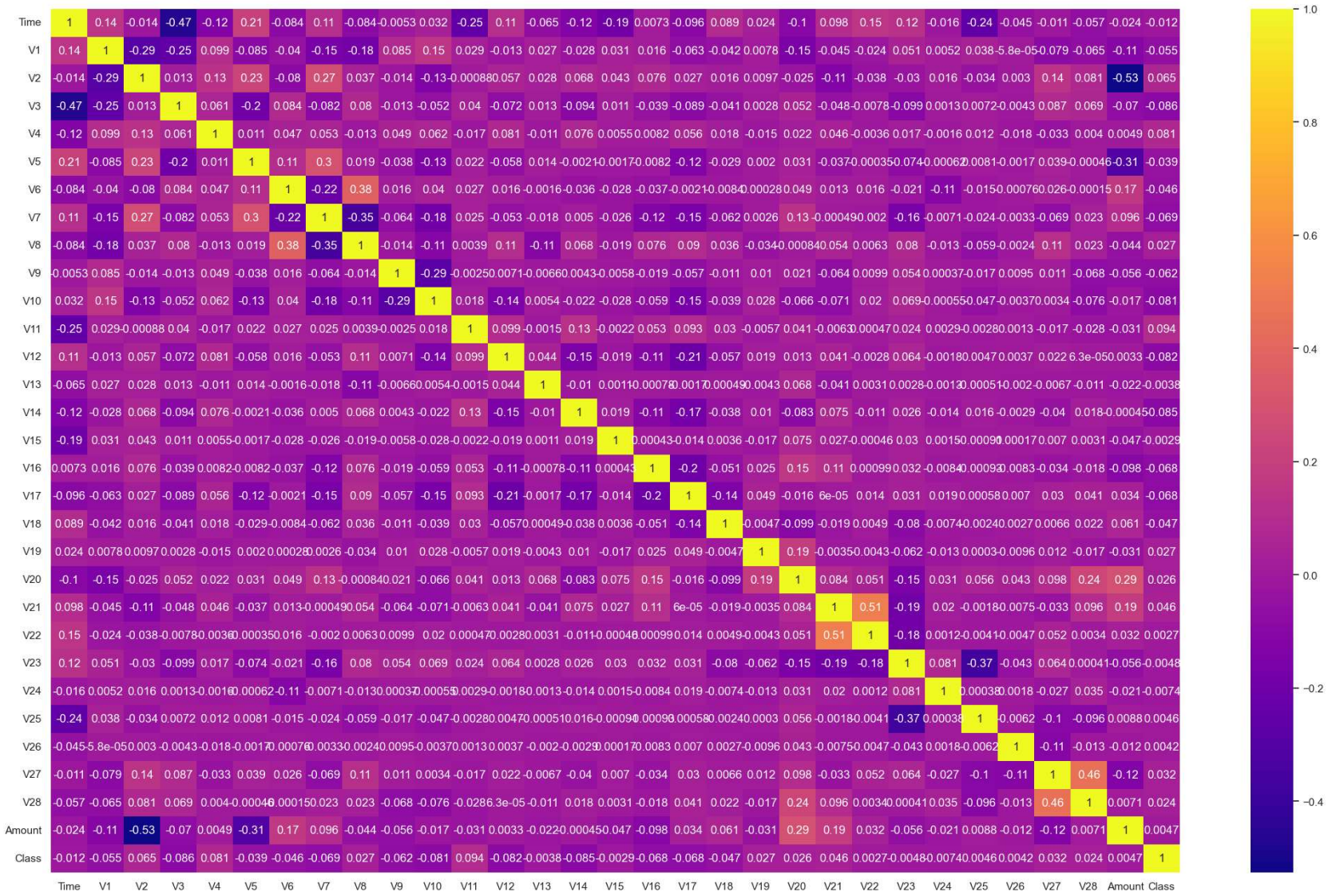
In [79]:
```
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 283726 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    283726 non-null  float64
 1   V1      283726 non-null  float64
 2   V2      283726 non-null  float64
 3   V3      283726 non-null  float64
 4   V4      283726 non-null  float64
 5   V5      283726 non-null  float64
 6   V6      283726 non-null  float64
 7   V7      283726 non-null  float64
 8   V8      283726 non-null  float64
 9   V9      283726 non-null  float64
 10  V10     283726 non-null  float64
 11  V11     283726 non-null  float64
 12  V12     283726 non-null  float64
 13  V13     283726 non-null  float64
 14  V14     283726 non-null  float64
 15  V15     283726 non-null  float64
 16  V16     283726 non-null  float64
 17  V17     283726 non-null  float64
 18  V18     283726 non-null  float64
 19  V19     283726 non-null  float64
 20  V20     283726 non-null  float64
 21  V21     283726 non-null  float64
 22  V22     283726 non-null  float64
 23  V23     283726 non-null  float64
 24  V24     283726 non-null  float64
 25  V25     283726 non-null  float64
 26  V26     283726 non-null  float64
 27  V27     283726 non-null  float64
 28  V28     283726 non-null  float64
 29  Amount  283726 non-null  float64
 30  Class   283726 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 77.3 MB
```

- As all the variables are of INT & FLOAT datatypes, so no need of ENCODING CONCEPT

HEATMAP Visualization to check Multi~Collinearity

In [80]:
```
plt.figure(figsize=(26,16))
corr=new_df.corr()
sns.heatmap(corr,annot=True,cmap='plasma')
plt.show()
```



- As it can be seen there are 4 variables whose values are same (V16 & V17,V8 & V19)
- So clearly we can see there is Multi~Collinearity as the variables are highly correlated with each other
- Now we need to remove one variable from each similar variable pair to remove Multi~Collinearity

In [81]:
```
new_df=new_df.drop(['V16','V19'],axis=1)
new_df
```

Out[81]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798
...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.0	-4.263980	2.901188	-3.764645	-2.066656	-2.642901	-2.517765	-2.236984	1.127502	1.914428	...	0.213454	0.111
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.565290	2.145527	-0.296827	0.708417	0.432454	...	0.232045	0.578
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643

283726 rows × 29 columns

- Now there are no highly correlated variables, so no Multi~Collinearity

(4) FEATURE SCALING

Feature Scaling is only done on Independent Variables

In [82]:
```
new_df
```

Out[82]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.0	-4.263980	2.901188	-3.764645	-2.066656	-2.642901	-2.517765	-2.236984	1.127502	1.914428	...	0.213454	0.
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.565290	2.145527	-0.296827	0.708417	0.432454	...	0.232045	0.
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.

