The aim is to implement PCA and LDA technique on credit card dataset to reduce the dimensionality of the dataset.

In [110]:
```
#Importing the Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [111]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn.feature_selection import VarianceThreshold
from sklearn.preprocessing import StandardScaler
```

In [112]:
```
#Reading the dataset
dataset = pd.read_csv("creditcard.csv", nrows = 20000)
```

In [113]:
```
dataset.head()
```

Out[113]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V25	V26	V
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	0.128539	-0.189115	0.1335
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	0.167170	0.125895	-0.0089
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	-0.327642	-0.139097	-0.0553
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	0.647376	-0.221929	0.0627
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.206010	0.502292	0.2194

5 rows × 35 columns

Data Pre-Processing

1. Shape of the dataset

In [114]:
```
dataset.shape
```

Out[114]:
```
(20000, 35)
```

Conclusion: There are 20000 rows and 31 columns in the dataset

2. Checking for missing values

In [115]:
```
dataset.isna().sum()
```

Out[115]:
```
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
V29       0
V30       0
V31       0
V32       0
Amount    0
Class     0
dtype: int64
```

Conclusion: There are no missing values in the dataset

3. Checking for duplicate rows

In [116]:
```
dataset.duplicated().any()
```

Out[116]:
```
True
```

Conclusion: There are duplicate rows present in the dataset

Implementation of Dimensionality Reduction Techniques

In [117]:
```
#Setting the values for x and y variable
# Drop "Class" and assign it to target variable
X = dataset.drop('Class', 1)
y = dataset['Class']
```

In [118]:
```
X.shape, y.shape
```

Out[118]:
```
((20000, 34), (20000,))
```

In [119]:
```
#Splitting the dataset into train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0, stratify = y)
```

Removing constant features using variance threshold

In [120]:
```
#Removing constant features using variance threshold
constant_filter = VarianceThreshold(threshold=0.10)

#Applying the filter on the training set
constant_filter.fit(X_train)

#Remove constant features from training and test sets, we can use the transform() method of the constant_filter
X_train_filter = constant_filter.transform(X_train)
X_test_filter = constant_filter.transform(X_test)
X_train_filter.shape, X_test_filter.shape
```

Out[120]:
```
((16000, 33), (4000, 33))
```

In [121]:
```
#Number of duplicated features in the dataset, We remove one of the feature
X_train_T = X_train_filter.T
X_test_T = X_test_filter.T
X_train_T = pd.DataFrame(X_train_T)
X_test_T =pd.DataFrame(X_test_T)
X_train_T.duplicated().sum()
```

Out[121]:
```
4
```

Conclusion: We remove constant features from the data with the threshold value of 0.10. That means we remove the features which have 90% similarity among them. There are 4 similar columns in the dataset that are 90% similar.

Removing duplicate columns

In [122]:
```
duplicated_features = X_train_T.duplicated()
features_to_keep = [not index for index in duplicated_features]
X_train_unique = X_train_T[features_to_keep].T
X_test_unique = X_test_T[features_to_keep].T
```

In [123]:
```
#standardize the data to get the same scale
scaler = StandardScaler().fit(X_train_unique)
X_train_unique = scaler.transform(X_train_unique)
X_test_unique = scaler.transform(X_test_unique)
X_train_unique = pd.DataFrame(X_train_unique)
X_test_unique = pd.DataFrame(X_test_unique)
X_train_unique.shape, X_test_unique.shape
```

Out[123]:
```
((16000, 29), (4000, 29))
```

Conclusion: There are 4 columns which are similar. Therefore drop those columns.

In [124]:
```
#Correlation matrix
corrmat = X_train_unique.corr()
corrmat.head()
```

Out[124]:

	0	1	2	3	4	5	6	7	8	9	...	19	20	
0	1.000000	0.00749	-0.032406	-0.088782	0.027535	-0.085750	-0.043481	-0.054680	0.060932	-0.143732	...	-0.010458	0.016940	0.
1	0.007490	1.00000	-0.307000	0.393050	-0.130860	0.177574	0.133403	0.300916	-0.160428	0.015538	...	-0.000729	-0.126978	-0.
2	-0.032406	-0.30700	1.000000	-0.392820	0.167686	-0.233508	-0.067286	-0.155604	0.111692	-0.101254	...	-0.020067	0.004393	0.
3	-0.088782	0.39305	-0.392820	1.000000	-0.221292	0.377193	0.069352	0.503921	-0.365870	0.220517	...	-0.043102	-0.146534	-0.
4	0.027535	-0.13086	0.167686	-0.221292	1.000000	-0.149768	-0.053925	-0.194087	0.123479	-0.154267	...	-0.023042	0.018345	0.

5 rows × 29 columns

In [145]:
```
#Finding the number of correlated features
def get_correlation(data, threshold):
    corr_col = set()
    corrmat = data.corr()
    for i in range(len(corrmat.columns)):
        for j in range(i):
            if abs(corrmat.iloc[i, j]) > threshold:
                colname = corrmat.columns[i]
                corr_col.add(colname)
    return corr_col

corr_features = get_correlation(X_train_unique, 0.40)
print('correlated features: ', len(set(corr_features)) )
```

```
correlated features:  3
```

Conclusion: There are three correlated features

In [147]:
```
X_train_uncorr = X_train_unique.drop(labels=corr_features, axis = 1)
X_test_uncorr = X_test_unique.drop(labels = corr_features, axis = 1)
X_train_uncorr.shape, X_test_uncorr.shape
```

Out[147]:
```
((16000, 26), (4000, 26))
```

Conclusion: we can observe that the features are reduced from 29 to 26 features.

Feature Dimensionality Reduction by Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis is a supervised algorithm as it takes the class label into consideration. It is a way to reduce ‘dimensionality’ while at the same time preserving as much of the class discrimination information as possible.

LDA helps you find the boundaries around clusters of classes. It projects your data points on a line so that your clusters are as separated as possible, with each cluster having a relative (close) distance to a centroid.

In [149]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

In [150]:
```
#Transforming the data by using fit_transform()
lda = LDA(n_components=1)
X_train_lda = lda.fit_transform(X_train_uncorr, y_train)
X_test_lda = lda.transform(X_test_uncorr)
```

In [151]:
```
#Transformed data
X_train_lda.shape, X_test_lda.shape
```

Out[151]:
```
((16000, 1), (4000, 1))
```

In [152]:
```
from sklearn.ensemble import RandomForestClassifier
def run_randomForest(X_train, X_test, y_train, y_test):
    clf = RandomForestClassifier(n_estimators=100, random_state=0, n_jobs=-1)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print('Accuracy on test set: ')
    print(accuracy_score(y_test, y_pred))
```

In [153]:
```
#Running this on the transformed dataset
%%time
run_randomForest(X_train_lda, X_test_lda, y_train, y_test)
```

```
Accuracy on test set: 
0.9975
Wall time: 301 ms
```

In [154]:
```
#Running this on the original dataset
%%time
run_randomForest(X_train, X_test, y_train, y_test)
```

```
Accuracy on test set: 
0.99825
Wall time: 884 ms
```

Conclusion: Accuracy on the original dataset is more as compared to transformed dataset. But, the training time original dataset is double than tranformed version and the dimension also has been reduced. From this, we can observe LDA won't guarantee on the accuracy but it will give guarantee on the reduction in dimension and CPU time.

Feature Dimensionality Reduction by Principal Component Analysis (PCA)

PCA is that it is an Unsupervised dimensionality reduction technique, you can cluster the similar data points based on the feature correlation between them without any supervision (or labels)

Principal Component Analysis (PCA) is a linear dimensionality reduction technique that can be utilized for extracting information from a high-dimensional space by projecting it into a lower-dimensional sub-space. It tries to preserve the essential parts that have more variation of the data and remove the non-essential parts with fewer variation.

In [155]:
```
from sklearn.decomposition import PCA
```

In [156]:
```
#Removing the features
pca = PCA(n_components=2, random_state=42)
pca.fit(X_train_uncorr)
PCA(copy=True, iterated_power='auto', n_components=2, random_state=42, svd_solver='auto', tol=0.0, whiten=False)
```

Out[156]:
```
PCA(n_components=2, random_state=42)
```

In [157]:
```
#Training and testing dataset by PCA transformation
X_train_pca = pca.transform(X_train_uncorr)
X_test_pca = pca.transform(X_test_uncorr)
X_train_pca.shape, X_test_pca.shape
```

Out[157]:
```
((16000, 2), (4000, 2))
```

In [158]:

```
#Running this on the transformed dataset
%%time
run_randomForest(X_train_pca, X_test_pca, y_train, y_test)
```

```
Accuracy on test set:
0.99775
Wall time: 364 ms
```

In [159]:

```
#Running this on the original dataset
%%time
run_randomForest(X_train, X_test, y_train, y_test)
```

```
Accuracy on test set:
0.99825
Wall time: 883 ms
```

In [160]:
```
#Checking the accuracy for various selected components
for component in range(1,5):
    pca = PCA(n_components=component, random_state=42)
    pca.fit(X_train_uncorr)
    X_train_pca = pca.transform(X_train_uncorr)
    X_test_pca = pca.transform(X_test_uncorr)
    print('Selected Components: ', component)
    run_randomForest(X_train_pca, X_test_pca, y_train, y_test)
    print()
```

```
Selected Components:  1
Accuracy on test set:
0.9945

Selected Components:  2
Accuracy on test set:
0.99775

Selected Components:  3
Accuracy on test set:
0.99775

Selected Components:  4
Accuracy on test set:
0.99825
```

Conclusion: Component 4 is the best fit for the model as it shows maximum accuracy.