Name: Suhani Shankarrao Bhokare.

Project Name: Credit Card Fraud Detection.

Introduction:

Data Understanding

- The Data has 32 features from V1-V28 which are unknown for confidentiality, TIme, Amount and Class
- The input features are V1-V28, Time and Amount
- The target variable is Class
- The Data does not have any missing values as evident from the below mentioned code, thus need not be handled
- The Data consists of all numerical features, and only the Target Variable Class is a categorical feature.
    - Class 0: Legitimate Transaction
    - Class 1: Fraud Transaction

```
: #importing libraries
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import os
import pandas as pd
```

```
]: nRowsRead = 1000
dt= pd.read_csv('creditcard.csv', delimiter=',', nrows = nRowsRead)
dt.dataframeName = 'creditcard.csv'
nRow, nCol = dt.shape
print(f'There are {nRow} rows and {nCol} columns')
```

```
There are 1000 rows and 31 columns
```

```
: dt.head(5)
```

:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V2
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	0.066928	0.12853
1	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	-0.339846	0.16717
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.32764
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.64737
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20601

5 rows × 31 columns

```
dt.tail(5)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V
995	751	-0.654892	0.608319	1.585102	-3.009429	0.037593	-1.954023	1.335977	-0.612858	0.690254	...	-0.078527	-0.064194	-0.107350	0.961776	-0.067
996	752	-2.101171	-0.227365	1.624668	-0.291123	1.902446	-1.483921	-0.275117	0.085964	-0.563098	...	-0.313782	-0.804784	-0.474101	0.008102	0.259
997	753	-1.248163	0.315246	3.708935	0.687280	-0.873071	1.091287	0.297707	-0.633135	1.102317	...	-0.824013	0.057907	-0.282351	0.630774	0.283
998	755	1.374134	-1.767210	-0.433352	-2.229552	0.331135	3.924775	-2.049947	1.001403	-1.183310	...	-0.252942	-0.461534	-0.030890	0.997119	0.384
999	755	-2.497436	1.402769	0.184840	-2.504117	-0.111803	-0.902909	0.110183	-3.655788	2.231761	...	2.777155	-0.664909	0.594689	0.330380	0.064

5 rows × 31 columns

Data Preparation

- he Data does not have any missing values and hence, need not be handled.
- The Data has only Target Variable Class as the categorical variable.
- Remaining Features are numerical and need to be only standardized for comparison after balancing the dataset
- The mean of the amount of money in transactions is 66.43
- The standard deviation of amount of money in transactions is 187.27
- The time is distributed throughout the data equitably and hence, serves as an independent feature
- It is best to not remove or drop any data or features in this case and try to tune the model assuming them as independent features initially

```
# Print the shape of the data
dt.shape
```

```
(1000, 31)
```

```
dt.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	10
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	...	1000.000000	
mean	371.478000	-0.185541	0.224434	0.873905	0.241431	-0.033144	0.152717	0.096350	-0.061222	-0.001297	...	0.004469	
std	223.289659	1.319761	1.154144	1.026305	1.257776	1.084735	1.235014	0.840424	0.883285	0.891584	...	0.630712	
min	0.000000	-6.093248	-12.114213	-5.694973	-4.657545	-6.631951	-3.498447	-4.925568	-7.494658	-2.980624	...	-4.134608	
25%	170.500000	-0.935897	-0.186676	0.307137	-0.492527	-0.562147	-0.632045	-0.334229	-0.174967	-0.493081	...	-0.219778	
50%	368.000000	-0.377203	0.284608	0.884642	0.326839	-0.148192	-0.122938	0.089204	0.030920	-0.063950	...	-0.080389	
75%	559.250000	1.126162	0.867528	1.529554	1.067740	0.440822	0.476314	0.561682	0.252395	0.439243	...	0.083731	
max	755.000000	1.685314	5.267376	4.017561	4.861129	7.672544	5.122103	4.808426	3.877662	5.459274	...	5.273420	