Need to split the dataset into independent & dependent variables

In [82]:
```
x=new_df.drop(['Class'],axis=1)
x.head()
```

Out[82]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	0.251412	-0.018307	0.27
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.069083	-0.225775	-0.63
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.524980	0.247998	0.77
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.208038	-0.108300	0.00
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	0.408542	-0.009431	0.79

5 rows × 28 columns

In [83]:
```
y=new_df['Class']
pd.DataFrame(y)
```

Out[83]:

	Class
0	0
1	0
2	0
3	0
4	0
...	...
284802	0
284803	0
284804	0
284805	0
284806	0

283726 rows × 1 columns

In [84]:
```
from sklearn.preprocessing import StandardScaler
std=StandardScaler()
fs_x=std.fit_transform(x)
pd.DataFrame(fs_x)
```

Out[84]:

	0	1	2	3	4	5	6	7	8	9	...	18	
0	-1.996823	-0.905339	-0.106795	1.832393	1.076136	-0.312970	0.553477	0.266288	0.064273	0.373700	...	0.879771	-0.009
1	-1.996823	0.699237	0.195961	0.101260	0.365766	0.057256	0.009681	-0.102272	0.036182	-0.234530	...	-0.129561	-0.650
2	-1.996802	-0.904425	-1.238900	1.274939	0.313539	-0.466212	1.889249	0.905086	0.372075	-1.471427	...	1.741314	0.812
3	-1.996802	-0.657870	-0.207238	1.289390	-0.635964	-0.008107	1.336920	0.263985	0.640171	-1.346060	...	-0.567169	-0.287
4	-1.996781	-0.778582	0.742268	1.110953	0.331302	-0.376983	0.187652	0.675292	-0.698590	0.819602	...	1.374619	0.017
...	...	...	...	...	...	...	...	...	...	...	...	...	
283721	1.642235	-2.731585	2.549742	-2.770335	-1.555138	-2.454912	-2.421464	-2.600433	2.189875	1.896841	...	2.135737	0.706
283722	1.642257	-0.511048	-0.090983	1.466192	-0.540712	0.808431	1.148462	0.017108	0.469580	0.590794	...	0.275749	0.708
283723	1.642278	1.156845	-0.310881	-2.394136	-0.402640	2.385726	2.233672	-0.354641	1.324008	0.441150	...	0.092398	0.763
283724	1.642278	-0.201442	0.432078	0.492818	0.550344	-0.349813	0.714515	-0.805329	1.263530	0.401498	...	0.489327	0.866
283725	1.642362	-0.385673	-0.211264	0.493422	-0.363259	-0.010186	-0.556582	1.814379	-0.996351	0.493922	...	1.294015	0.853

283726 rows × 28 columns

(5) HANDLING IMBALANCE DATA

- Imbalance data handling is the most important data pre~processing step in Classification Model Building

Checking whether the data is Balanced or Imbalanced

In [85]:
```
new_df['Class'].value_counts()
```

Out[85]:
```
0    283253
1       473
Name: Class, dtype: int64
```

- ['Class'] is our dependent variable
- 0 ~ means No Fraud & 1 ~ means Fraud

In [86]:
```
new_df['Class'].value_counts()/len(new_df)*100
```

Out[86]:
```
0    99.83329
1     0.16671
Name: Class, dtype: float64
```

- Clearly it can be seen the data is an imbalanced data

Treating the Imbalanced data by making it a Balanced one

In [88]:
```
!pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\lenovo\anaconda3\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\lenovo\anaconda3\lib\site-packages (from imblear
n) (0.10.1)
Requirement already satisfied: numpy>=1.17.3 in c:\users\lenovo\anaconda3\lib\site-packages (from imbalanced
-learn->imblearn) (1.23.5)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\lenovo\anaconda3\lib\site-packages (from imb
alanced-learn->imblearn) (2.2.0)
Requirement already satisfied: scipy>=1.3.2 in c:\users\lenovo\anaconda3\lib\site-packages (from imbalanced-
learn->imblearn) (1.10.0)
Requirement already satisfied: joblib>=1.1.1 in c:\users\lenovo\anaconda3\lib\site-packages (from imbalanced
-learn->imblearn) (1.1.1)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\lenovo\anaconda3\lib\site-packages (from imba
lanced-learn->imblearn) (1.2.1)
```

In [87]:
```
import imblearn
```

OVERSAMPLING METHOD

- This method is used to treat Imbalance data & make it a Balanced one

In [88]:
```
from imblearn.over_sampling import RandomOverSampler
ros=RandomOverSampler()
x_bal,y_bal=ros.fit_resample(fs_x,y)
```

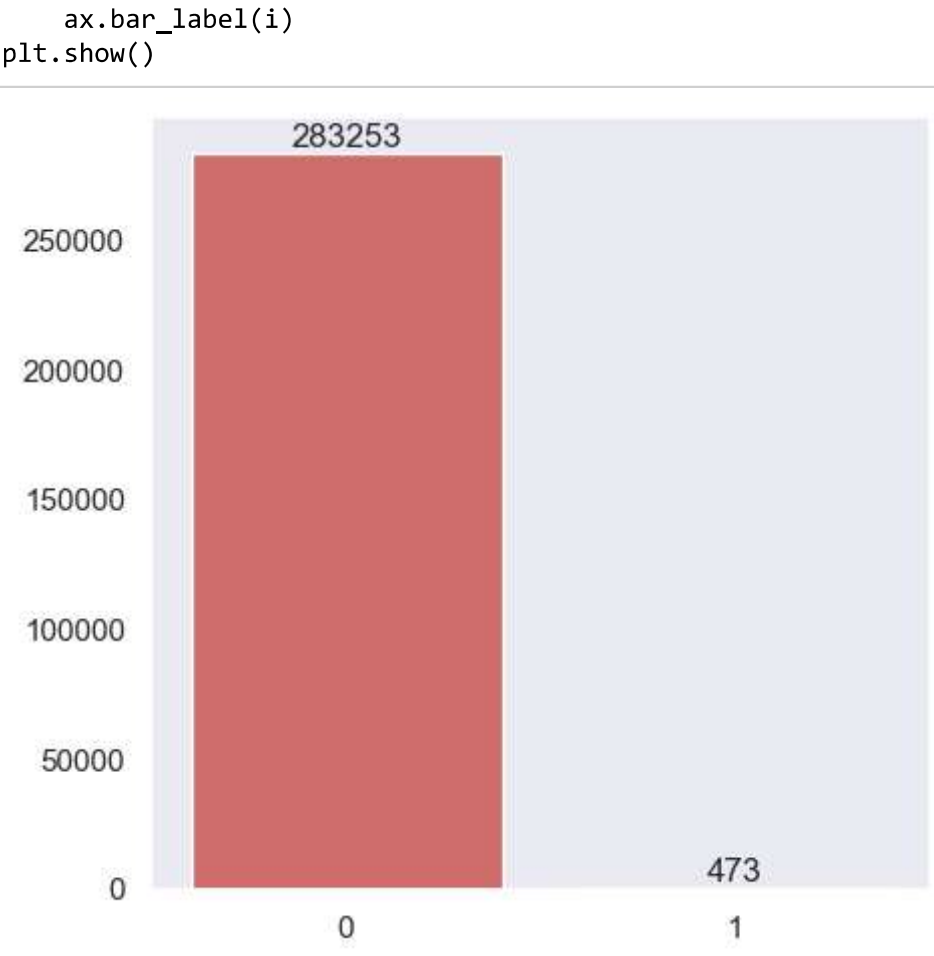
In [89]:
```
print('IMBALANCE DATA:',y.value_counts())
print()
print('BALANCED DATA:',y_bal.value_counts())
```

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

BALANCED DATA: 0    283253
1    283253
Name: Class, dtype: int64
```

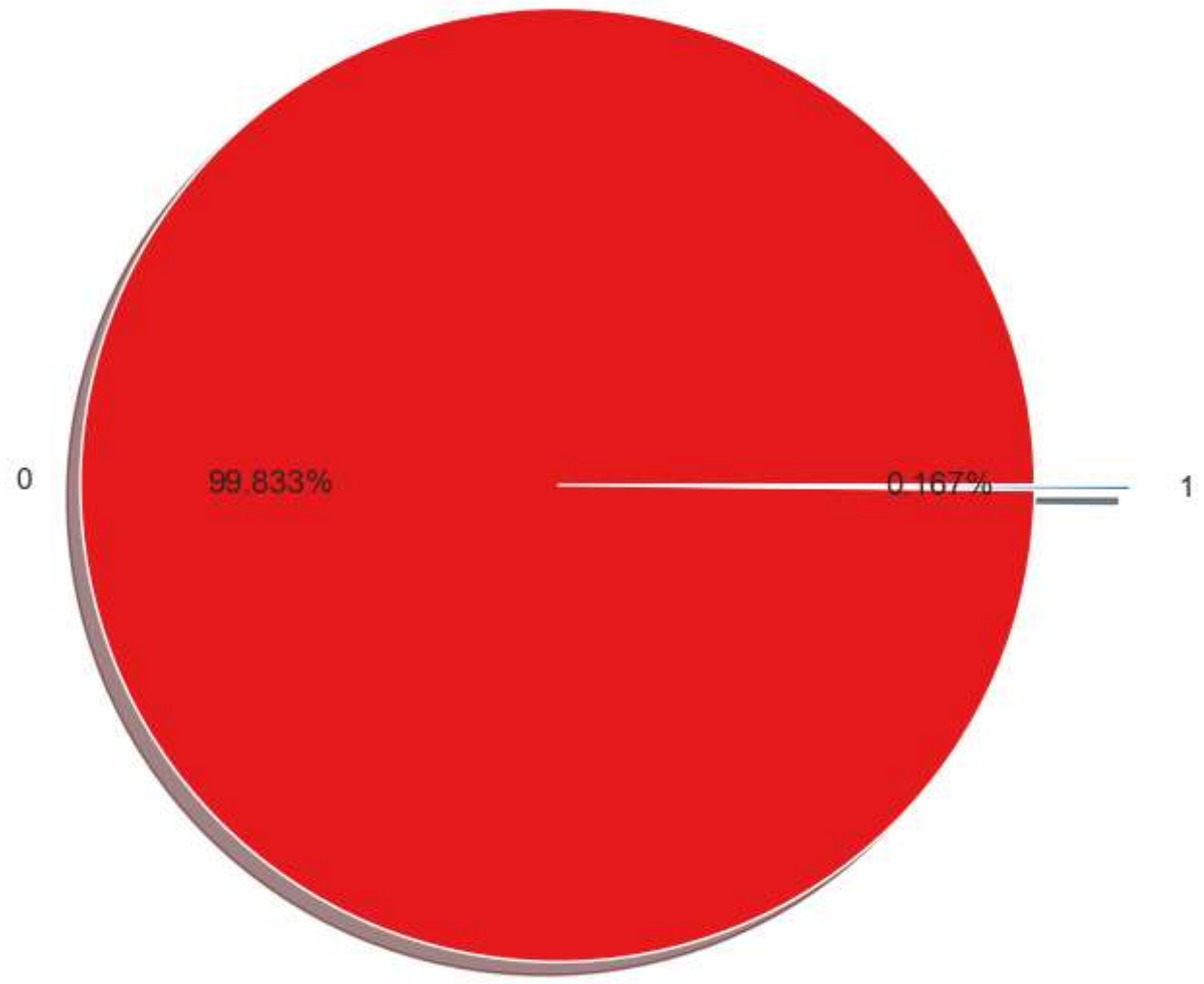
In [90]:
```
print('IMBALANCE DATA %:',y.value_counts()/len(new_df)*100)
print()
print('BALANCED DATA %:',y_bal.value_counts()/len(new_df)*100)
```

