In this notebook, I will predict fraud transactions from a given dataset. Since the data is imbalanced, using standard metrics like accuracy to evaluate the classification algorithm would be misleading. Instead, I'll focus on metrics like Sensitivity (true positive rate) and Specificity (true negative rate). These two metrics are interdependent, so the goal is to find an optimal trade-off between them. The ideal balance depends on the specific application, and in the case of fraud detection, I'll prioritize high Sensitivity—I want to detect fraudulent transactions with high probability

IMPORTING LIBRARIES:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pylab import rcParams
import warnings
warnings.filterwarnings('ignore')
```

READING DATASET :

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

```
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.2778
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.6386
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.7716
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.0052
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.7982

5 rows × 31 columns

NULL VALUES:

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

File "<ipython-input-52-1ca740324b01>", line 2 data = data.dropna(subset=['C ...

You

Please explain the error:

```
  File "<ipython-input-52-1ca740324b01>", line 2
    data = data.dropna(subset=['Class'])
    ^
IndentationError: unexpected indent
```

Gemini

Hypothesis

The code has an unexpected indent on line 2. Python is sensitive to indentation and this could cause issues with how the code is interpreted.

Suggested Changes

```
# Drop rows with NaN in 'Class' column from the original DataFrame
data = data.dropna(subset=['Class'])

# Redo the steps to create X2, y2, and the train-test split
X2 = data.drop(['Class'], axis=1)
```

Use code with caution

▶ Sources

Rate this answer

	0
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	1
V21	1
V22	1
V23	1
V24	1
V25	1
V26	1
V27	1
V28	1
Amount	1
Class	1

Thus there are no null values in the dataset.

INFORMATION

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11959 entries, 0 to 11958
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Time    11959 non-null  int64  
 1   V1      11959 non-null  float64
 2   V2      11959 non-null  float64
 3   V3      11959 non-null  float64
 4   V4      11959 non-null  float64
 5   V5      11959 non-null  float64
 6   V6      11959 non-null  float64
 7   V7      11959 non-null  float64
 8   V8      11959 non-null  float64
 9   V9      11959 non-null  float64
 10  V10     11959 non-null  float64
 11  V11     11959 non-null  float64
 12  V12     11959 non-null  float64
 13  V13     11959 non-null  float64
 14  V14     11959 non-null  float64
 15  V15     11959 non-null  float64
 16  V16     11959 non-null  float64
 17  V17     11959 non-null  float64
 18  V18     11959 non-null  float64
 19  V19     11959 non-null  float64
 20  V20     11958 non-null  float64
 21  V21     11958 non-null  float64
 22  V22     11958 non-null  float64
 23  V23     11958 non-null  float64
 24  V24     11958 non-null  float64
 25  V25     11958 non-null  float64
 26  V26     11958 non-null  float64
 27  V27     11958 non-null  float64
 28  V28     11958 non-null  float64
 29  Amount  11958 non-null  float64
 30  Class   11958 non-null  float64
dtypes: float64(30), int64(1)
memory usage: 2.8 MB
```

DESCRIPTIVE STATISTICS

```
data.describe().T.head()
```

	count	mean	std	min	25%	50%	75%	max
Time	11959.0	8009.996822	6204.332248	0.000000	2542.000000	6662.000000	12382.000000	20642.000000
V1	11959.0	-0.216230	1.583914	-27.670569	-0.978944	-0.340742	1.161273	1.960497
V2	11959.0	0.277097	1.308884	-34.607649	-0.261503	0.256346	0.883626	9.092123
V3	11959.0	0.889505	1.331824	-22.804686	0.417186	0.951223	1.613678	4.101716
V4	11959.0	0.282606	1.478162	-4.657545	-0.622456	0.213029	1.159141	11.927512

```
data.shape
```

```
(11959, 31)
```

Thus there are 284807 rows and 31 columns.

```
data.columns
```

```
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')
```

FRAUD CASES AND GENUINE CASES

```
fraud_cases=len(data[data['Class']==1])
```

```
print(' Number of Fraud Cases:',fraud_cases)
```

```
 Number of Fraud Cases: 52
```

```
non_fraud_cases=len(data[data['Class']==0])
```

```
print('Number of Non Fraud Cases:',non_fraud_cases)
```

```
Number of Non Fraud Cases: 11906
```

```
fraud=data[data['Class']==1]
```

```
genuine=data[data['Class']==0]
```

```
fraud.Amount.describe()
```

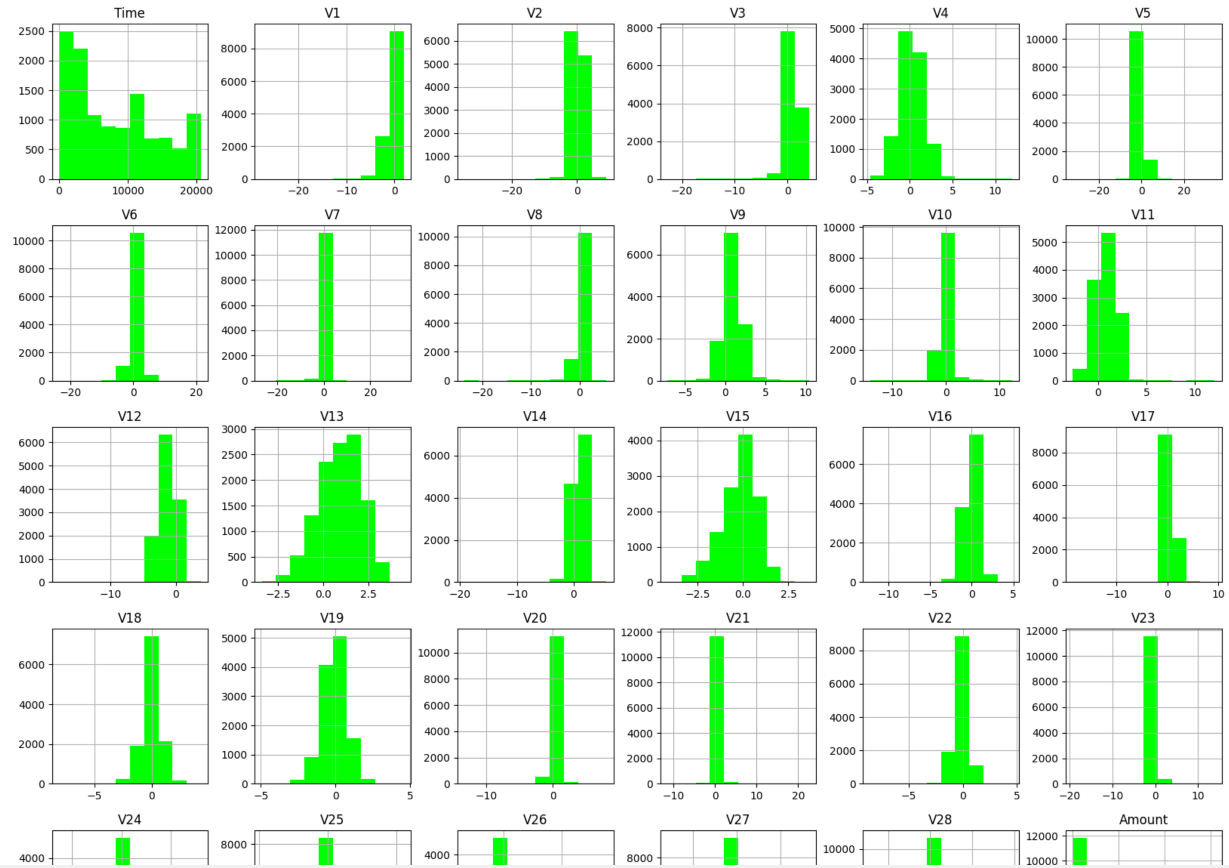
	Amount
count	52.000000
mean	97.724808
std	321.188775
min	0.000000
25%	1.000000
50%	1.000000
75%	1.772500
max	1809.680000

```
genuine.Amount.describe()
```

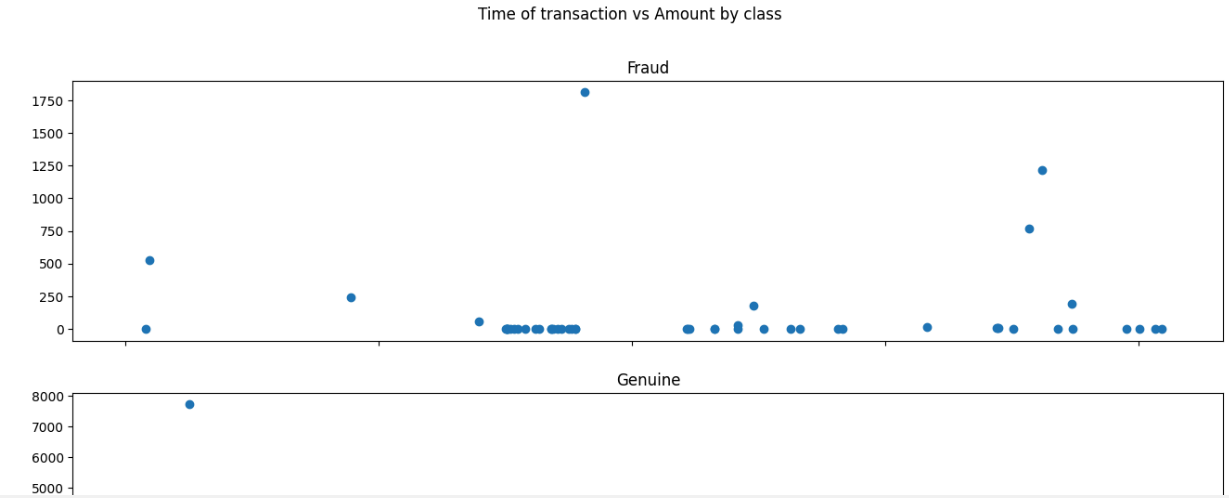
	Amount
count	11906.000000
mean	62.198127
std	177.379105
min	0.000000
25%	5.292500
50%	15.950000
75%	50.000000
max	7712.430000

EDA

```
data.hist(figsize=(20,20),color='lime')
plt.show()
```



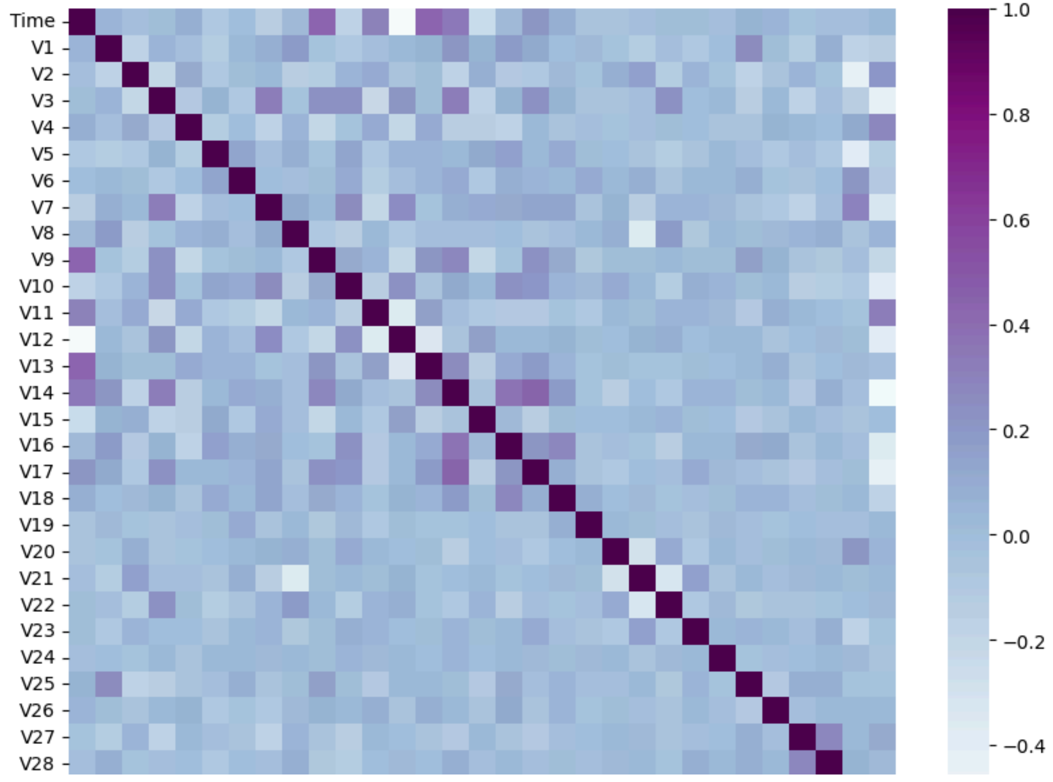

```
rcParams['figure.figsize'] = 16, 8
f,(ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Time of transaction vs Amount by class')
ax1.scatter(fraud.Time, fraud.Amount)
ax1.set_title('Fraud')
ax2.scatter(genuine.Time, genuine.Amount)
ax2.set_title('Genuine')
plt.xlabel('Time (in Seconds)')
plt.ylabel('Amount')
plt.show()
```



CORRELATION

```
plt.figure(figsize=(10,8))
corr=data.corr()
sns.heatmap(corr,cmap='BuPu')
```

```
<Axes: >
```



Let us build our models:

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

Model 1:

```
X=data.drop(['Class'],axis=1)
```

```
y=data['Class']
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random_state=123)
```

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

```
rfc=RandomForestClassifier()
```

```
model=rfc.fit(X_train,y_train)
```

```
X_train.dropna(inplace=True)
# Ensure y_train is also updated to match the dropped rows
y_train = y_train[X_train.index]
model=rfc.fit(X_train,y_train)
```

```
prediction=model.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test,prediction)
```

```
0.9994425863991081
```

Model 2:

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

```
X1=data.drop(['Class'],axis=1)
```

```
y1=data['Class']
```

```
X1_train,X1_test,y1_train,y1_test=train_test_split(X1,y1,test_size=0.3,random_state=123)
```

```
lr=LogisticRegression()
```

```
model2=lr.fit(X1_train,y1_train)
```

```
X1_train.dropna(inplace=True)
y1_train = y1_train[X1_train.index] # Make sure y1_train matches X1_train after dropping rows
model2 = lr.fit(X1_train, y1_train)
```

```
prediction2=model2.predict(X1_test)
```

```
accuracy_score(y1_test,prediction2)
```

```
0.9983277591973244
```

Model 3:

```
from sklearn.tree import DecisionTreeRegressor
```

```
X2=data.drop(['Class'],axis=1)
```

```
y2=data['Class']

dt=DecisionTreeRegressor()

X2_train,X2_test,y2_train,y2_test=train_test_split(X2,y2,test_size=0.3,random_state=123)

model3=dt.fit(X2_train,y2_train)

# ipython-input-49-9e40dd3bc30f
```

Enter a prompt here

0 / 400