8 rows × 31 columns

```
: dt.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Time    1000 non-null   int64  
 1   V1      1000 non-null   float64
 2   V2      1000 non-null   float64
 3   V3      1000 non-null   float64
 4   V4      1000 non-null   float64
 5   V5      1000 non-null   float64
 6   V6      1000 non-null   float64
 7   V7      1000 non-null   float64
 8   V8      1000 non-null   float64
 9   V9      1000 non-null   float64
 10  V10     1000 non-null   float64
 11  V11     1000 non-null   float64
 12  V12     1000 non-null   float64
 13  V13     1000 non-null   float64
 14  V14     1000 non-null   float64
 15  V15     1000 non-null   float64
 16  V16     1000 non-null   float64
 17  V17     1000 non-null   float64
 18  V18     1000 non-null   float64
 19  V19     1000 non-null   float64
 20  V20     1000 non-null   float64
 21  V21     1000 non-null   float64
 22  V22     1000 non-null   float64
 23  V23     1000 non-null   float64
 24  V24     1000 non-null   float64
 25  V25     1000 non-null   float64
 26  V26     1000 non-null   float64
 27  V27     1000 non-null   float64
 28  V28     1000 non-null   float64
 29  Amount  1000 non-null   float64
 30  Class   1000 non-null   int64  
dtypes: float64(29), int64(2)
memory usage: 242.3 KB
```

```
#to check nulls values in the dataset
dt.isnull().sum()
```

```
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
```

```
dt.duplicated().sum()
```

```
7
```

```
]: duplicate_rows_data = dt[dt.duplicated()]
   print("Number of duplicated rows: ", duplicate_rows_data.shape)
```

```
Number of duplicated rows:  (7, 31)
```

```
dt.nunique()
```

```
Time      540
V1        992
V2        992
V3        992
V4        992
V5        992
V6        992
V7        992
V8        992
V9        992
V10       992
V11       992
V12       992
V13       992
V14       992
V15       992
V16       992
V17       992
V18       992
V19       992
V20       992
V21       992
V22       992
V23       992
V24       992
V25       992
V26       992
V27       992
V28       992
Amount    639
Class       2
dtype: int64
```

The dataset contains transactions made by credit cards.

To determine fraud case we have perform operation on class column

```
# Determine number of fraud cases in dataset
fraud = dt[dt['Class'] == 1]
valid = dt[dt['Class'] == 0]
outlierFraction = len(fraud)/float(len(valid))
print(outlierFraction)
print('Total Fraud Cases: {}'.format(len(dt[dt['Class'] == 1])))
print('Valid Transactions: {}'.format(len(dt[dt['Class'] == 0])))
```

```
0.002004008016032064
Total Fraud Cases: 2
Valid Transactions: 998
```

Here we have describe amount of each fraud transaction

```
print("Amount details of the fraudulent transaction")
fraud.Amount.describe()
```

```
Amount details of the fraudulent transaction

count      2.000000
mean     264.500000
std      374.059487
min        0.000000
25%      132.250000
50%      264.500000
75%      396.750000
max      529.000000
Name: Amount, dtype: float64
```

Below we have describe valid transaction made by credit cards

```
: print("details of valid transaction")
  valid.Amount.describe()
```

```
  details of valid transaction

: count     998.000000
  mean       66.033347
  std       186.874480
  min         0.000000
  25%         5.452500
  50%        16.185000
  75%        54.997500
  max      3828.040000
  Name: Amount, dtype: float64
```

we have to find maximum transaction of each credut cards

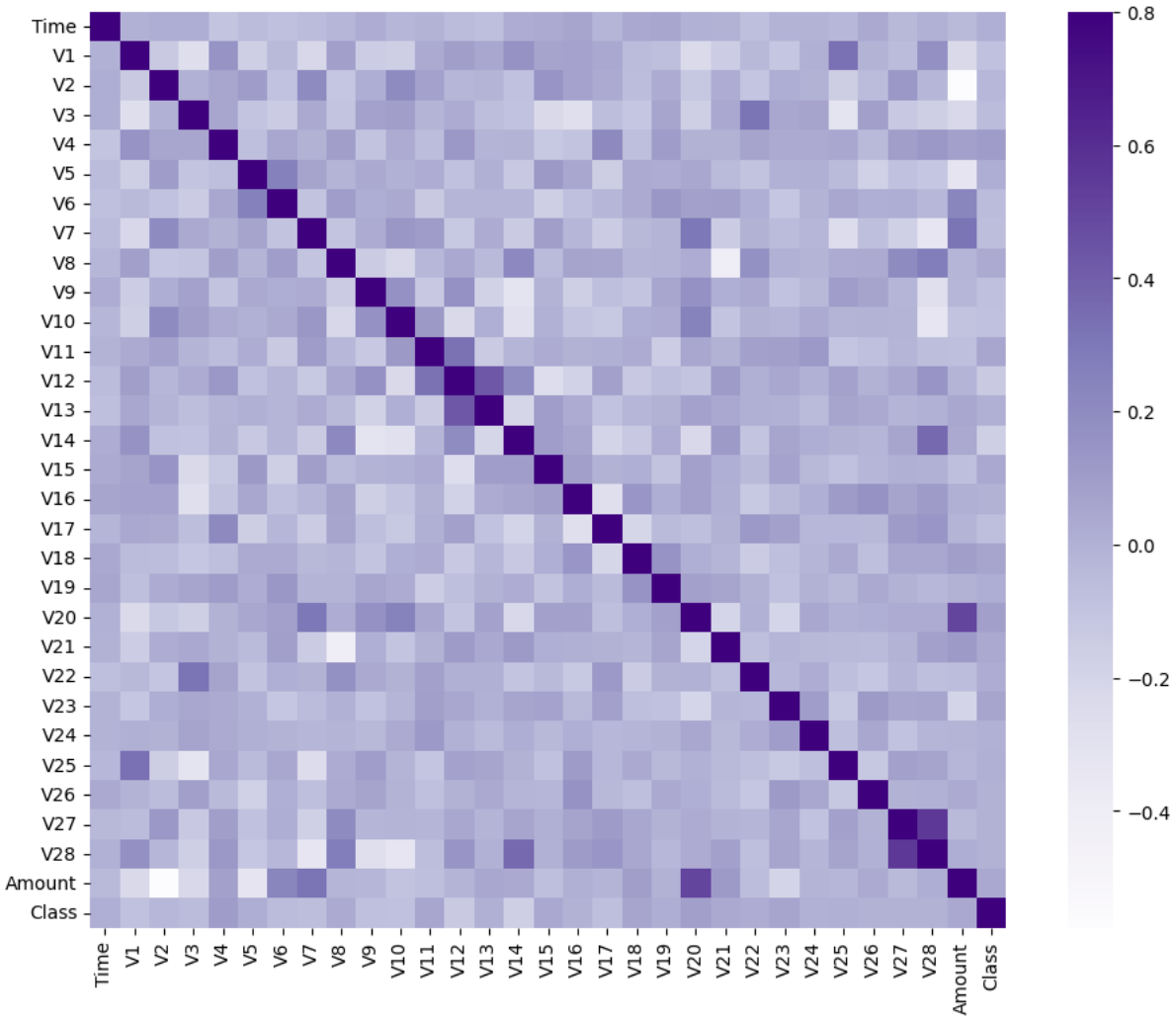
```
: dt.apply(np.max)
```

```
: Time        755.000000
  V1            1.685314
  V2            5.267376
  V3            4.017561
  V4            4.861129
  V5            7.672544
  V6            5.122103
  V7            4.808426
  V8            3.877662
  V9            5.459274
  V10           8.821756
  V11           3.202033
  V12           2.152055
  V13           2.142710
  V14           1.977296
  V15           2.703685
  V16           1.898748
  V17           3.986289
  V18           2.689762
  V19           2.862706
  V20           7.744222
  V21           5.273420
  V22           1.574750
  V23           3.150413
  V24           1.215279
  V25           1.206820
  V26           3.087444
  V27           2.490503
  V28           1.575380
  Amount     3828.040000
  Class         1.000000
  dtype: float64
```

Modelling

It defines nothing but correlation between transaction and amount

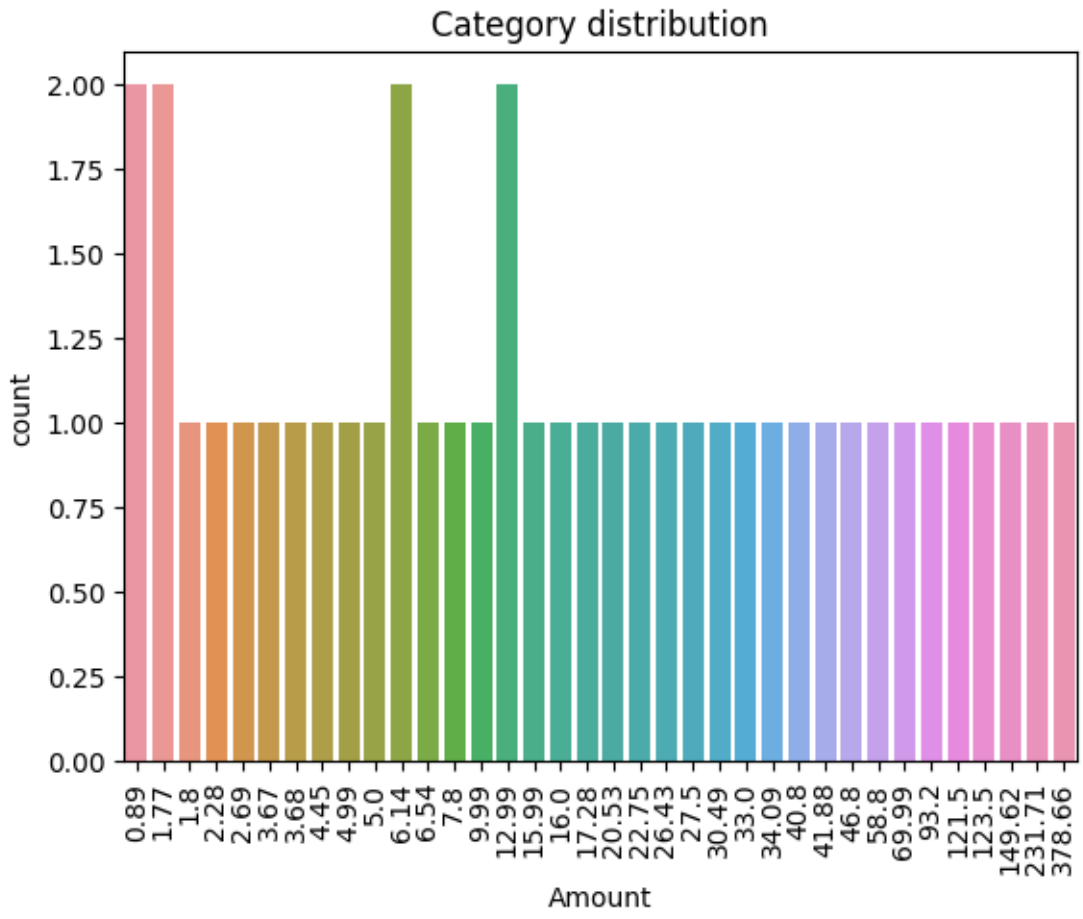
```
:  #Correlation matrix
   #The correlation matrix graphically gives us an idea of
   #how features correlate with each other and can help us predict
   #what are the features that are most relevant for the prediction.
   corrmat = dt.corr()
   fig = plt.figure(figsize = (12, 9))
   sns.heatmap(corrmat, vmax = .8, square = True,cmap='Purples')
   plt.show()
```



Each amount is categorized according transaction made by credit cards

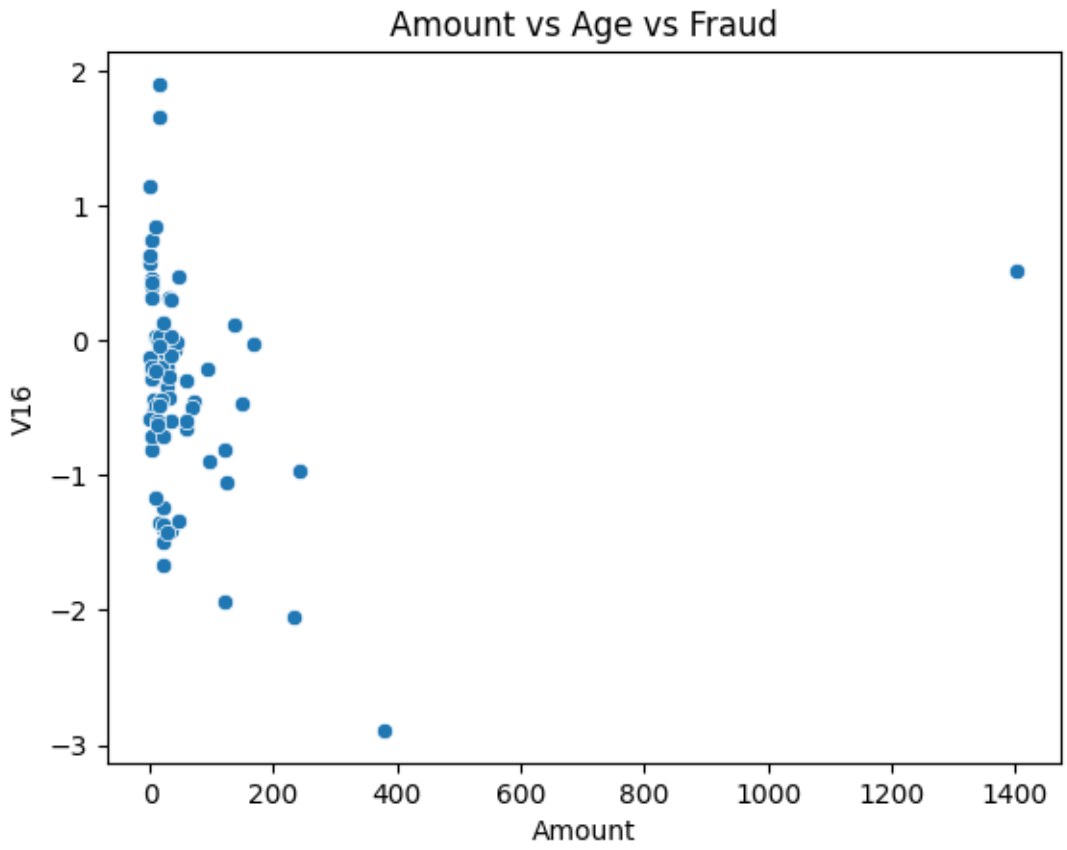
```
: #with help of countplot we can compare amount with fraud
  sns.countplot(x='Amount',data=dt.head(40))
  plt.title('Category distribution')
  plt.xticks(rotation=90)

  plt.show()
```



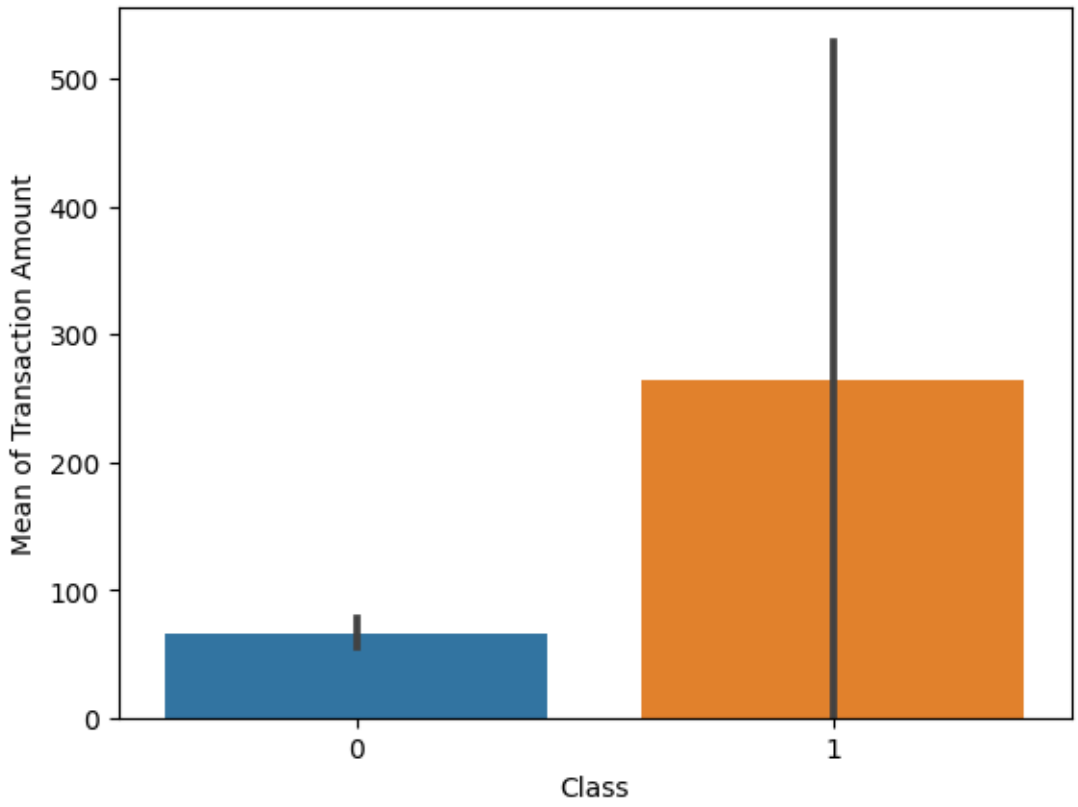
Here the transaction made by credit card v16 is specified according to the amount

```
#using scatterplot we have specify fraud of V16 column
sns.scatterplot(data=dt.head(80), x='Amount',y ='V16')
plt.title('Amount vs Age vs Fraud')
plt.show()
```