```
IMBALANCE DATA %: 0    99.83329
1     0.16671
Name: Class, dtype: float64

BALANCED DATA %: 0    99.83329
1    99.83329
Name: Class, dtype: float64
```

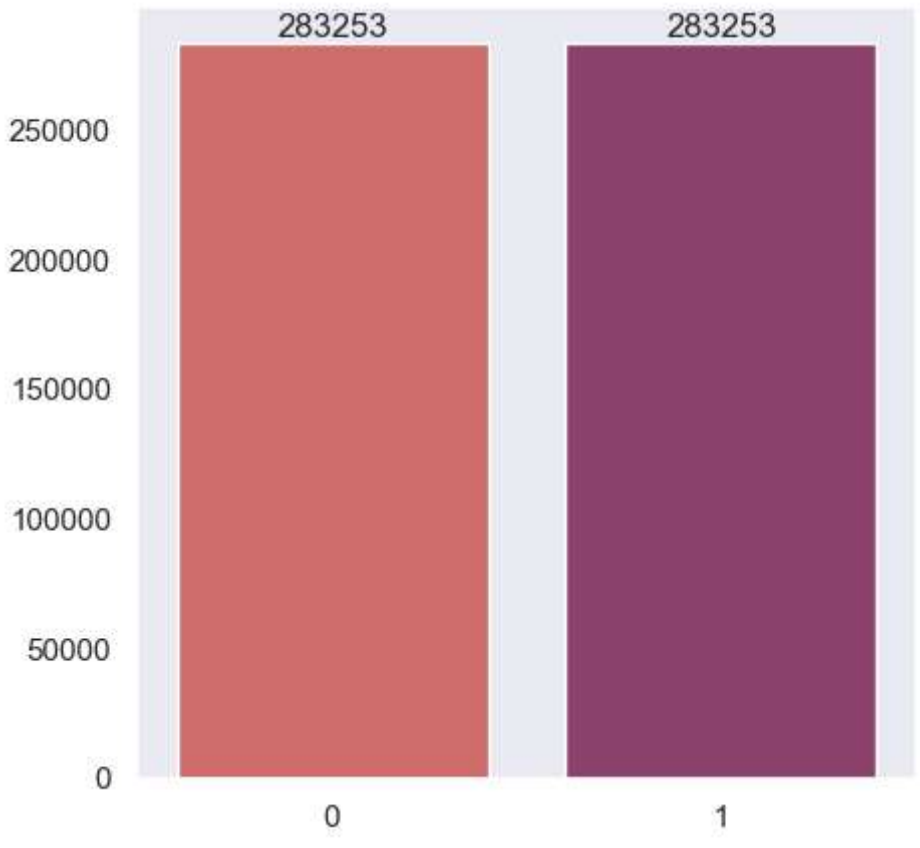
In [91]:
```
### IMBALANCE DATA VISUALIZATION
plt.figure(figsize=(5,5))
ax=sns.barplot(x=y.value_counts().index,y=y.value_counts().values,data=new_df,palette='flare')
for i in ax.containers:
    ax.bar_label(i)
plt.show()
```



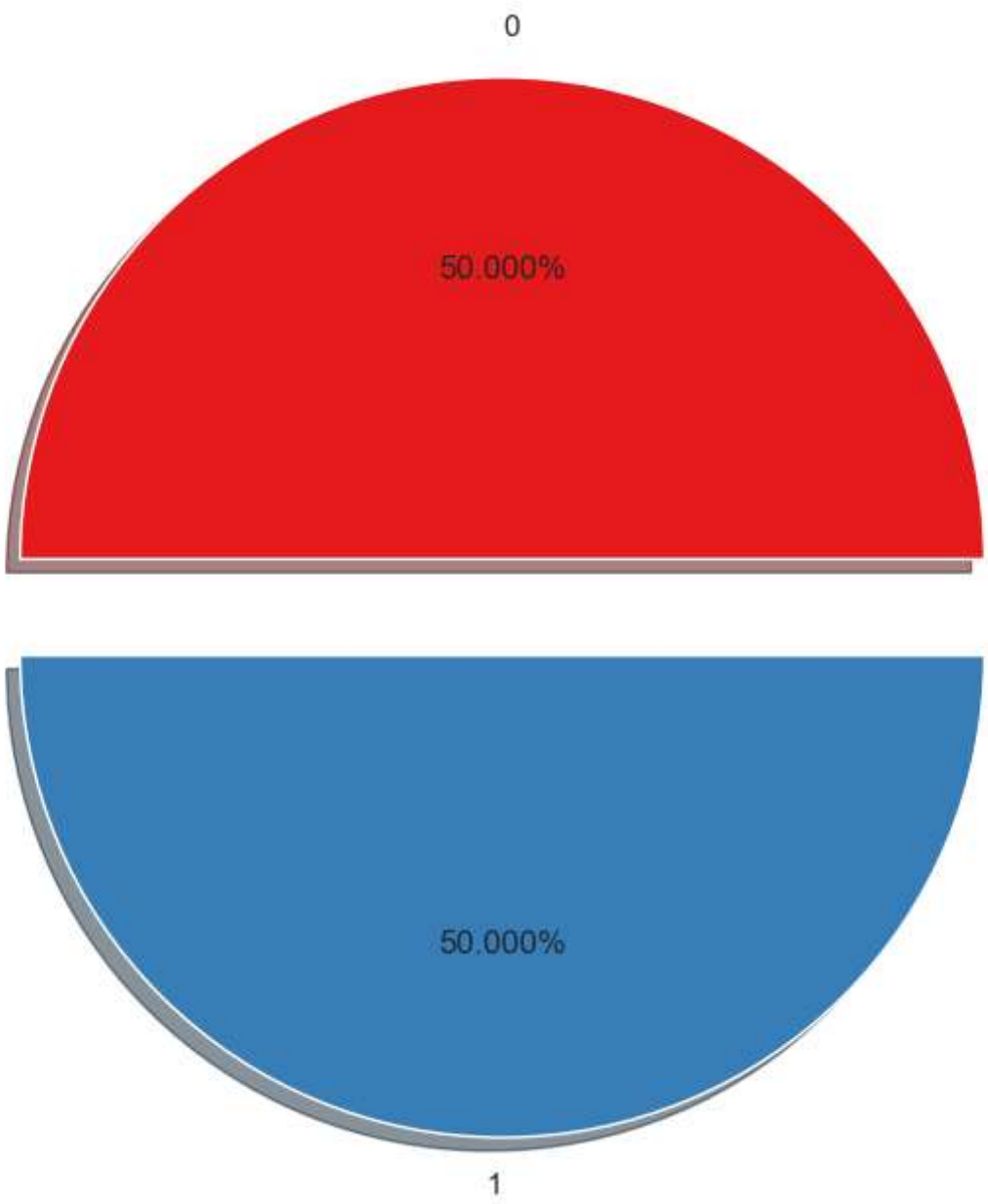
In [92]:
```
### PIE CHART VISUALIZATION
plt.figure(figsize=(8,8),dpi=100)
plt.pie(y.value_counts()*100/len(new_df),
        labels=y.value_counts().index,
        autopct='%.3f%%',shadow=True,explode=[0.1 for i in range(2)],colors=sns.color_palette('Set1'))
plt.show()
```



In [93]:
```
### BALANCED DATA VISUALIZATION
plt.figure(figsize=(5,5))
ax=sns.barplot(x=y_bal.value_counts().index,y=y_bal.value_counts().values,data=new_df,palette='flare')
for i in ax.containers:
    ax.bar_label(i)
plt.show()
```



In [94]:
```
### PIE CHART VISUALIZATION
plt.figure(figsize=(8,8),dpi=100)
plt.pie(y_bal.value_counts()*100/len(new_df),
        labels=y_bal.value_counts().index,
        autopct='%.3f%%',shadow=True,explode=[0.1 for i in range(2)],colors=sns.color_palette('Set1'))
plt.show()
```



- Now the data is a Balanced data
- Now we can predict a good model on this Balanced data

TRAIN ~ TEST SPLIT METHOD

- The train-test split method is a ML method which is used to split your data into a training set & a testing set
- By this we can train our models on training set, & then test their accuracy on the unseen testing set
- Training set ~ 70-80% of data & Testing set ~ 20-30% of data

In [95]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_bal,y_bal,test_size=0.25,random_state=101,stratify=y_bal)
```

In [96]:
```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[96]:
```
((424879, 28), (141627, 28), (424879,), (141627,))
```

BUILDING A BINARY CLASSIFICATION MODEL

Approach-1 (Logistic Regression Method)

In [156]:
```
from sklearn.linear_model import LogisticRegression
LR_Model=LogisticRegression()
LR_Model.fit(x_train,y_train)
```

Out[156]:
```
LogisticRegression
LogisticRegression()
```

In [157]:
```
y_train_pred=LR_Model.predict(x_train)
y_test_pred=LR_Model.predict(x_test)
```

In [158]:
```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,roc_auc_score
```

In [159]:
```
print('TRAIN ACCURACY:',accuracy_score(y_train,y_train_pred))
print()
print('TEST ACCURACY:',accuracy_score(y_test,y_test_pred))
```

```
TRAIN ACCURACY: 0.9480816891397316

TEST ACCURACY: 0.9478771703135701
```

In [160]:
```
print('TRAIN DATA:',confusion_matrix(y_train,y_train_pred))
print()
print('TEST DATA:',confusion_matrix(y_test,y_test_pred))
```

```
TRAIN DATA: [[206123   6317]
 [ 15742 196697]]

TEST DATA: [[68719  2094]
 [ 5288 65526]]
```

In [161]:
```
print('TRAIN DATA:',classification_report(y_train,y_train_pred))
print()
print('TEST DATA:',classification_report(y_test,y_test_pred))
```

```
TRAIN DATA:               precision    recall  f1-score   support

           0       0.93      0.97      0.95    212440
           1       0.97      0.93      0.95    212439

    accuracy                           0.95    424879
   macro avg       0.95      0.95      0.95    424879
weighted avg       0.95      0.95      0.95    424879


TEST DATA:               precision    recall  f1-score   support

           0       0.93      0.97      0.95     70813
           1       0.97      0.93      0.95     70814

    accuracy                           0.95    141627
   macro avg       0.95      0.95      0.95    141627
weighted avg       0.95      0.95      0.95    141627
```

In [162]:
```
print('TRAIN ACCURACY:',roc_auc_score(y_train,y_train_pred))
print()
print('TEST ACCURACY:',roc_auc_score(y_test,y_test_pred))
```

```
TRAIN ACCURACY: 0.9480816369297885

TEST ACCURACY: 0.9478773295475363
```

Approach-2 (Random Forest Method)

In [103]:
```
from sklearn.ensemble import RandomForestClassifier
RFC_Model=RandomForestClassifier()
RFC_Model.fit(x_train,y_train)
```

Out[103]:
```
RandomForestClassifier()
```

In [163]:
```
y_train_pred=RFC_Model.predict(x_train)
y_test_pred=RFC_Model.predict(x_test)
```

In [164]:
```
print('TRAIN ACCURACY:',accuracy_score(y_train,y_train_pred))
print()
print('TEST ACCURACY:',accuracy_score(y_test,y_test_pred))
```

```
TRAIN ACCURACY: 1.0

TEST ACCURACY: 0.9999717567977857
```

In [165]:
```
print('TRAIN DATA:',confusion_matrix(y_train,y_train_pred))
print()
print('TEST DATA:',confusion_matrix(y_test,y_test_pred))
```

```
TRAIN DATA: [[212440      0]
 [     0 212439]]

TEST DATA: [[70809     4]
 [    0 70814]]
```

In [166]:
```
print('TRAIN DATA:',classification_report(y_train,y_train_pred))
print()
print('TEST DATA:',classification_report(y_test,y_test_pred))
```

```
TRAIN DATA:                precision    recall  f1-score   support

           0       1.00      1.00      1.00    212440
           1       1.00      1.00      1.00    212439

    accuracy                           1.00    424879
   macro avg       1.00      1.00      1.00    424879
weighted avg       1.00      1.00      1.00    424879


TEST DATA:               precision    recall  f1-score   support

           0       1.00      1.00      1.00     70813
           1       1.00      1.00      1.00     70814

    accuracy                           1.00    141627
   macro avg       1.00      1.00      1.00    141627
weighted avg       1.00      1.00      1.00    141627
```

In [ ]:
```
### Doing Cross_Val score to treat the overfitting problem above(100% accuracy is not acceptable)
```

In [140]:
```
from sklearn.model_selection import cross_val_score
print('TRAINING SCORE:',cross_val_score(RFC_Model,x_train,y_train,cv=10))
print()
print('TESTING SCORE:',cross_val_score(RFC_Model,x_test,y_test,cv=10))
```

```
TRAINING SCORE: [0.99990586 0.99995293 0.99990586 0.99997646 0.99997646 0.99990586
 0.99995293 0.99997646 0.99995293 1.        ]

TESTING SCORE: [0.99964697 0.99992939 1.         0.99985879 0.99985879 1.
 0.99992939 1.         0.99992939 0.99992939]
```

In [141]:
```
print('AVG TRAINING SCORE:',cross_val_score(RFC_Model,x_train,y_train,cv=10).mean())
print()
print('AVG TESTING SCORE:',cross_val_score(RFC_Model,x_test,y_test,cv=10).mean())
```

```
AVG TRAINING SCORE: 0.9999482206740726

AVG TESTING SCORE: 0.9999011493904224
```

Approach-3 (Gradient Descent Method)

In [108]:
```
from sklearn.linear_model import SGDClassifier
GDC_Model=SGDClassifier()
GDC_Model.fit(x_train,y_train)
```

Out[108]:
```
SGDClassifier

SGDClassifier()
```

In [109]:
```
y_pred_train=GDC_Model.predict(x_train)
y_pred_test=GDC_Model.predict(x_test)
```

In [110]:
```
print('TRAIN ACCURACY:',accuracy_score(y_train,y_pred_train))
print()
print('TEST ACCURACY:',accuracy_score(y_test,y_pred_test))
```

```
TRAIN ACCURACY: 0.9447560364244879

TEST ACCURACY: 0.9447068708650187
```

In [111]:
```
print('TRAIN DATA:',confusion_matrix(y_train,y_pred_train))
print()
print('TEST DATA:',confusion_matrix(y_test,y_pred_test))
```

```
TRAIN DATA: [[208288   4152]
 [ 19320 193119]]

TEST DATA: [[69450  1363]
 [ 6468 64346]]
```

In [112]:
```
print('TRAIN DATA:',classification_report(y_train,y_pred_train))
print()
print('TEST DATA:',classification_report(y_test,y_pred_test))
```

```
TRAIN DATA:               precision    recall  f1-score   support

           0       0.92      0.98      0.95    212440
           1       0.98      0.91      0.94    212439

    accuracy                           0.94    424879
   macro avg       0.95      0.94      0.94    424879
weighted avg       0.95      0.94      0.94    424879


