In [159…
```
import numpy as np
import math
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn_extra.cluster import KMedoids
from sklearn.decomposition import PCA
```

In [160…
```
def normalization(arr):
    norm=[] #Create empty restoring place
    cols=len(arr[0])
    for i in range(cols):
        print(f'Col #{i}:', end=' ')
        norm.append((arr[:,i]-arr[:,i].min())/(arr[:,i].max()-arr[:,i].min()))
        print(norm[i])
    return norm

def reshaped(arr):
    col=len(arr)
    row=len(arr[0])
    print(f'Inf {row} rows, {col} cols')
    arr=np.array(arr)
    res=np.array([[None]*col]*row)
    # print(res.shape)
    for i in range(len(arr)):
        res[:,i]=arr[i]
    print(f'Reshaped array: \n', res)
    return res

def accuracy(true_labels, predicted_labels):
    return ((true_labels == predicted_labels+1).sum() / len(true_labels))*100


def K_MeansAll(data, n_clusters, ground_truth):
    kmeans = KMeans(n_clusters=n_clusters)
    km_res=kmeans.fit(data)
    accuracyR = accuracy(ground_truth, km_res.labels_)
    return accuracyR

def compare_labels(labels1, labels2):
    if labels1>labels2:
        print(f'label 1:{labels1} > label 2:{labels2}')
    elif labels1<labels2:
        print(f'label 1:{labels1} < label 2:{labels2}')
    else:
        print(f'label 1:{labels1} = label 2:{labels2}')
```

In [161…
```
dt=pd.read_csv('BankNotes_fulldata.arff')
dt.head()
```

Out[161…

	V1	V2	V3	V4	Class (target)
0	3.62160	8.6661	-2.8073	-0.44699	1
1	4.54590	8.1674	-2.4586	-1.46210	1
2	3.86600	-2.6383	1.9242	0.10645	1
3	3.45660	9.5228	-4.0112	-3.59440	1
4	0.32924	-4.4552	4.5718	-0.98880	1

In [162…

```
truth=dt['Class (target)']
dt.drop('Class (target)', axis=1, inplace=True) #Removing the target column
```

In [163…

```
cols=dt.shape[1] # number of columns
rows=dt.shape[0] # number of rows
```

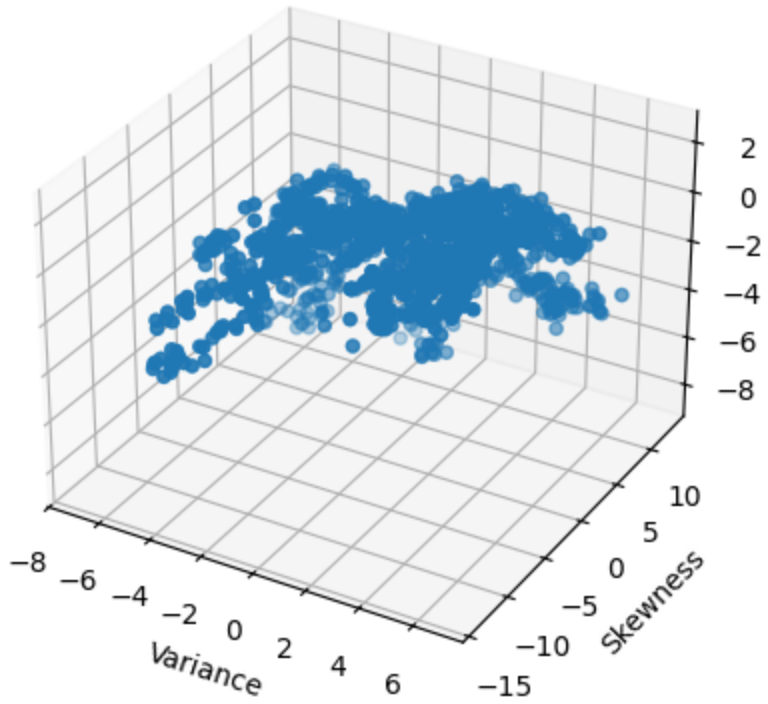
In [164…

```
#Range of each features

means=[]
stds=[]
for i in range(cols):
    vCol=dt[f'V{i+1}']
    print(f'V{i+1} \n Range: {vCol.max()-vCol.min()}, Mean: {vCol.mean()}, Std: {vC
```

```
V1
 Range: 13.8669, Mean: 0.43373525728862977, Std: 2.8427625862451658
V2
 Range: 26.7247, Mean: 1.9223531209912539, Std: 5.869046743580378
V3
 Range: 23.2135, Mean: 1.397627116618076, Std: 4.310030090315984
V4
 Range: 10.9977, Mean: -1.1916565211370262, Std: 2.1010131367390685
```

In [165…

```
fig=plt.figure()
ax=fig.add_subplot(111, projection='3d')


ax.scatter(dt['V1'],dt['V2'],dt['V4'])
ax.set_xlabel('Variance')
ax.set_ylabel('Skewness')
ax.set_zlabel('Entropy')

plt.show()
```



In [166…
```
taken_data=dt[['V1', 'V2', 'V3','V4']]
```

In [167…
```
#Because, with my norm function it returned the array with shape [[values of col 1]
#With the reshaped function it will transform the data from [[a1, a2, ..., an], [b1
#to
# [[a1, b1],
#  [a2, b2],
#  ...
#  [an, bn]]
norm = reshaped(normalization(taken_data.values))
pca = PCA(n_components=2)
norm = pca.fit_transform(norm)
```

```
Col #0: [0.76900389 0.83565902 0.78662859 ... 0.23738543 0.25084193 0.32452819]
Col #1: [0.83964273 0.82098209 0.41664827 ... 0.01176814 0.20170105 0.49074676]
Col #2: [0.10678269 0.12180412 0.31060805 ... 0.98560321 0.76158701 0.34334762]
Col #3: [0.73662766 0.64432563 0.78695091 ... 0.52475518 0.6606745  0.88594888]
Inf 1372 rows, 4 cols
Reshaped array:
 [[0.7690038869538253 0.8396427275142472 0.10678269110646822
  0.7366276585104158]
 [0.835659015353107 0.8209820877315742 0.12180412260107266
  0.6443256317229966]
 [0.7866285903842964 0.41664826920414444 0.3106080513494303
  0.78695090791711]
 ...
 [0.23738542861057624 0.011768139586225437 0.985603205031555
  0.5247551760822717]
 [0.25084193294824364 0.20170104809408523 0.761587007560256
  0.660674504669158]
 [0.32452819303521335 0.4907467623584175 0.343347620996403
  0.8859488802204096]]
```

In [168…

```
meansNorm=[]
stdsNorm=[]
for i in range(norm.shape[1]):
    vCol=norm[:,i]
    print(f'Col {i+1} \n Range: {vCol.max()-vCol.min()}, Mean: {vCol.mean()}, Std:
```

```
Col 1
 Range: 1.4233712433932628, Mean: 3.832373356140482e-16, Std: 0.29873771391446025
Col 2
 Range: 1.244468876700243, Mean: -2.1751308237554087e-16, Std: 0.2264570530426456
```

In [180…

```
taken_data=dt[['V1', 'V2']]
oldNorm=normalization(taken_data.values)#With V1 and V2 only (For compared)
oldNorm=reshaped(oldNorm)
```

```
Col #0: [0.76900389 0.83565902 0.78662859 ... 0.23738543 0.25084193 0.32452819]
Col #1: [0.83964273 0.82098209 0.41664827 ... 0.01176814 0.20170105 0.49074676]
Inf 1372 rows, 2 cols
Reshaped array:
 [[0.7690038869538253 0.8396427275142472]
 [0.835659015353107 0.8209820877315742]
 [0.7866285903842964 0.41664826920414444]
 ...
 [0.23738542861057624 0.011768139586225437]
 [0.25084193294824364 0.20170104809408523]
 [0.32452819303521335 0.4907467623584175]]
```

In [181…

```
#Apply K-means clustering to the min-max normalized data
kmeans = KMeans(n_clusters=2, init='k-means++', random_state=42)
km_res=kmeans.fit(norm)

#Apply K-means clustering to the min-max normalized data
kmeans = KMeans(n_clusters=2)
km_res_old=kmeans.fit(oldNorm)
```

In [182…

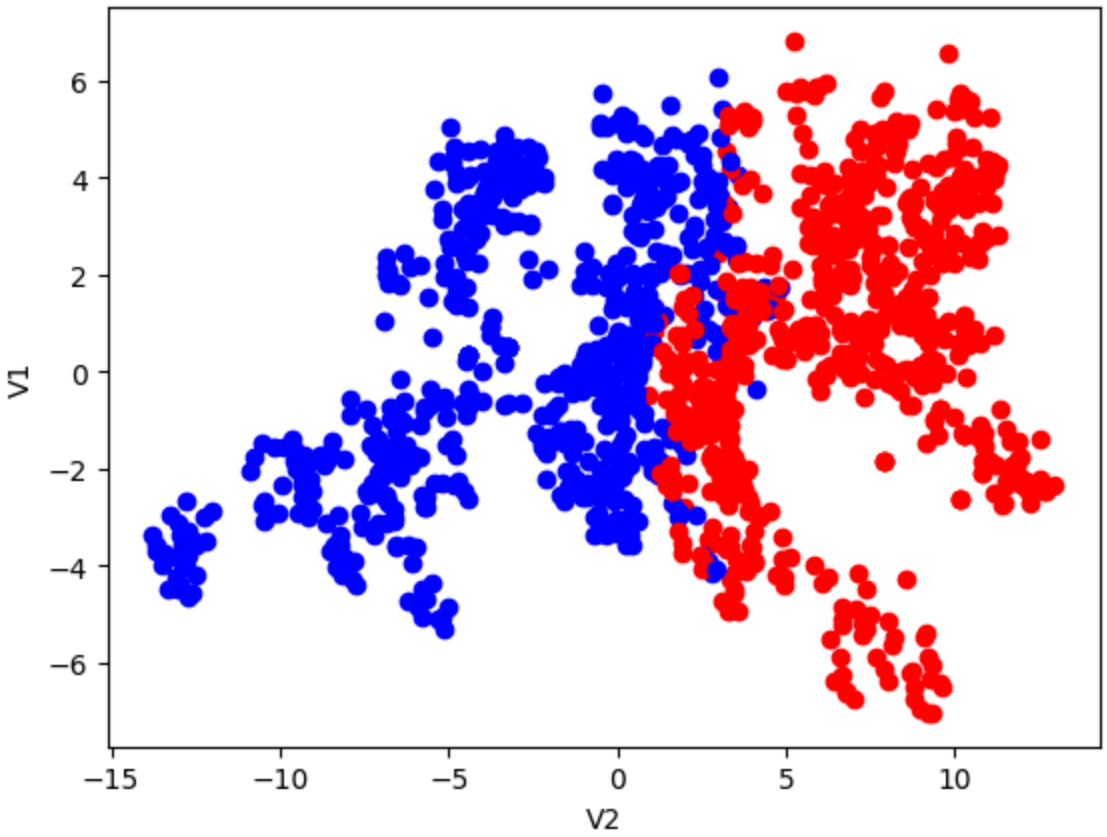
```
labels=km_res.labels_ #Getting the labels of each data point
```

```
labels_old=km_res_old.labels_ #Getting the labels of each data point
```

In [183…

```
plt.xlabel('V2')
plt.ylabel('V1')
colors=['red','blue','green','pink']
# Loop through each cluster
for i in range(rows):
    plt.scatter(dt.iloc[i][1], dt.iloc[i][0], color=colors[labels[i]])
```

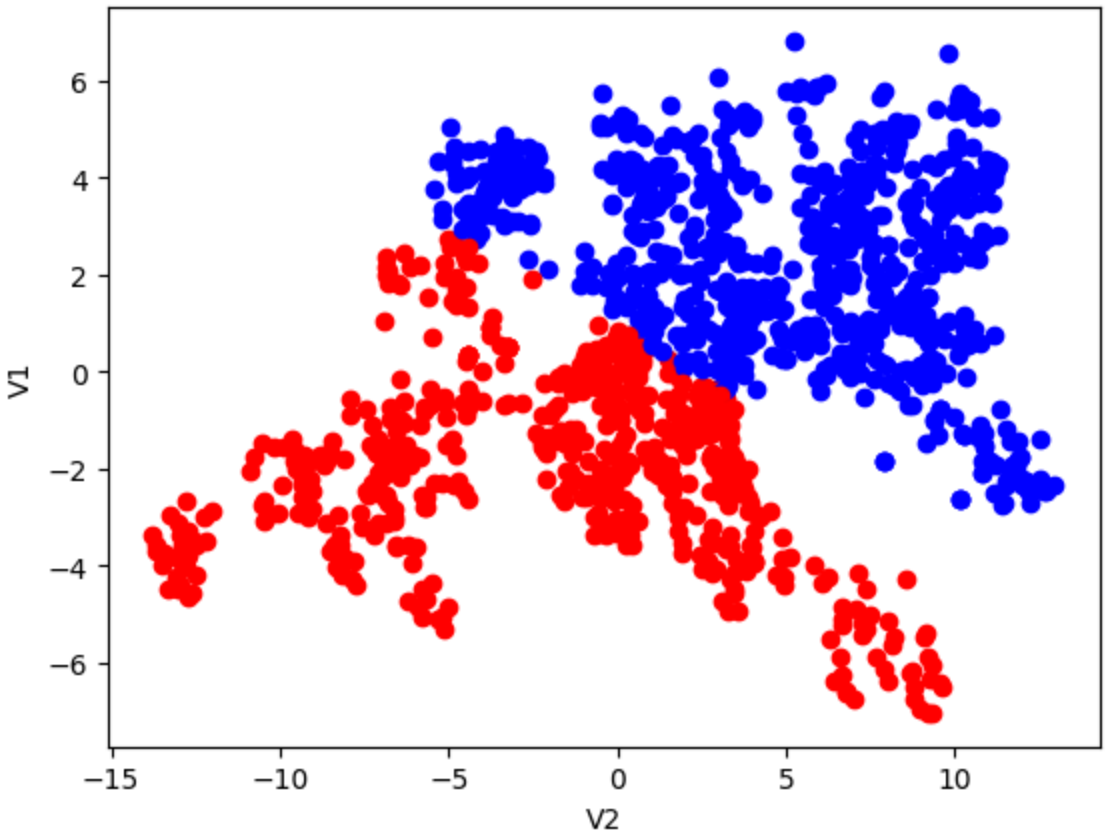
```
C:\Users\DELL\AppData\Local\Temp\ipykernel_26068\2524564833.py:6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  plt.scatter(dt.iloc[i][1], dt.iloc[i][0], color=colors[labels[i]])
```



In [184…

```
plt.xlabel('V2')
plt.ylabel('V1')
colors=['red','blue','green','pink']
# Loop through each cluster
for i in range(rows):
    plt.scatter(dt.iloc[i][1], dt.iloc[i][0], color=colors[labels_old[i]])
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_26068\363375154.py:6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  plt.scatter(dt.iloc[i][1], dt.iloc[i][0], color=colors[labels_old[i]])
```



In [185…
```
labels1=accuracy(truth, km_res.labels_)
labels2=accuracy(truth, km_res_old.labels_)
compare_labels(labels1, labels2)
```

```
label 1:57.5801749271137 > label 2:12.609329446064141
```

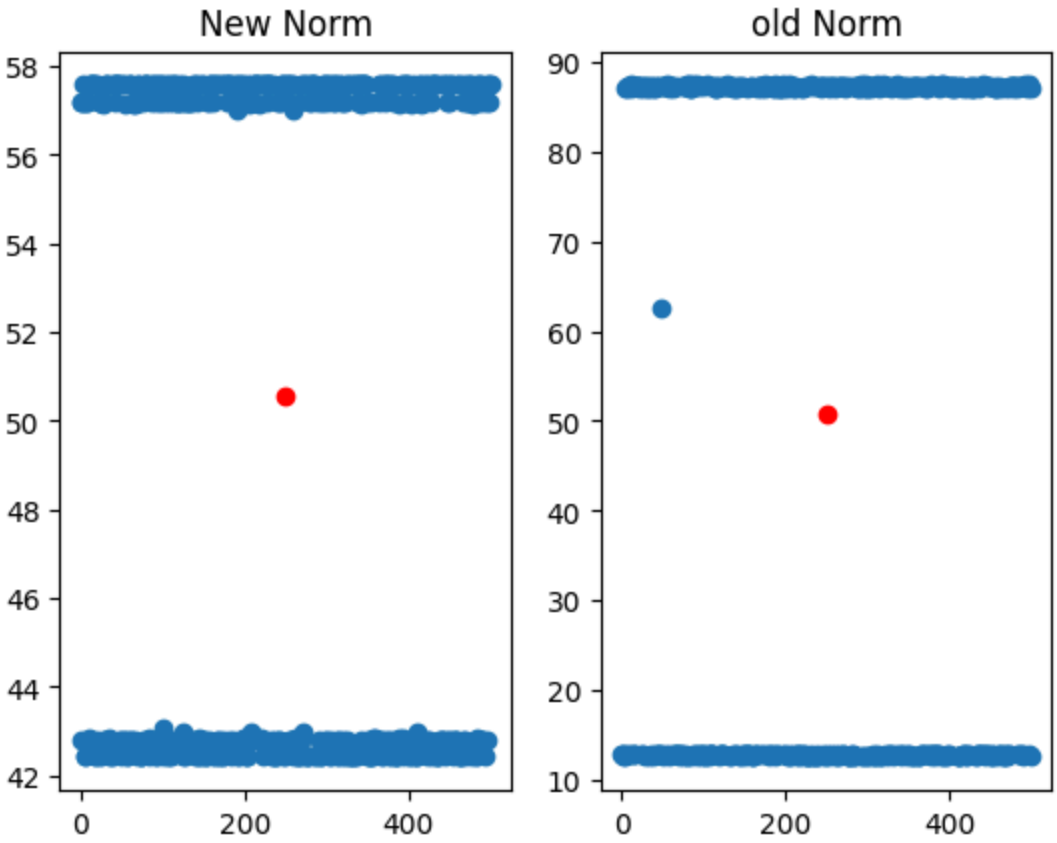
label 1:86.58892128279884 = label 2:86.58892128279884 With old_norm (V1,V2 only) label 1:86.58892128279884 = label 2:86.58892128279884 With old_norm (V1,V2,V4)

In [186…
```
new_res=[]
old_res=[]
for i in range(500):
    new_res.append(K_MeansAll(norm, 2, truth))
    old_res.append(K_MeansAll(oldNorm, 2, truth))

new_res=np.array(new_res)
old_res=np.array(old_res)

means=[new_res.mean(), old_res.mean()]
highests=[new_res.max(), old_res.max()]
smallest=[new_res.min(), old_res.min()]

print(f'New Norm: Mean: {means[0]}, Highest: {highests[0]}, Smallest: {smallest[0]}
print(f'Old Norm: Mean: {means[1]}, Highest: {highests[1]}, Smallest: {smallest[1]}


fig, (axs1,axs2) = plt.subplots(1, 2)

axs1.scatter(list(range(500)), new_res)
```

```
axs1.scatter(250,means[0], color='red')
axs1.set_title('New Norm')

axs2.scatter(list(range(500)), old_res)
axs2.scatter(250,means[1], color='red')
axs2.set_title('old Norm')
```

```
New Norm: Mean: 50.58352769679301, Highest: 57.5801749271137, Smallest: 42.419825072
8863
Old Norm: Mean: 50.69373177842566, Highest: 87.39067055393586, Smallest: 12.60932944
6064141
```

Out[186… `Text(0.5, 1.0, 'old Norm')`



Old Norm not tuned New Norm: Mean: 49.64620991253645, Highest: 86.07871720116617, Smallest: 13.92128279883382 Old Norm: Mean: 52.52376093294461, Highest: 87.39067055393586, Smallest: 12.609329446064141

New Norm: Mean: 49.63119533527697, Highest: 86.07871720116617, Smallest: 13.92128279883382 Old Norm: Mean: 53.268367346938774, Highest: 87.39067055393586, Smallest: 12.609329446064141

Both tuned already New Norm: Mean: 50.401457725947516, Highest: 86.07871720116617, Smallest: 13.92128279883382 Old Norm: Mean: 51.98265306122449, Highest: 87.39067055393586, Smallest: 12.609329446064141

New Norm: Mean: 49.59752186588921, Highest: 86.07871720116617, Smallest: 13.92128279883382 Old Norm: Mean: 54.75510204081633, Highest: 87.39067055393586,

Smallest: 12.609329446064141

In [ ]: