pandas is to read the dataset

numpy for numbering in array or dealing with numbers in array form

mathplotlib for drawing graph and data visualization

seaborn uses matplotlib underneath to plot graphs .it is also use for visual random distribution

pylab is a module that provides a matlib like namespace by importing functions from the modules numpy and matplotlib(rcParams) where by rc stands for "Run Command"

warnings alert the user of some condition in a program,where that condition (normally) doesn't warrant raising an exception and terminating the program.

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA, TruncatedSVD
import matplotlib.patches as mpatches
import time

# Classifier Libraries
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import collections


# Other Libraries
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from imblearn.pipeline import make_pipeline as imbalanced_make_pipeline
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss
from imblearn.metrics import classification_report_imbalanced
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, accuracy_score, classificat
from collections import Counter
from sklearn.model_selection import KFold, StratifiedKFold
import warnings
warnings.filterwarnings("ignore")
```

Reading dataset

In [2]:
```
dataset=pd.read_csv('creditcard.csv')
```

In [3]:
```
dataset.head(10)
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23
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	-0.110474
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	0.101288
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	0.909412
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	-0.190321
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	-0.137458
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794

10 rows × 31 columns

Null values

Null values

In [4]:
```
dataset.isnull().sum()
```

Out[4]:
```
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
```

Thus there are no null values in the dataset

Information

In [5]:
```
dataset.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
```

descriptive statistics

In [6]:
```
dataset.describe().T.head()
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
Time	284807.0	9.481386e+04	47488.145955	0.000000	54201.500000	84692.000000	139320.500000	172792.000000
V1	284807.0	1.168375e-15	1.958696	-56.407510	-0.920373	0.018109	1.315642	2.454930
V2	284807.0	3.416908e-16	1.651309	-72.715728	-0.598550	0.065486	0.803724	22.057729
V3	284807.0	-1.379537e-15	1.516255	-48.325589	-0.890365	0.179846	1.027196	9.382558
V4	284807.0	2.074095e-15	1.415869	-5.683171	-0.848640	-0.019847	0.743341	16.875344

shape

In [7]:
```
dataset.shape
```

Out[7]:
```
(284807, 31)
```

Thus there are 284807 rows and 31 columns

In [8]:
```
dataset.columns
```

Out[8]:
```
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

In [9]:
```
fraud_cases=len(dataset[dataset['Class']==1])
```

In [10]:
```
print('number of fraud cases',fraud_cases)
```

```
number of fraud cases 492
```

In [11]:
```
non_fraud_cases=len(dataset[dataset['Class']==0])
```

In [12]:
```
print('number of non fraud cases',non_fraud_cases)
```

```
number of non fraud cases 284315
```

In [13]:
```
fraud=dataset[dataset['Class']==1]
```

In [14]:
```
genuine=dataset[dataset['Class']==0]
```

fraud amount describe

In [15]:
```
fraud.Amount.describe()
```

Out[15]:
```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

genuine amount describe

In [16]:
```
genuine.Amount.describe()
```

Out[16]:
```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

