Github Link: https://github.com/SrikanthMajhi/Assignment---5

Question – 1

Source Code:

In [6]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler, normalize
```

In [7]:
```
df= pd.read_csv(r"D:datasets/CC.csv")

df.head()
```

Out[7]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUEN
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083

In [8]:
```
#Trying to find if there is any null value
df.isnull().sum()
```

In [8]:
```
#Trying to find if there is any null value
df.isnull().sum()
```


Out[8]:
```
CUST_ID                              0
BALANCE                              0
BALANCE_FREQUENCY                    0
PURCHASES                            0
ONEOFF_PURCHASES                     0
INSTALLMENTS_PURCHASES               0
CASH_ADVANCE                         0
PURCHASES_FREQUENCY                  0
ONEOFF_PURCHASES_FREQUENCY           0
PURCHASES_INSTALLMENTS_FREQUENCY     0
CASH_ADVANCE_FREQUENCY               0
CASH_ADVANCE_TRX                     0
PURCHASES_TRX                        0
CREDIT_LIMIT                         1
PAYMENTS                             0
MINIMUM_PAYMENTS                   313
PRC_FULL_PAYMENT                     0
TENURE                               0
dtype: int64
```

In [9]:
```
# Null values are found in the CREDIT_LIMIT and MINIMUM_PAYMENTS
# Now filling those null values with the mean of that coloumn values
df['CREDIT_LIMIT'].fillna(df['CREDIT_LIMIT'].mean(), inplace=True)
df['MINIMUM_PAYMENTS'].fillna(df['MINIMUM_PAYMENTS'].mean(), inplace=True)
```

In [10]:
```
# dropping the categorical values in the column of CUST_ID)
df.drop('CUST_ID', axis=1, inplace=True)
df.head(1)
```

Out[10]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOF
0	40.900749	0.818182	95.4	0.0	95.4	0.0	0.166667	

In [11]:

```
#Question 1(a)

# Performing Principal Component Analysis to transform data into 2 dimensions for visualization
# because it is highly difficult to visualize data in 17 dimensions
pca = PCA(n_components = 2)
X_principal = pca.fit_transform(df)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']
X_principal.head(2)
```

Out[11]:

	P1	P2
0	-4326.383956	921.566884
1	4118.916676	-2432.846347

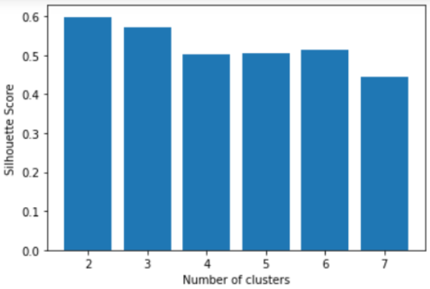
In [12]:

```
#Question 1(b)

silhouette_scores = []

for n_cluster in range(2, 8):
    silhouette_scores.append(
        silhouette_score(X_principal, KMeans(n_clusters = n_cluster).fit_predict(X_principal)))

# Plotting a bar graph to compare the results
k = [2, 3, 4, 5, 6, 7]
plt.bar(k, silhouette_scores)
plt.xlabel('Number of clusters', fontsize = 10)
plt.ylabel('Silhouette Score', fontsize = 10)
plt.show()
```



Perform Scaling+PCA+K-Means

In [13]:

```
# Question 1(c)
scaler = StandardScaler()
scaled_dataframe = scaler.fit_transform(df)
```

In [14]:

```
# Normalizing the Data
normalized_dataframe = normalize(scaled_dataframe)

# Converting the numpy array into a pandas DataFrame
normalized_dataframe = pd.DataFrame(normalized_dataframe)
```

In [15]:

```
# Performing Principal Component Analysis to transform data into 2 dimensions for visualization
# because it is highly difficult to visualize data in 17 dimensions
pca = PCA(n_components = 2)
X_principal = pca.fit_transform(normalized_dataframe)
```

```python
# Converting the numpy array into a pandas DataFrame
normalized_dataframe = pd.DataFrame(normalized_dataframe)
```

In [15]:

```python
# Performing Principal Component Analysis to transform data into 2 dimensions for visualization
# because it is highly difficult to visualize data in 17 dimensions
pca = PCA(n_components = 2)
X_principal = pca.fit_transform(normalized_dataframe)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']
X_principal.head(2)
```

Out[15]:

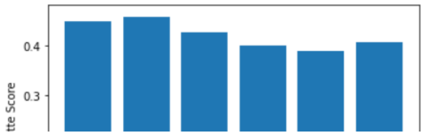
	P1	P2
0	-0.489826	-0.679679
1	-0.518791	0.545010

In [16]:

```python
silhouette_scores = []

for n_cluster in range(2, 8):
    silhouette_scores.append(
        silhouette_score(X_principal, KMeans(n_clusters = n_cluster).fit_predict(X_principal)))

# Plotting a bar graph to compare the results
k = [2, 3, 4, 5, 6, 7]
plt.bar(k, silhouette_scores)
plt.xlabel('Number of clusters', fontsize = 10)
plt.ylabel('Silhouette Score', fontsize = 10)
plt.show()
```

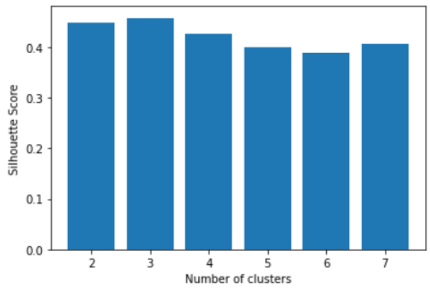


In [16]:

```python
silhouette_scores = []

for n_cluster in range(2, 8):
    silhouette_scores.append(
        silhouette_score(X_principal, KMeans(n_clusters = n_cluster).fit_predict(X_principal)))

# Plotting a bar graph to compare the results
k = [2, 3, 4, 5, 6, 7]
plt.bar(k, silhouette_scores)
plt.xlabel('Number of clusters', fontsize = 10)
plt.ylabel('Silhouette Score', fontsize = 10)
plt.show()
```



After Performing the Feature Scaling it increased the Silhouette Score.

Question – 2

Source Code:

```
In [2]: from sklearn.decomposition import PCA
        from sklearn.preprocessing import StandardScaler
        from sklearn.model_selection import train_test_split
        import pandas as pd
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import accuracy_score
        import matplotlib.pyplot as plt
        import warnings
        warnings.filterwarnings('ignore')
        from sklearn.cluster import KMeans
        from sklearn.metrics import silhouette_score
        from sklearn.preprocessing import StandardScaler, normalize
        from sklearn.svm import SVC
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import classification_report
        from sklearn.metrics import confusion_matrix
        from sklearn.svm import SVC
```

```
In [3]: df= pd.read_csv(r"D:datasets/pd_speech_features.csv")

        df.head()
```

Out[3]:

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPulses	stdDevPeriodPulses	locPctJitter	...	tqwt_kurtosisValue_dec_28	tq
0	0	1	0.85247	0.71826	0.57227	240	239	0.008064	0.000087	0.00218	...	1.5620	
1	0	1	0.76686	0.69481	0.53966	234	233	0.008258	0.000073	0.00195	...	1.5589	
2	0	1	0.85083	0.67604	0.58982	232	231	0.008340	0.000060	0.00176	...	1.5643	
3	1	0	0.41121	0.79672	0.59257	178	177	0.010858	0.000183	0.00419	...	3.7805	
4	1	0	0.32790	0.79782	0.53028	236	235	0.008162	0.002669	0.00535	...	6.1727	

5 rows × 755 columns

```
In [4]: df['class'].value_counts()
Out[4]: 1    564
        0    192
        Name: class, dtype: int64
```

```
In [5]: #Question 2(a)
        # Performing Scaling
```

```
In [6]: scaler = StandardScaler()
```

```
In [7]: X = df.drop('class',axis=1).values
        y = df['class'].values

        X_Scale = scaler.fit_transform(X)
```

```
In [8]: #Question 2 (b)
        # Applying PCA when k = 3
```

```
In [9]: pca3 = PCA(n_components=3)
        principalComponents = pca3.fit_transform(X_Scale)

        principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2', 'principal co

        finalDf = pd.concat([principalDf, df[['class']]], axis = 1)
        finalDf.head()
```

Out[9]:

principal component 1	principal component 2	principal component 3	class

Out[9]:

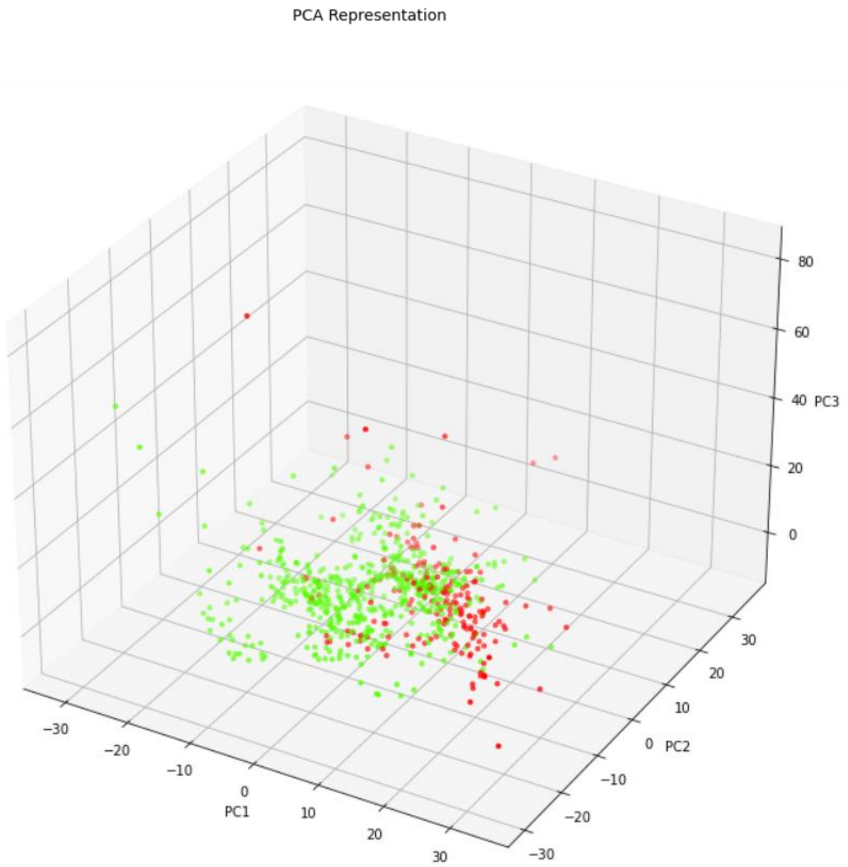
	principal component 1	principal component 2	principal component 3	class
0	-10.047372	1.471076	-6.846400	1
1	-10.637725	1.583747	-6.830982	1
2	-13.516185	-1.253543	-6.818695	1
3	-9.155083	8.833600	15.290865	1
4	-6.764469	4.611469	15.637086	1

In [10]:
```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(9,9))
axes = Axes3D(fig)
axes.set_title('PCA Representation', size=14)
axes.set_xlabel('PC1')
axes.set_ylabel('PC2')
axes.set_zlabel('PC3')

axes.scatter(finalDf['principal component 1'],finalDf['principal component 2'],finalDf['principal component 3'],c=finalDf['class'
```

Out[10]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x1dca2200970>



In [11]:
```
#Question 2(c)
X = df.drop('class',axis=1).values
y = df['class'].values
```

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,random_state=0)
```

In [13]:
```
from sklearn.svm import SVC, LinearSVC

classifier = LinearSVC()

classifier.fit(X_train, y_train)


y_pred = classifier.predict(X_train)

# Summary of the predictions made by the classifier
print(classification_report(y_train, y_pred))
print(confusion_matrix(y_train, y_pred))
# Accuracy score
from sklearn.metrics import accuracy_score
print('accuracy for our training dataset with PCA is ',accuracy_score(y_train, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       135
           1       0.74      1.00      0.85       394

    accuracy                           0.74       529
   macro avg       0.37      0.50      0.43       529
weighted avg       0.55      0.74      0.64       529

[[  0 135]
 [  0 394]]
accuracy for our training dataset with PCA is  0.7448015122873346
```

In [14]:
```
from sklearn.svm import SVC, LinearSVC

classifier = LinearSVC()

classifier.fit(X_train, y_train)


y_pred = classifier.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score
from sklearn.metrics import accuracy_score
print('accuracy for our test dataset with PCA is',accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        57
           1       0.75      1.00      0.86       170

    accuracy                           0.75       227
   macro avg       0.37      0.50      0.43       227
weighted avg       0.56      0.75      0.64       227

[[  0  57]
 [  0 170]]
accuracy for our test dataset with PCA is 0.748898678414097
```

In [15]:
```
scaler = StandardScaler()
```

In [16]:
```
# Fit on training set only.
scaler.fit(X_train)

# Apply transform to both the training set and the test set.
```

In [15]:
```
scaler = StandardScaler()
```

In [16]:
```
# Fit on training set only.
scaler.fit(X_train)

# Apply transform to both the training set and the test set.
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [17]:
```
from sklearn.svm import SVC, LinearSVC

classifier = LinearSVC()

classifier.fit(X_train_scaled, y_train)


y_pred = classifier.predict(X_train_scaled)

# Summary of the predictions made by the classifier
print(classification_report(y_train, y_pred))
print(confusion_matrix(y_train, y_pred))
# Accuracy score
from sklearn.metrics import accuracy_score
print('accuracy for our training dataset with PCA is ',accuracy_score(y_train, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       135
           1       1.00      1.00      1.00       394

    accuracy                           1.00       529
   macro avg       1.00      1.00      1.00       529
weighted avg       1.00      1.00      1.00       529

[[135   0]
 [  0 394]]
```


```
# Summary of the predictions made by the classifier
print(classification_report(y_train, y_pred))
print(confusion_matrix(y_train, y_pred))
# Accuracy score
from sklearn.metrics import accuracy_score
print('accuracy for our training dataset with PCA is ',accuracy_score(y_train, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       135
           1       1.00      1.00      1.00       394

    accuracy                           1.00       529
   macro avg       1.00      1.00      1.00       529
weighted avg       1.00      1.00      1.00       529

[[135   0]
 [  0 394]]
```


```
accuracy for our training dataset with PCA is  1.0
```

In [19]:
```
from sklearn.svm import SVC, LinearSVC

classifier = LinearSVC()

classifier.fit(X_train, y_train)


y_pred = classifier.predict(X_test_scaled)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score
from sklearn.metrics import accuracy_score
print('accuracy for our test dataset with PCA is',accuracy_score(y_test, y_pred))
```

```python
# Accuracy score
from sklearn.metrics import accuracy_score
print('accuracy for our test dataset with PCA is',accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.27      0.61      0.37        57
           1       0.77      0.44      0.56       170

    accuracy                           0.48       227
   macro avg       0.52      0.52      0.46       227
weighted avg       0.64      0.48      0.51       227

[[35 22]
 [96 74]]
accuracy for our test dataset with PCA is 0.4801762114537445
```

SVM Performance without scaling for Training set & Test Set is 0.7448015122873346 and 0.748898678414097

SVM Performance without scaling for Training set & Test Set is 1.0 and 0.4801762114537445

In [ ]:

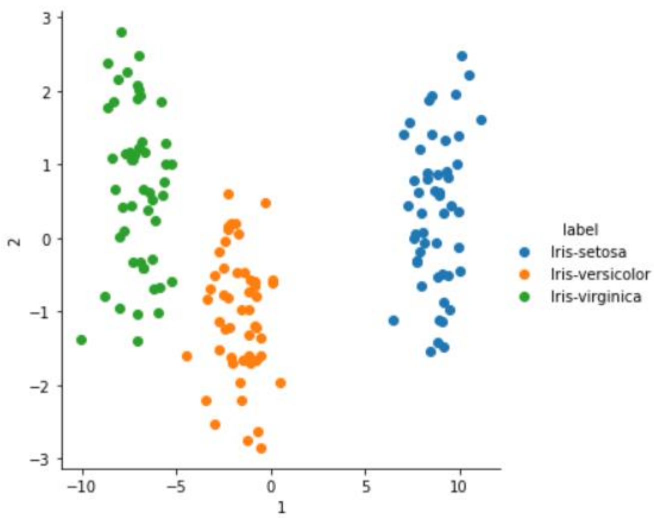
Question – 3

LDA on IRIS Dataset

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tqdm import tqdm
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sn
from sklearn.metrics.pairwise import euclidean_distances
import warnings
warnings.filterwarnings("ignore")
```

In [4]:
```
dataset = pd.read_csv('D:datasets/iris.csv') #read the data into dataframe
X = dataset.iloc[:, :-1].values   #store the dependent features in X
y = dataset.iloc[:, 5].values   #store the independent variable in y
X = StandardScaler().fit_transform(X)
```

In [6]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis(n_components=2)
lda_data = lda.fit(X, y).transform(X)
# attaching the label for each 2-d data point
lda_data = np.vstack((lda_data.T, y)).T
# creating a new data fram which help us in ploting the result data
lda_df = pd.DataFrame(data=lda_data, columns=("1", "2", "label"))
sn.FacetGrid(lda_df, hue="label", size=5).map(plt.scatter, '1', '2').add_legend()
plt.show()
```



Question - 4

Differences b/w PCA and LDA

PCA

In [1]:
```
"""Principal Component Analysis (PCA) works by identifying the directions (components) that maximize the variance in a dataset. I
```

Out[1]:
```
'Principal Component Analysis (PCA) works by identifying the directions (components) that maximize the variance in a dataset. I
n other words, it seeks to find the linear combination of features that captures as much variance as possible. The first compon
ent is the one that captures the maximum variance, the second component is orthogonal to the first and captures the remaining v
ariance, and so on. '
```

LDA

In [2]:
```
"""Linear discriminant analysis (LDA) is another linear transformation technique that is used for dimensionality reduction. Unlik
```

Out[2]:
```
'Linear discriminant analysis (LDA) is another linear transformation technique that is used for dimensionality reduction. Unlik
e PCA, however, LDA is a supervised learning method, which means it takes class labels into account when finding directions of
maximum variance. This makes LDA particularly well-suited for classification tasks where you want to maximize class separabilit
y.'
```

In [ ]: