Exercise 7: Use Naïve Bayes classifier to solve the credit card fraud detection problem over a skewed dataset.

Importing required packages

```
import numpy as np
import pandas as pd
from sklearn.naive_bayes import GaussianNB
```

Importing Other required libraries

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import plotly.express as px
```

Reading Credit Card Data into a Dataframe

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# loading wheat seeds data into a dataframe
from google.colab import drive
drive.mount("/content/gdrive")

# displaying the first 5 rows of wheet seeds data
df = pd.read_csv("gdrive/My Drive/ml/creditcard.csv")
```

```
Mounted at /content/gdrive
```

Getting Basic Description of the Credit Card Data

```
df.describe()
```

	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	2.848070e+
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+

8 rows × 31 columns

Inferences from the Description of the Dataset

- The data is presented with Time, Amount, Class and a series of columns with naming that ranges from V1 to V28
- Due to confidentiality issues, the actual names of V1-V28 is not provided by the source
- V1-V28 are principal components obtained via PCA
- This means V1 through V28 are important in determining whether a transaction is fraud or not and none of them can be neglected
- 'Time' and 'Amount' columns are not transformed with PCA
- Feature 'Class' is the target column, non-fraud transactions are represented by a 0 and
- fraud transactions are represented by 1

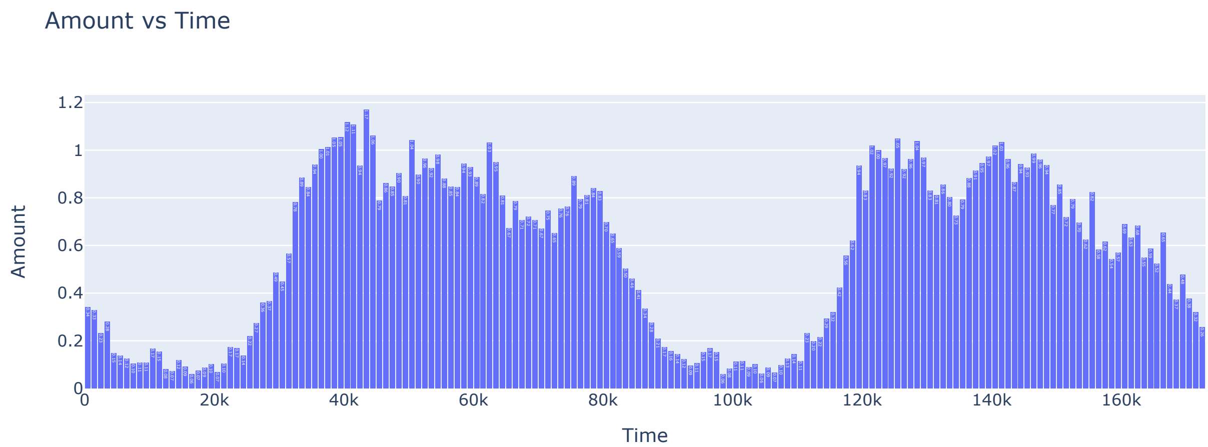
Displaying the Shape of Dataset and Unique Values of Class column

```
print('The total number of transactions in dataset : ', len(df))
print('The total number of columns : ',len(list(df)))
print('The dimension of data : ', df.shape)
print('The target column is : ', list(df)[30])
print('Total number of unique values in target column is : ', len(df['Class'].unique()))
print('The unique values in Class column : ', df.Class.unique())
```

```
The total number of transactions in dataset :  284807
The total number of columns :  31
The dimension of data :  (284807, 31)
The target column is :  Class
Total number of unique values in target column is :  2
The unique values in Class column :  [0 1]
```

Plot "Time vs Amount" to identify if there is any relationship between transaction amount over time

```
fig = px.histogram(df ,
                   x = 'Time' ,
                   y = 'Amount',
                   width = 1000,
                   height = 400,
                   title = 'Amount vs Time',
                   text_auto = '.2f',
                   histnorm = 'percent')

fig.update_layout( bargap = 0.2 , yaxis_title = 'Amount')
fig.show()
```



- The above graph clearly illustrates there is absolutely no relationship between transaction amount over time
- This means the transaction time column can be eliminated from the original data frame before further analysis

Deleting 'Time' column from original dataframe

```
df = df.drop(['Time'],axis=1)
```

Scale the 'Amount' column before further analysis, name it as a new column and drop the 'Amount' column

```
df['Normalized_Amount'] = StandardScaler().fit_transform(df['Amount'].values.reshape(-1, 1))
df = df.drop(['Amount'] , axis = 1)
df.head()
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.1
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.1
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.9
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.1
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.1

5 rows × 30 columns

Change the index of Normalized_Amount and insert the same in the beginning to have a better look of data frame

```
Normalized_Amount = df['Normalized_Amount']
df = df.drop(['Normalized_Amount'] , axis = 1)
df.insert(0, 'Normalized_Amount', Normalized_Amount)
df.head()
```

	Normalized_Amount	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	
0	0.244964	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	0.251412	-0.01
1	-0.342475	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.069083	-0.22
2	1.160686	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.524980	0.24
3	0.140534	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.208038	-0.10
4	-0.073403	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	0.408542	-0.00

5 rows × 30 columns

Well before diving into processing, let us see if there is/are any missing values in the dataframe.

```
print("Missing Values in Dataset:")
df.isnull().sum()
```

```
Missing Values in Dataset:
Normalized_Amount    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
Class                0
dtype: int64
```

Splitting the dataset into training data and test data

```
X = df.drop(['Class'], axis = 1)
y = df['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 27)
```

More on Class column

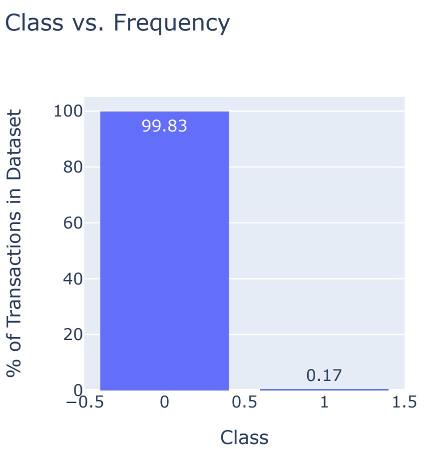
Checking number of fraud transactions and non-fraud transactions in the dataset

```
from collections import Counter

print('The number of Fraud & Non-Fraud Transactions in original dataset  %s' % Counter(df.Class))
```

```
The number of Fraud & Non-Fraud Transactions in original dataset  Counter({0: 284315, 1: 492})
```

Plotting the percentage of fraud transactions & non-fraud transactions in the dataset

```
fig = px.histogram(df ,
                   x = 'Class' ,
                   width = 400,
                   height = 400,
                   title = 'Class vs. Frequency',
                   text_auto = '.2f',
                   histnorm = 'percent')

fig.update_layout( bargap = 0.2 , yaxis_title = '% of Transactions in Dataset')
fig.show()
```



*Class column inference *

- The target column is heavily imbalanced
- Percentage of fraud transactions over total transactions is just 0.17%
- Building a model with this target column will definitely lead to overfitting issue
- Accuracy of such a model(irrespective of algorithm) will be > 99%

*Feature Engineering requirements *

- 'Class' column is heavily biased. So,it is not advised to proceed without doing something for the bias
- 'Time' and 'Amount' columns are not transformed. So, it is required to transform them to match with the other values(V1 - V28)

Dealing with class imbalance

The approach for handling imbalanced class data is Synthetic Minority Over-sampling Technique (SMOTE)

https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html

```
!pip install imblearn
```

```
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (from imblearn) (0.10.1)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.11.4)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.:
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (3
Installing collected packages: imblearn
Successfully installed imblearn-0.0
```

```
from imblearn.over_sampling import SMOTE

print('The number of Fraud & Non-Fraud Transactions in original dataset  %s' % Counter(y_train))

ros = SMOTE(random_state = 424)
X_train_ros, y_train_ros = ros.fit_resample(X_train, y_train)

print('The number of Fraud & Non-Fraud Transactions in Resampled datasete %s' % Counter(y_train_ros))
```

```
The number of Fraud & Non-Fraud Transactions in original dataset  Counter({0: 213245, 1: 360})
The number of Fraud & Non-Fraud Transactions in Resampled datasete Counter({0: 213245, 1: 213245})
```

Naive Bayes Classifier

```
nbclf = GaussianNB().fit(X_train_ros, y_train_ros)
print('Accuracy of GaussianNB classifier on test set: {:.2f}'.format(nbclf.score(X_test, y_test)*100))
```

```
Accuracy of GaussianNB classifier on test set: 97.65
```