recorder the columns amount,time then the reset

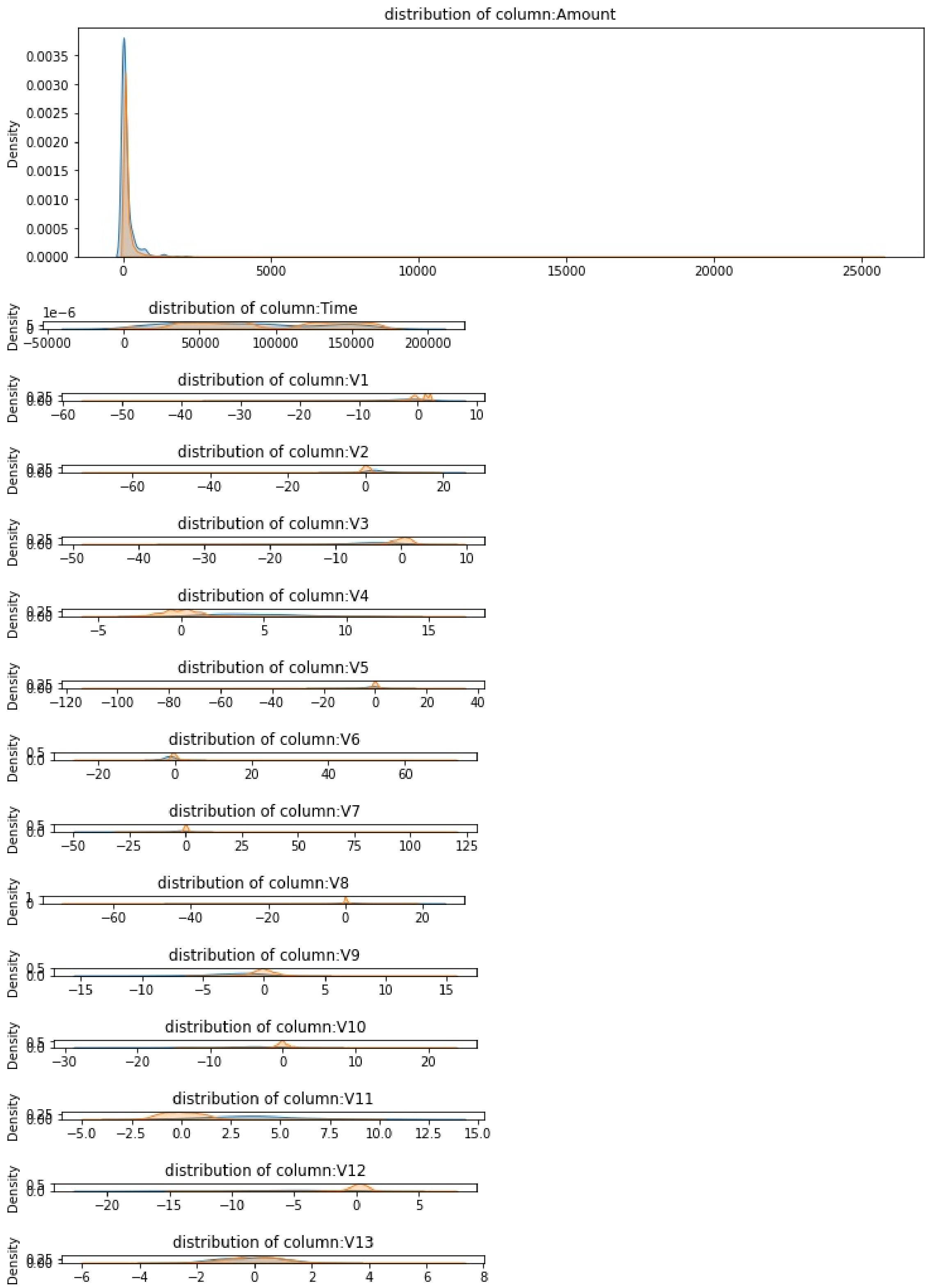
In [17]:
```
data_plot=dataset.copy()
```

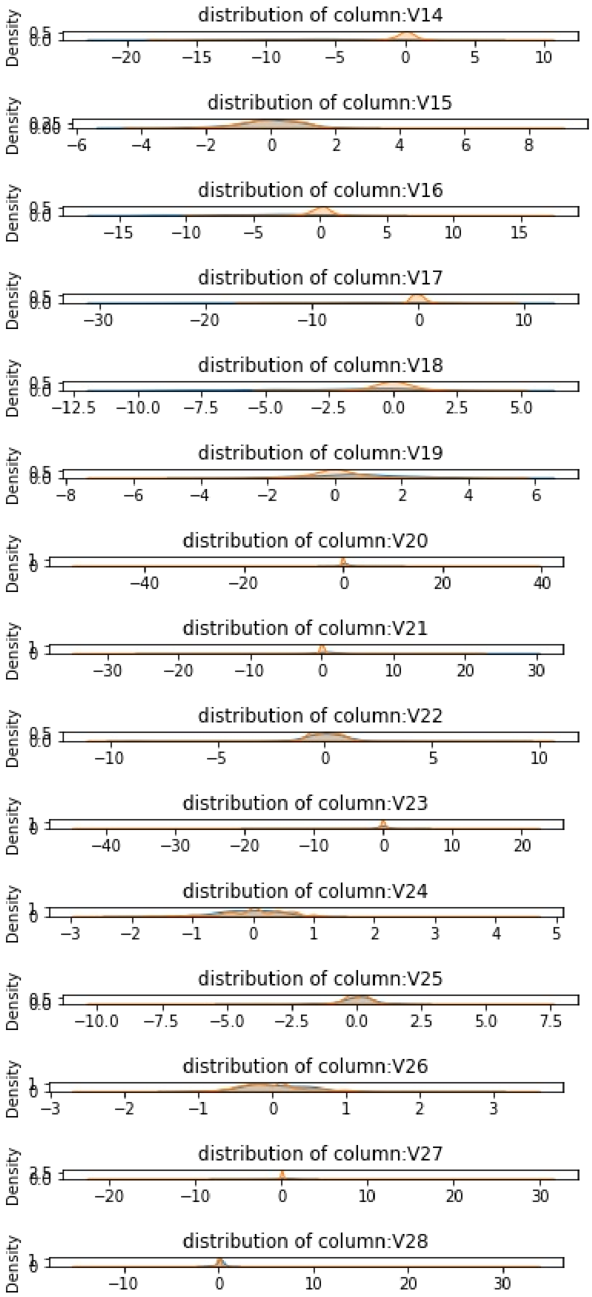
```
amount=data_plot['Amount']
data_plot.drop(labels=['Amount'],axis=1,inplace=True)
data_plot.insert(0,'Amount',amount)
```