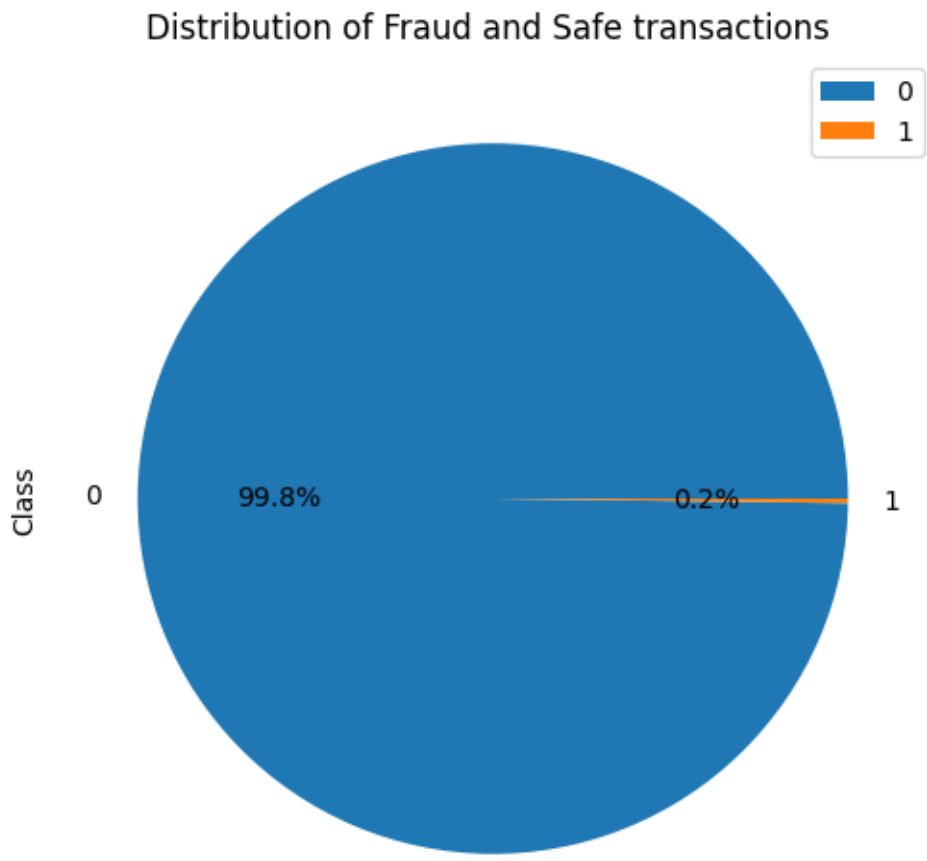
These defines mean of transaction amount with respect to class

```
sns.barplot(data=dt, x='Class', y='Amount')
plt.ylabel('Mean of Transaction Amount')
```



Distribution between fraud and valid

```
m = dt['Class'].value_counts()
m.plot(kind ='pie', autopct = '%1.1f%%', figsize = (12,6))
plt.legend()
plt.title('Distribution of Fraud and Safe transactions ')
```



Conclusion:

- The fraud transactions are equitable distributed throughout time and there is no clear relationship of time with commiting of fraud.
- The number of fraud transactions are very few comparted to legitimate transactions and it has to be balanced in order for a fair comparison to prevent the model from overfitting.