TEST DATA:              precision    recall  f1-score   support

           0       0.91      0.98      0.95     70813
           1       0.98      0.91      0.94     70814

    accuracy                           0.94    141627
   macro avg       0.95      0.94      0.94    141627
weighted avg       0.95      0.94      0.94    141627
```

Approach-4 (Bagging Method)

In [113]:
```
from sklearn.ensemble import BaggingClassifier
BAG_Model=BaggingClassifier()
BAG_Model.fit(x_train,y_train)
```

Out[113]:
```
BaggingClassifier
BaggingClassifier()
```

In [114]:
```
y_pred_train=BAG_Model.predict(x_train)
y_pred_test=BAG_Model.predict(x_test)
```

In [115]:
```
print('TRAIN ACCURACY:',accuracy_score(y_train,y_pred_train))
print()
print('TEST ACCURACY:',accuracy_score(y_test,y_pred_test))
```

```
TRAIN ACCURACY: 0.9999952927774731

TEST ACCURACY: 0.9998587839889287
```

In [116]:
```
print('TRAIN DATA:',confusion_matrix(y_train,y_pred_train))
print()
print('TEST DATA:',confusion_matrix(y_test,y_pred_test))
```

```
TRAIN DATA: [[212438      2]
 [     0 212439]]

TEST DATA: [[70793    20]
 [    0 70814]]
```

In [117]:
```
print('TRAIN DATA:',classification_report(y_train,y_pred_train))
print()
print('TEST DATA:',classification_report(y_test,y_pred_test))
```

```
TRAIN DATA:               precision    recall  f1-score   support

           0       1.00      1.00      1.00    212440
           1       1.00      1.00      1.00    212439

    accuracy                           1.00    424879
   macro avg       1.00      1.00      1.00    424879
weighted avg       1.00      1.00      1.00    424879


TEST DATA:              precision    recall  f1-score   support

           0       1.00      1.00      1.00     70813
           1       1.00      1.00      1.00     70814

    accuracy                           1.00    141627
   macro avg       1.00      1.00      1.00    141627
weighted avg       1.00      1.00      1.00    141627
```

Approach-5 (Decision Tree Method)

In [118]:
```
from sklearn.tree import DecisionTreeClassifier
DTC_Model=DecisionTreeClassifier(criterion='entropy')
DTC_Model.fit(x_train,y_train)
```

Out[118]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy')
```

In [119]:
```
y_pred_train=DTC_Model.predict(x_train)
y_pred_test=DTC_Model.predict(x_test)
```

In [120]:
```
print('TRAIN ACCURACY:',accuracy_score(y_train,y_pred_train))
print()
print('TEST ACCURACY:',accuracy_score(y_test,y_pred_test))
```

```
TRAIN ACCURACY: 1.0

TEST ACCURACY: 0.9998234799861608
```

In [121]:
```
print('TRAIN DATA:',confusion_matrix(y_train,y_pred_train))
print()
print('TEST DATA:',confusion_matrix(y_test,y_pred_test))
```

```
TRAIN DATA: [[212440      0]
 [     0 212439]]

TEST DATA: [[70788    25]
 [    0 70814]]
```

In [122]:
```
print('TRAIN DATA:',classification_report(y_train,y_pred_train))
print()
print('TEST DATA:',classification_report(y_test,y_pred_test))
```

```
TRAIN DATA:               precision    recall  f1-score   support

           0       1.00      1.00      1.00    212440
           1       1.00      1.00      1.00    212439

    accuracy                           1.00    424879
   macro avg       1.00      1.00      1.00    424879
weighted avg       1.00      1.00      1.00    424879


TEST DATA:               precision    recall  f1-score   support

           0       1.00      1.00      1.00     70813
           1       1.00      1.00      1.00     70814

    accuracy                           1.00    141627
   macro avg       1.00      1.00      1.00    141627
weighted avg       1.00      1.00      1.00    141627
```

In [111]:
```
from sklearn.model_selection import cross_val_score
print('TRAINING SCORE:',cross_val_score(DTC_Model,x_train,y_train,cv=10))
print()
print('TESTING SCORE:',cross_val_score(DTC_Model,x_test,y_test,cv=10))
```

```
TRAINING SCORE: [0.9996705  0.99971757 0.99976464 0.99988232 0.99983525 0.9996705
 0.99978818 0.99964696 0.99976464 0.99985878]

TESTING SCORE: [0.99957636 0.9989409  0.99922333 0.9988703  0.99929393 0.99929393
 0.99957636 0.99971755 0.9993645  0.99943511]
```

In [112]:
```
print('AVG. TRAINING SCORE:',cross_val_score(DTC_Model,x_train,y_train,cv=10).mean())
print()
print('AVG. TESTING SCORE:',cross_val_score(DTC_Model,x_test,y_test,cv=10).mean())
```

```
AVG. TRAINING SCORE: 0.999734042276212

AVG. TESTING SCORE: 0.9992939204424672
```

Feature Importance

In [123]:
```
DTC_Model.feature_importances_
```

Out[123]:
```
array([0.01631425, 0.00156493, 0.01264652, 0.01292485, 0.07072274,
       0.00540467, 0.00263345, 0.02283507, 0.02791156, 0.01345844,
       0.0041927 , 0.01527842, 0.05075419, 0.00820028, 0.62974003,
       0.00484612, 0.01434775, 0.01603113, 0.00742561, 0.004533  ,
       0.00896081, 0.00291709, 0.00430616, 0.00944563, 0.00607109,
       0.0173292 , 0.00124532, 0.00795897])
```

In [124]:
```
DTC_MODEL=pd.DataFrame()
```

In [125]:
```
DTC_MODEL['Features']=x.columns
```

In [126]:
```
DTC_MODEL['Feature Importance Value']=DTC_Model.feature_importances_
```

In [127]:
```
DTC_MODEL
```

Out[127]:

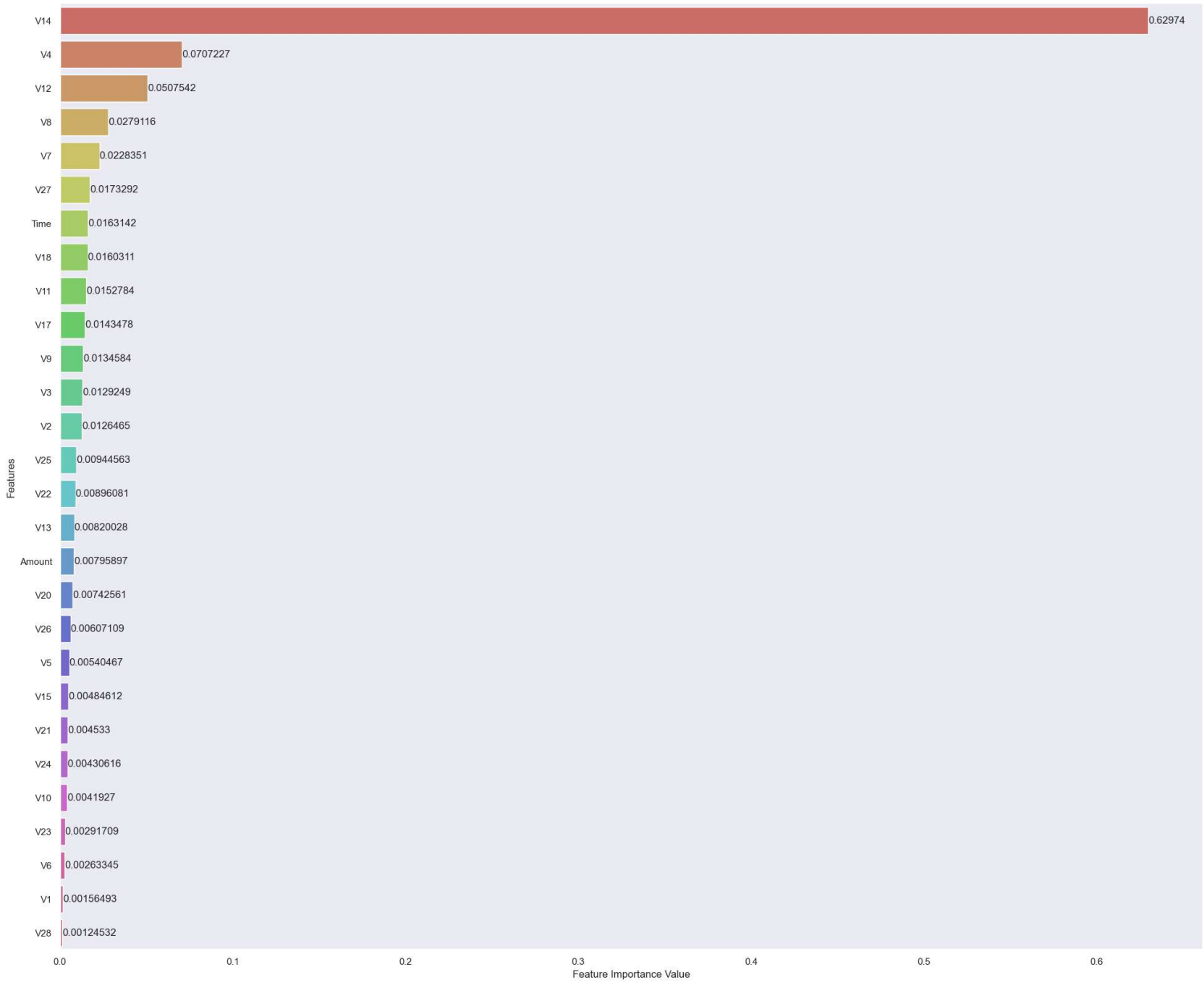
	Features	Feature Importance Value
0	Time	0.016314
1	V1	0.001565
2	V2	0.012647
3	V3	0.012925
4	V4	0.070723
5	V5	0.005405
6	V6	0.002633
7	V7	0.022835
8	V8	0.027912
9	V9	0.013458
10	V10	0.004193
11	V11	0.015278
12	V12	0.050754
13	V13	0.008200
14	V14	0.629740
15	V15	0.004846
16	V17	0.014348
17	V18	0.016031
18	V20	0.007426
19	V21	0.004533
20	V22	0.008961
21	V23	0.002917
22	V24	0.004306
23	V25	0.009446
24	V26	0.006071
25	V27	0.017329
26	V28	0.001245
27	Amount	0.007959

In [128]:
```
DTC_MODEL=DTC_MODEL.sort_values(by='Feature Importance Value',ascending=False)
DTC_MODEL
```

Out[128]:

	Features	Feature Importance Value
14	V14	0.629740
4	V4	0.070723
12	V12	0.050754
8	V8	0.027912
7	V7	0.022835
25	V27	0.017329
0	Time	0.016314
17	V18	0.016031
11	V11	0.015278
16	V17	0.014348
9	V9	0.013458
3	V3	0.012925
2	V2	0.012647
23	V25	0.009446
20	V22	0.008961
13	V13	0.008200
27	Amount	0.007959
18	V20	0.007426
24	V26	0.006071
5	V5	0.005405
15	V15	0.004846
19	V21	0.004533
22	V24	0.004306
10	V10	0.004193
21	V23	0.002917
6	V6	0.002633
1	V1	0.001565
26	V28	0.001245

In [129]:
```
plt.figure(figsize=(24,20))
ax=sns.barplot(y='Features',x='Feature Importance Value',data=DTC_MODEL,palette='hls')
for i in ax.containers:
    ax.bar_label(i)
plt.show()
```



- Here we can see that V14 is the most important feature

POST-PRUNNING METHOD

This method is used to handle the Overfitting problem in Decision Tree Model

In [130]:
```
prunned_dtree=DecisionTreeClassifier(max_depth=2)
prunned_dtree.fit(x_train,y_train)
```

Out[130]:
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=2)
```

In [131]:
```
y_pred_train=prunned_dtree.predict(x_train)
y_pred_test=prunned_dtree.predict(x_test)
```

In [132]:
```
print('TRAIN ACCURACY:',accuracy_score(y_train,y_pred_train))
print()
print('TEST ACCURACY:',accuracy_score(y_test,y_pred_test))
```

```
TRAIN ACCURACY: 0.9184826738906842

TEST ACCURACY: 0.9186595776229108
```

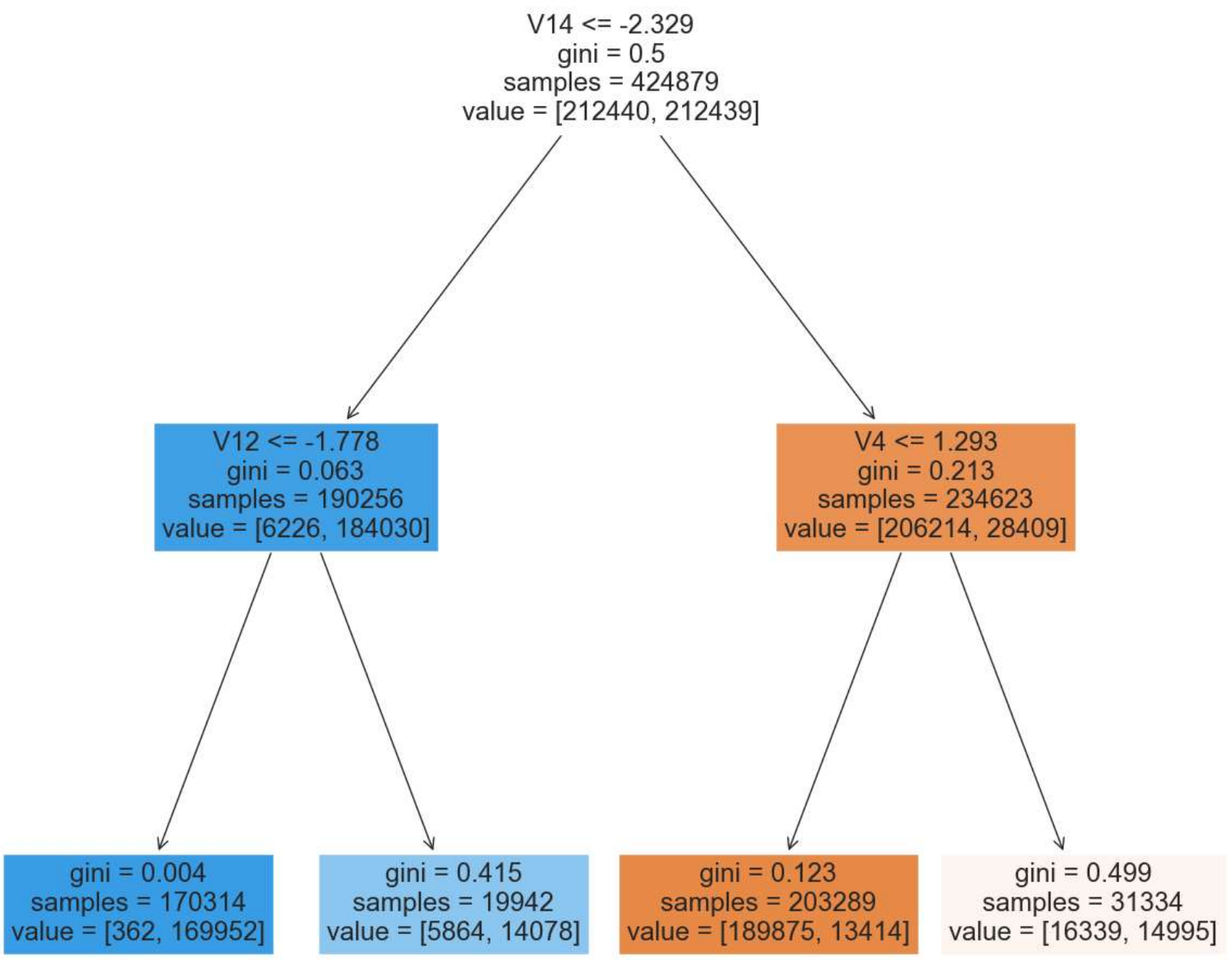
In [133]:
```
print('TRAIN DATA:',confusion_matrix(y_train,y_pred_train))
print()
print('TEST DATA:',confusion_matrix(y_test,y_pred_test))
```

```
TRAIN DATA: [[206214   6226]
 [ 28409 184030]]

TEST DATA: [[68742  2071]
 [ 9449 61365]]
```

In [134]:
```
print('TRAIN DATA:',classification_report(y_train,y_pred_train))
print()
print('TEST DATA:',classification_report(y_test,y_pred_test))
```

```
TRAIN DATA:               precision    recall  f1-score   support

           0       0.88      0.97      0.92    212440
           1       0.97      0.87      0.91    212439

    accuracy                           0.92    424879
   macro avg       0.92      0.92      0.92    424879
weighted avg       0.92      0.92      0.92    424879


TEST DATA:              precision    recall  f1-score   support

           0       0.88      0.97      0.92     70813
           1       0.97      0.87      0.91     70814

    accuracy                           0.92    141627
   macro avg       0.92      0.92      0.92    141627
weighted avg       0.92      0.92      0.92    141627
```

In [135]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(15,15))
plot_tree(prunned_dtree,filled=True,feature_names=x.columns)
plt.show()
```



Approach-6 (K-Nearest Neighbors Method)

In [136]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [137]:
```
KNN_Model=KNeighborsClassifier(n_neighbors=5)
KNN_Model.fit(x_train,y_train)
```

Out[137]:
```
KNeighborsClassifier

KNeighborsClassifier()
```

In [138]:
```
y_pred_train=KNN_Model.predict(x_train)
y_pred_test=KNN_Model.predict(x_test)
```

In [139]:
```
print('TRAIN ACCURACY:',accuracy_score(y_train,y_pred_train))
print()
print('TEST ACCURACY:',accuracy_score(y_test,y_pred_test))
```

```
TRAIN ACCURACY: 0.9996704944231181

TEST ACCURACY: 0.9995269263629111
```

In [140]:
```
print('TRAIN DATA:',confusion_matrix(y_train,y_pred_train))
print()
print('TEST DATA:',confusion_matrix(y_test,y_pred_test))
```

```
TRAIN DATA: [[212300    140]
 [     0 212439]]

TEST DATA: [[70746    67]
 [    0 70814]]
```

In [141]:
```
print('TRAIN DATA:',classification_report(y_train,y_pred_train))
print()
print('TEST DATA:',classification_report(y_test,y_pred_test))
```

```
TRAIN DATA:               precision    recall  f1-score   support

           0       1.00      1.00      1.00    212440
           1       1.00      1.00      1.00    212439

    accuracy                           1.00    424879
   macro avg       1.00      1.00      1.00    424879
weighted avg       1.00      1.00      1.00    424879


TEST DATA:               precision    recall  f1-score   support

           0       1.00      1.00      1.00     70813
           1       1.00      1.00      1.00     70814

    accuracy                           1.00    141627
   macro avg       1.00      1.00      1.00    141627
weighted avg       1.00      1.00      1.00    141627
```

In [143]:
```
from sklearn.model_selection import cross_val_score
print('TRAINING SCORE:',cross_val_score(KNN_Model,x_train,y_train,cv=10))
print()
print('TESTING SCORE:',cross_val_score(KNN_Model,x_test,y_test,cv=10))
```

```
TRAINING SCORE: [0.99955281 0.99943513 0.99959989 0.99948221 0.99962342 0.99943513
 0.99955281 0.99957635 0.99931745 0.99943512]

TESTING SCORE: [0.99830544 0.99858787 0.99908212 0.99908212 0.99872908 0.99816423
 0.99851726 0.99865838 0.99908205 0.99901144]
```

In [144]:
```
print('TRAINING SCORE:',cross_val_score(KNN_Model,x_train,y_train,cv=10).mean())
print()
print('TESTING SCORE:',cross_val_score(KNN_Model,x_test,y_test,cv=10).mean())
```

```
TRAINING SCORE: 0.999501034257017

TESTING SCORE: 0.9987219992365404
```

CONCLUSION

- Overally, we have successfully predict a good classification model that involves a systematic & iterative approach including data understanding, preprocessing, model selection, hyperparameter tuning, and thorough evaluation.
- We saw above that by using all the algorithms the train & test accuracy of our model is more then >90%(Good Accuracy).
- The success of the model reflects not only technical skills but also a deep understanding of the data and the problem at hand, which can drive meaningful and positive impacts in various applications and industries.