In [18]:
```
from matplotlib import gridspec
```

plot the distrubutions of the features

In [19]:
```
columns=data_plot.iloc[:,0:30].columns
plt.figure(figsize=(12,30*4))
grids=gridspec.GridSpec(30,1)
for grid, index in enumerate (data_plot[columns]):
    AX=plt.subplot(grids[grid])
    sns.distplot(data_plot[index][data_plot.Class==1],hist=False,kde_kws={'shade':True},bins=50)
    sns.distplot(data_plot[index][data_plot.Class==0],hist=False,kde_kws={'shade':True},bins=50)
    AX.set_xlabel('')
    AX.set_title('distribution of column:'+str(index))
    plt.show()
```

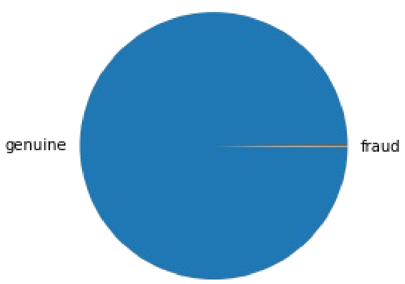




pie chart

In [20]:
```
yinka=dataset.copy()
yinka['']=np.where(yinka['Class']==1,'fraud','genuine')
yinka[''].value_counts().plot(kind='pie')
```

Out[20]: <AxesSubplot:>



mean for fraudlent transaction

In [21]:
```
print('average fraudulent transaction:'+str(dataset[dataset['Class']==1].mean()))
```

```
average fraudulent transaction:Time      80746.806911
V1            -4.771948
V2             3.623778
V3            -7.033281
V4             4.542029
V5            -3.151225
V6            -1.397737
V7            -5.568731
V8             0.570636
V9            -2.581123
V10           -5.676883
V11            3.800173
V12           -6.259393
V13           -0.109334
V14           -6.971723
V15           -0.092929
V16           -4.139946
V17           -6.665836
V18           -2.246308
V19            0.680659
V20            0.372319
V21            0.713588
V22            0.014049
V23           -0.040308
V24           -0.105130
V25            0.041449
V26            0.051648
V27            0.170575
V28            0.075667
Amount       122.211321
Class          1.000000
dtype: float64
```

mean valid transaction

In [22]:
```
print('average valid transaction:'+str(dataset[dataset['Class']==0].mean()))
```

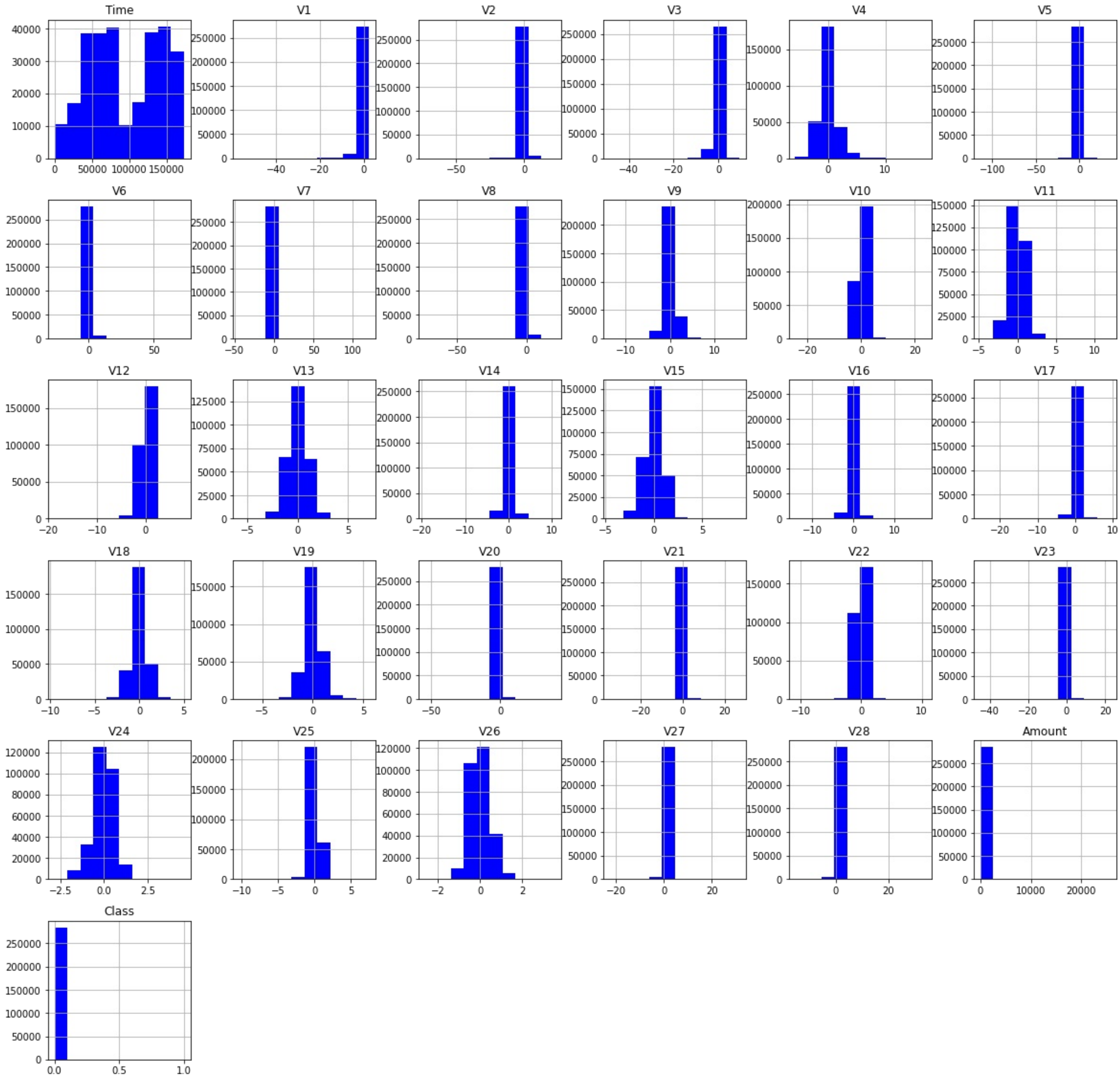
```
average valid transaction:Time      94838.202258
V1             0.008258
V2            -0.006271
V3             0.012171
V4            -0.007860
V5             0.005453
V6             0.002419
V7             0.009637
V8            -0.000987
V9             0.004467
V10            0.009824
V11           -0.006576
V12            0.010832
V13            0.000189
V14            0.012064
V15            0.000161
V16            0.007164
V17            0.011535
V18            0.003887
V19           -0.001178
V20           -0.000644
V21           -0.001235
V22           -0.000024
V23            0.000070
V24            0.000182
V25           -0.000072
V26           -0.000089
V27           -0.000295
V28           -0.000131
Amount        88.291022
Class          0.000000
dtype: float64
```

In [23]:
```
print(dataset['Amount'].describe())
```

```
count    284807.000000
mean         88.349619
std         250.120109
min           0.000000
25%           5.600000
50%          22.000000
75%          77.165000
max       25691.160000
Name: Amount, dtype: float64
```

EDA

In [24]:
```
dataset.hist(figsize=(20,20),color='blue')
plt.show()
```



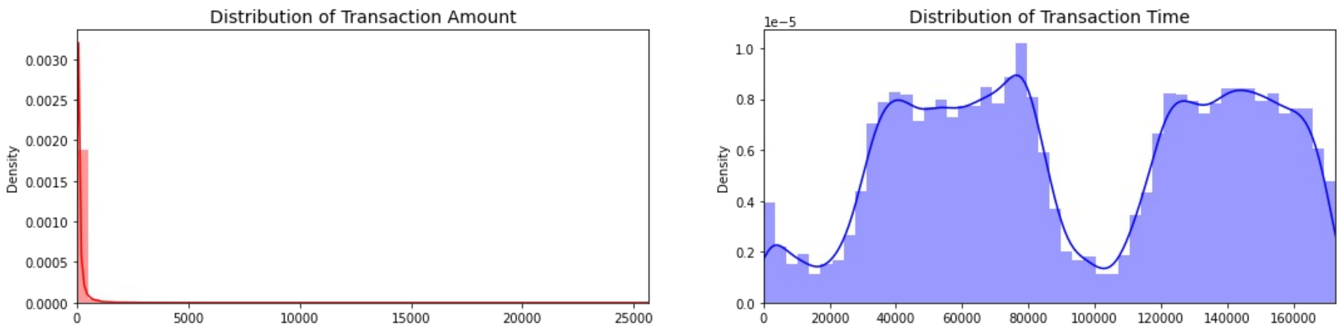
In [25]:
```
fig, ax = plt.subplots(1, 2, figsize=(18,4))

amount_val = dataset['Amount'].values
time_val = dataset['Time'].values

sns.distplot(amount_val, ax=ax[0], color='r')
ax[0].set_title('Distribution of Transaction Amount', fontsize=14)
ax[0].set_xlim([min(amount_val), max(amount_val)])

sns.distplot(time_val, ax=ax[1], color='b')
ax[1].set_title('Distribution of Transaction Time', fontsize=14)
ax[1].set_xlim([min(time_val), max(time_val)])

plt.show()
```



In [26]:
```
# Since our classes are highly skewed we should make them equivalent in order to have a normal distribution of

# Lets shuffle the data before creating the subsamples

dataset = dataset.sample(frac=1)
```

```
# amount of fraud classes 492 rows.
fraud_dataset = dataset.loc[dataset['Class'] == 1]
non_fraud_dataset = dataset.loc[dataset['Class'] == 0][:492]

normal_distributed_dataset = pd.concat([fraud_dataset, non_fraud_dataset])

# Shuffle dataframe rows
new_dataset = normal_distributed_dataset.sample(frac=1, random_state=42)

new_dataset.head(3)
```

Out[26]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
261878	160221.0	-0.056193	0.917390	0.057261	0.088053	0.434082	-0.835394	0.920280	-0.138401	0.163511	...	-0.132416	-0.192573	-0.0
197586	132086.0	-0.361428	1.133472	-2.971360	-0.283073	0.371452	-0.574680	4.031513	-0.934398	-0.768255	...	0.110815	0.563861	-0.4
274426	166002.0	-0.744504	1.697067	-1.539083	-0.822112	0.699408	-0.145664	0.226306	0.899748	-0.521053	...	-0.279424	-0.845548	0.1

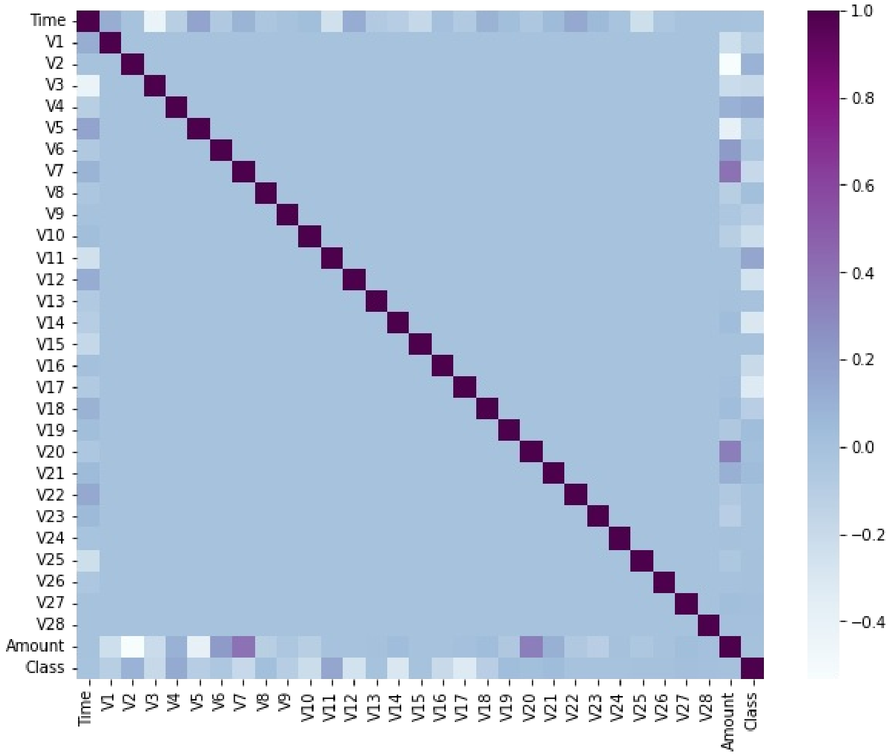
3 rows × 31 columns

correlation

In [27]:

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

Out[27]: <AxesSubplot:>



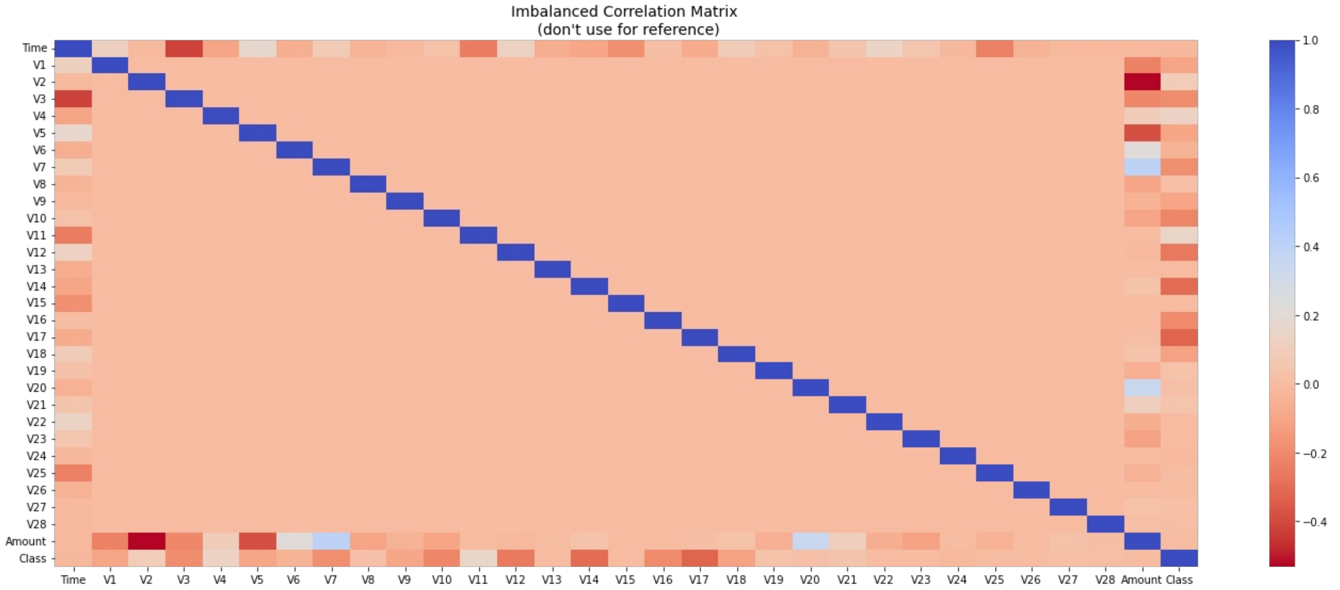
In [28]:

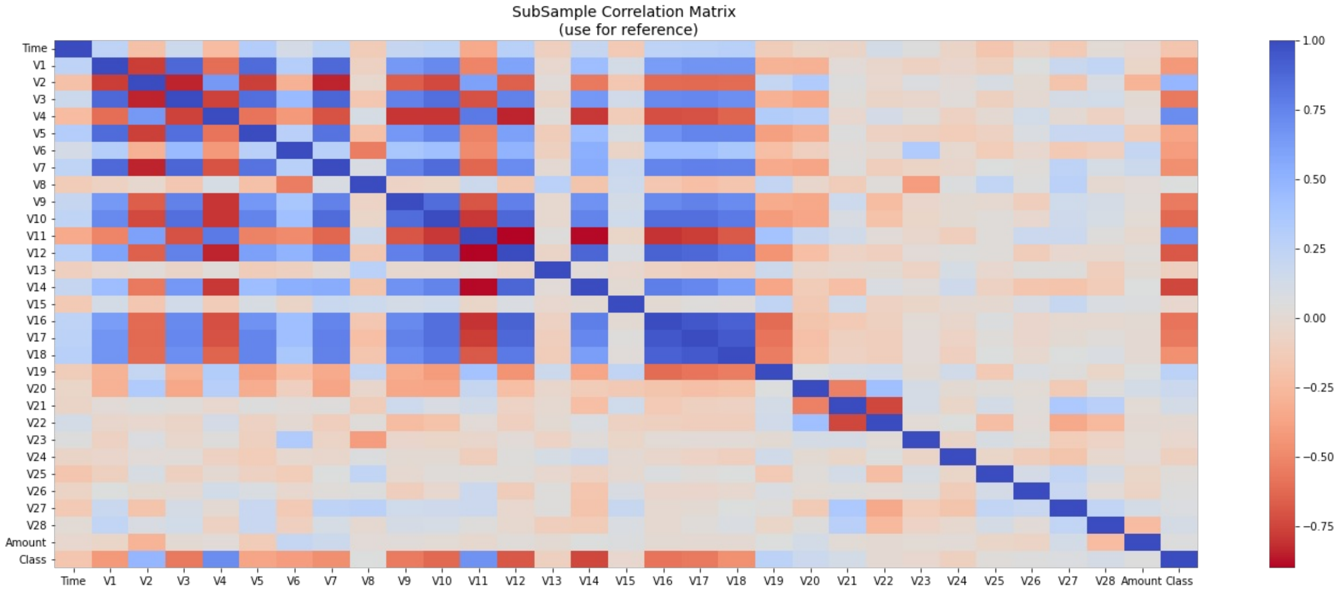
```
# Make sure we use the subsample in our correlation

f, (ax1, ax2) = plt.subplots(2, 1, figsize=(24,20))

# Entire DataFrame - df
corr = dataset.corr()
sns.heatmap(corr, cmap='coolwarm_r', annot_kws={'size':20}, ax=ax1)
ax1.set_title("Imbalanced Correlation Matrix \n (don't use for reference)", fontsize=14)

# Undersampled - new_df
sub_sample_corr = new_dataset.corr()
sns.heatmap(sub_sample_corr, cmap='coolwarm_r', annot_kws={'size':20}, ax=ax2)
ax2.set_title('SubSample Correlation Matrix \n (use for reference)', fontsize=14)
plt.show()
```





building a model

In [29]:

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

print('No Frauds', round(dataset['Class'].value_counts()[0]/len(dataset) * 100,2), '% of the dataset')
print('Frauds', round(dataset['Class'].value_counts()[1]/len(dataset) * 100,2), '% of the dataset')

X = dataset.drop('Class', axis=1)
y = dataset['Class']

sss = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)

for train_index, test_index in sss.split(X, y):
    print("Train:", train_index, "Test:", test_index)
    original_Xtrain, original_Xtest = X.iloc[train_index], X.iloc[test_index]
    original_ytrain, original_ytest = y.iloc[train_index], y.iloc[test_index]

# We already have X_train and y_train for undersample data thats why I am using original to distinguish and to 
# original_Xtrain, original_Xtest, original_ytrain, original_ytest = train_test_split(X, y, test_size=0.2, rand

# Check the Distribution of the labels


# Turn into an array
original_Xtrain = original_Xtrain.values
original_Xtest = original_Xtest.values
original_ytrain = original_ytrain.values
original_ytest = original_ytest.values

# See if both the train and test label distribution are similarly distributed
train_unique_label, train_counts_label = np.unique(original_ytrain, return_counts=True)
test_unique_label, test_counts_label = np.unique(original_ytest, return_counts=True)
print('-' * 100)

print('Label Distributions: \n')
```

```
print(train_counts_label/ len(original_ytrain))
print(test_counts_label/ len(original_ytest))
```

```
No Frauds 99.83 % of the dataset
Frauds 0.17 % of the dataset
Train: [ 56952  56953  56954 ... 284804 284805 284806] Test: [     0      1      2 ... 64100 64591 64901]
Train: [     0      1      2 ... 284804 284805 284806] Test: [ 56952  56953  56954 ... 118404 118623 118998]
Train: [     0      1      2 ... 284804 284805 284806] Test: [113915 113916 113917 ... 170898 170899 170900]
Train: [     0      1      2 ... 284804 284805 284806] Test: [165111 165439 165487 ... 227851 227852 227853]
Train: [     0      1      2 ... 227851 227852 227853] Test: [222132 222395 224212 ... 284804 284805 284806]
----------------------------------------------------------------------------------------------------
Label Distributions:

[0.99827076 0.00172924]
[0.99827952 0.00172048]
```

developing model

here in developing algorithms or model for credit card fruad detection using machine and deep learning approach i'll be making use of 3 alogorithms, which are listed below

1. multiple linear regression
2. logistic regression
3. naive bayes classifier

In [30]:
```
from sklearn.model_selection import train_test_split
```

In [31]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random_state=123)
```

In [32]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [33]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score as roc
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
import tensorflow as tf
from tensorflow import keras
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
```

1st model (multiple linear regression)

In [34]:
```
## let see the prediction
## seperating fraud from geninue
## fraud case
fraud_case=dataset.loc[dataset['Class']==1]
## geninue case
geninue_case=dataset.loc[dataset['Class']==0]
```

In [35]:
```
## storing linearRegression in a variable
lr=LinearRegression()
```

In [36]:
```
## fit the dataset to the train values
lr.fit(X_train,y_train)
```

Out[36]:
```
LinearRegression()
```

In [37]:
```
## predict the model on the trained values
predlr=lr.predict(X_train)
```

In [38]:
```
## predict the model on the train values and check resultfor i in range(0, len(predTest)):
for i in range(0, len(predlr)):
    if(predlr[i]>=0.5):
        predlr[i]=1
    else:
        predlr[i]=0
print(classification_report(y_train,predlr))
print('ROC AUC Score for linear regression is:',roc(y_train,predlr))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    199037
           1       0.83      0.43      0.57       327

    accuracy                           1.00    199364
   macro avg       0.91      0.72      0.78    199364
weighted avg       1.00      1.00      1.00    199364

ROC AUC Score for linear regression is: 0.7155234794987406
```

In [39]:
```
lr.fit(X_test,y_test)
```
Out[39]:
```
LinearRegression()
```
In [40]:
```
predlr2=lr.predict(X_test)
```
In [41]:
```
## checking test records using 0.5 threshold
predTest = lr.predict(X_test)
for i in range(0, len(predTest)):
    if(predTest[i]>=0.5):
        predTest[i]=1
    else:
        predTest[i]=0
print(classification_report(y_test, predTest))
print('ROC AUC Score for linear regression: ',roc(y_test, predTest))
```
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85278
           1       0.86      0.40      0.55       165

    accuracy                           1.00     85443
   macro avg       0.93      0.70      0.77     85443
weighted avg       1.00      1.00      1.00     85443

ROC AUC Score for linear regression:  0.6999355050540584
```

2nd model (logistic regression)

In [42]:
```
## creating regression object and scale the dataset
classifier=LogisticRegression(random_state=20)
classifier.fit(X_train,y_train)
```
Out[42]:
```
LogisticRegression(random_state=20)
```
In [43]:
```
## predict the model on the train values and check results

predTrain2=classifier.predict(X_train)
print(classification_report(y_train, predTrain2))
print('ROC AUC Score for logistic regression: ',roc(y_train, predTrain2))
```
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    199037
           1       0.71      0.61      0.66       327

    accuracy                           1.00    199364
   macro avg       0.86      0.81      0.83    199364
weighted avg       1.00      1.00      1.00    199364

ROC AUC Score for logistic regression:  0.8056094298942374
```
In [44]:
```
##predict test values and check results
predTest2=classifier.predict(X_test)
print(classification_report(y_test, predTest2))
print('ROC AUC Score for logistic regression: ',roc(y_test, predTest2))
```
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85278
           1       0.67      0.58      0.62       165

    accuracy                           1.00     85443
   macro avg       0.83      0.79      0.81     85443
weighted avg       1.00      1.00      1.00     85443

ROC AUC Score for logistic regression:  0.7906276584177098
```

3rd model (Gaussian naive_bayes classifier)

In [45]:
```
## creating regression object and scale the dataset
## fit the dataset to the train value

classifier2=GaussianNB()
classifier2.fit(X_train,y_train)
```
Out[45]:
```
GaussianNB()
```
In [46]:
```
## predict the model on the train values and check results

predG=classifier2.predict(X_train)
print(classification_report(y_train,predG))
print('ROC AUC Score GaussianNB',roc(y_train,predG))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00    199037
           1       0.14      0.65      0.23       327

    accuracy                           0.99    199364
   macro avg       0.57      0.82      0.61    199364
weighted avg       1.00      0.99      1.00    199364

ROC AUC Score GaussianNB 0.8193416360940006
```

In [47]:
```
## predict the model on the test values and check results
predGa=classifier2.predict(X_test)
print(classification_report(y_test,predGa))
print('ROC AUC Score GaussianNB',roc(y_test,predGa))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     85278
           1       0.16      0.62      0.25       165

    accuracy                           0.99     85443
   macro avg       0.58      0.81      0.62     85443
weighted avg       1.00      0.99      0.99     85443

ROC AUC Score GaussianNB 0.8059013408552563
```

checking model with highest accuracy

checking model with highest accuracy with model on the train value

In [48]:
```
print('ROC AUC Score for linear regression is:',roc(y_train,predlr))
print('ROC AUC Score for logistic regression: ',roc(y_train, predTrain2))
print('ROC AUC Score GaussianNB',roc(y_train,predG))
```

```
ROC AUC Score for linear regression is: 0.7155234794987406
ROC AUC Score for logistic regression:  0.8056094298942374
ROC AUC Score GaussianNB 0.8193416360940006
```

checking the model with highest accuracy with model on the test value

In [49]:
```
print('ROC AUC Score for linear regression: ',roc(y_test, predTest))
print('ROC AUC Score for logistic regression: ',roc(y_test, predTest2))
print('ROC AUC Score GaussianNB',roc(y_test,predGa))
```

```
ROC AUC Score for linear regression:  0.6999355050540584
ROC AUC Score for logistic regression:  0.7906276584177098
ROC AUC Score GaussianNB 0.8059013408552563
```

Best model to use is Gaussian naive_bayes classifier in both train and test values

In [50]:
```
##Best model in ordered form for model in train values

print('ROC AUC Score GaussianNB',roc(y_train,predG))
print('ROC AUC Score for logistic regression: ',roc(y_train, predTrain2))
print('ROC AUC Score for linear regression is:',roc(y_train,predlr))
```

```
ROC AUC Score GaussianNB 0.8193416360940006
ROC AUC Score for logistic regression:  0.8056094298942374
ROC AUC Score for linear regression is: 0.7155234794987406
```

In [51]:
```
##Best model in ordered form for model in test values
print('ROC AUC Score GaussianNB',roc(y_test,predGa))
print('ROC AUC Score for logistic regression: ',roc(y_test, predTest2))
print('ROC AUC Score for linear regression: ',roc(y_test, predTest))
```

```
ROC AUC Score GaussianNB 0.8059013408552563
ROC AUC Score for logistic regression:  0.7906276584177098
ROC AUC Score for linear regression:  0.6999355050540584
```

In [ ]: