Import Necessary Package

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn as sk
```

```
data=pd.read_csv('C:\Rohith\Backup\Desktop\SEM 6\Machine Learning Lab\Practices\Obesity D
```

```
data.head()
```

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Som
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Som
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Som
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Som
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Som

```
data.describe()
```

	Age	Height	Weight	FCVC	NCP	CH2O	
count	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	21
mean	24.312600	1.701677	86.586058	2.419043	2.685628	2.008011	
std	6.345968	0.093305	26.191172	0.533927	0.778039	0.612953	
min	14.000000	1.450000	39.000000	1.000000	1.000000	1.000000	
25%	19.947192	1.630000	65.473343	2.000000	2.658738	1.584812	
50%	22.777890	1.700499	83.000000	2.385502	3.000000	2.000000	
75%	26.000000	1.768464	107.430682	3.000000	3.000000	2.477420	

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
```

```
 0   Gender                          2111 non-null   object
 1   Age                             2111 non-null   float64
 2   Height                          2111 non-null   float64
 3   Weight                          2111 non-null   float64
 4   family_history_with_overweight  2111 non-null   object
 5   FAVC                            2111 non-null   object
 6   FCVC                            2111 non-null   float64
 7   NCP                             2111 non-null   float64
 8   CAEC                            2111 non-null   object
 9   SMOKE                           2111 non-null   object
 10  CH2O                            2111 non-null   float64
 11  SCC                             2111 non-null   object
 12  FAF                             2111 non-null   float64
 13  TUE                             2111 non-null   float64
 14  CALC                            2111 non-null   object
 15  MTRANS                          2111 non-null   object
 16  NObeyesdad                      2111 non-null   object
dtypes: float64(8), object(9)
memory usage: 280.5+ KB
```

```
data.shape
```

```
(2111, 17)
```

```
data.isnull().sum()
```

```
Gender                            0
Age                               0
Height                            0
Weight                            0
family_history_with_overweight    0
FAVC                              0
FCVC                              0
NCP                               0
CAEC                              0
SMOKE                             0
CH2O                              0
SCC                               0
FAF                               0
TUE                               0
CALC                              0
MTRANS                            0
NObeyesdad                        0
dtype: int64
```

```
data.duplicated().sum()
```

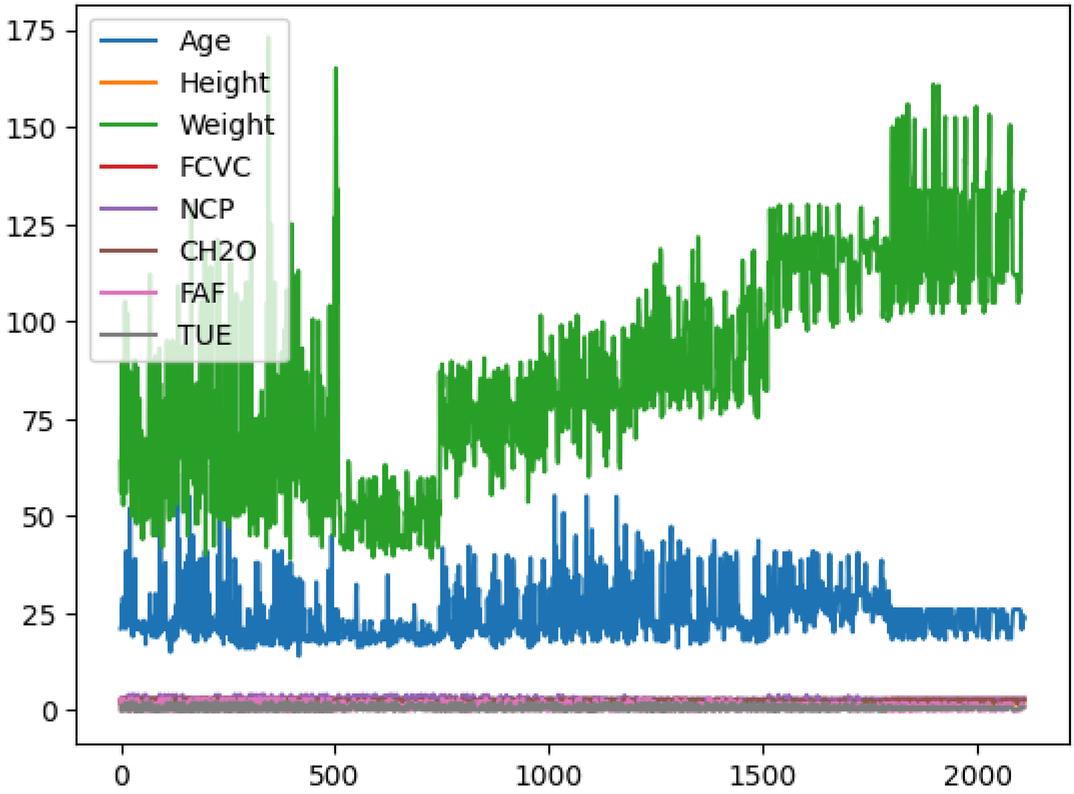
```
24
```

```
data.drop_duplicates(inplace=True)
data.shape
```

```
(2087, 17)
```

```
data.plot()
```

```
<Axes: >
```



```
data['Weight'].unique()
```

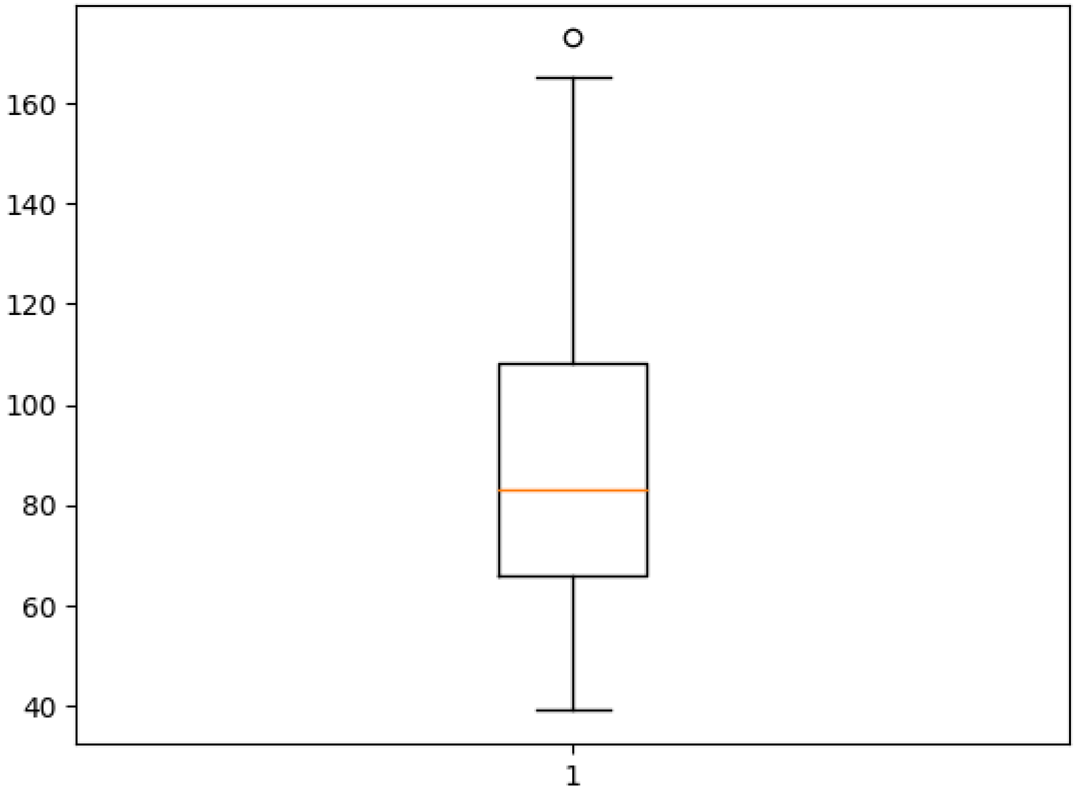
```
array([ 64.       ,  56.       ,  77.       , ..., 133.689352, 133.346641,
       133.472641])
```

```
max(data['Weight'].unique())
```

```
173.0
```

```
plt.boxplot(data['Weight'])
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x1d5a70e20b0>,
  <matplotlib.lines.Line2D at 0x1d5a70e2230>],
 'caps': [<matplotlib.lines.Line2D at 0x1d5a70e24d0>,
  <matplotlib.lines.Line2D at 0x1d5a70e2770>],
 'boxes': [<matplotlib.lines.Line2D at 0x1d5a70e1e10>],
 'medians': [<matplotlib.lines.Line2D at 0x1d5a70e2a10>],
 'fliers': [<matplotlib.lines.Line2D at 0x1d5a70e2cb0>],
 'means': []}
```

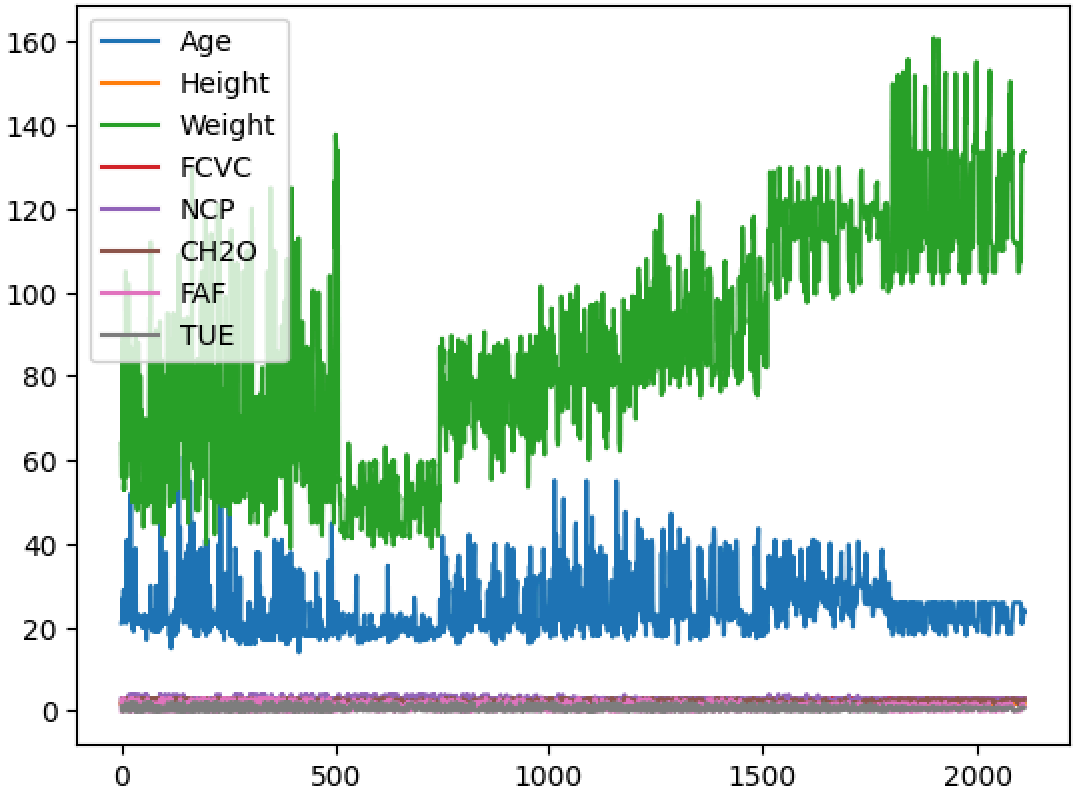


```
df=data[data['Weight']<=165]
df.shape
```

```
(2085, 17)
```

```
df.plot()
```

```
<Axes: >
```



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2085 entries, 0 to 2110
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Gender                          2085 non-null   object 
 1   Age                             2085 non-null   float64
 2   Height                          2085 non-null   float64
 3   Weight                          2085 non-null   float64
 4   family_history_with_overweight  2085 non-null   object 
 5   FAVC                            2085 non-null   object 
 6   FCVC                            2085 non-null   float64
 7   NCP                             2085 non-null   float64
 8   CAEC                            2085 non-null   object 
 9   SMOKE                           2085 non-null   object 
 10  CH2O                            2085 non-null   float64
 11  SCC                             2085 non-null   object 
 12  FAF                             2085 non-null   float64
 13  TUE                             2085 non-null   float64
 14  CALC                            2085 non-null   object 
 15  MTRANS                          2085 non-null   object 
 16  NObeyesdad                      2085 non-null   object 
dtypes: float64(8), object(9)
memory usage: 293.2+ KB
```

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

```
le.fit(df['Gender'])
df['Gender']=le.transform(df['Gender'])

le.fit(df['family_history_with_overweight'])
df['family_history_with_overweight']=le.transform(df['family_history_with_overweight'])

le.fit(df['FAVC'])
df['FAVC']=le.transform(df['FAVC'])

le.fit(df['CAEC'])
df['CAEC']=le.transform(df['CAEC'])

le.fit(df['SMOKE'])
df['SMOKE']=le.transform(df['SMOKE'])

le.fit(df['SCC'])
df['SCC']=le.transform(df['SCC'])

le.fit(df['CALC'])
df['CALC']=le.transform(df['CALC'])

le.fit(df['MTRANS'])
df['MTRANS']=le.transform(df['MTRANS'])

le.fit(df['NObeyesdad'])
df['NObeyesdad']=le.transform(df['NObeyesdad'])
```

```
C:\Users\rohit\AppData\Local\Temp\ipykernel_16104\288179486.py:2: SettingWithCopyWarn
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['Gender']=le.transform(df['Gender'])
C:\Users\rohit\AppData\Local\Temp\ipykernel_16104\288179486.py:5: SettingWithCopyWarn
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['family_history_with_overweight']=le.transform(df['family_history_with_overweigh
C:\Users\rohit\AppData\Local\Temp\ipykernel_16104\288179486.py:8: SettingWithCopyWarn
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['FAVC']=le.transform(df['FAVC'])
C:\Users\rohit\AppData\Local\Temp\ipykernel_16104\288179486.py:11: SettingWithCopyWar
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['CAEC']=le.transform(df['CAEC'])
C:\Users\rohit\AppData\Local\Temp\ipykernel_16104\288179486.py:14: SettingWithCopyWar
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

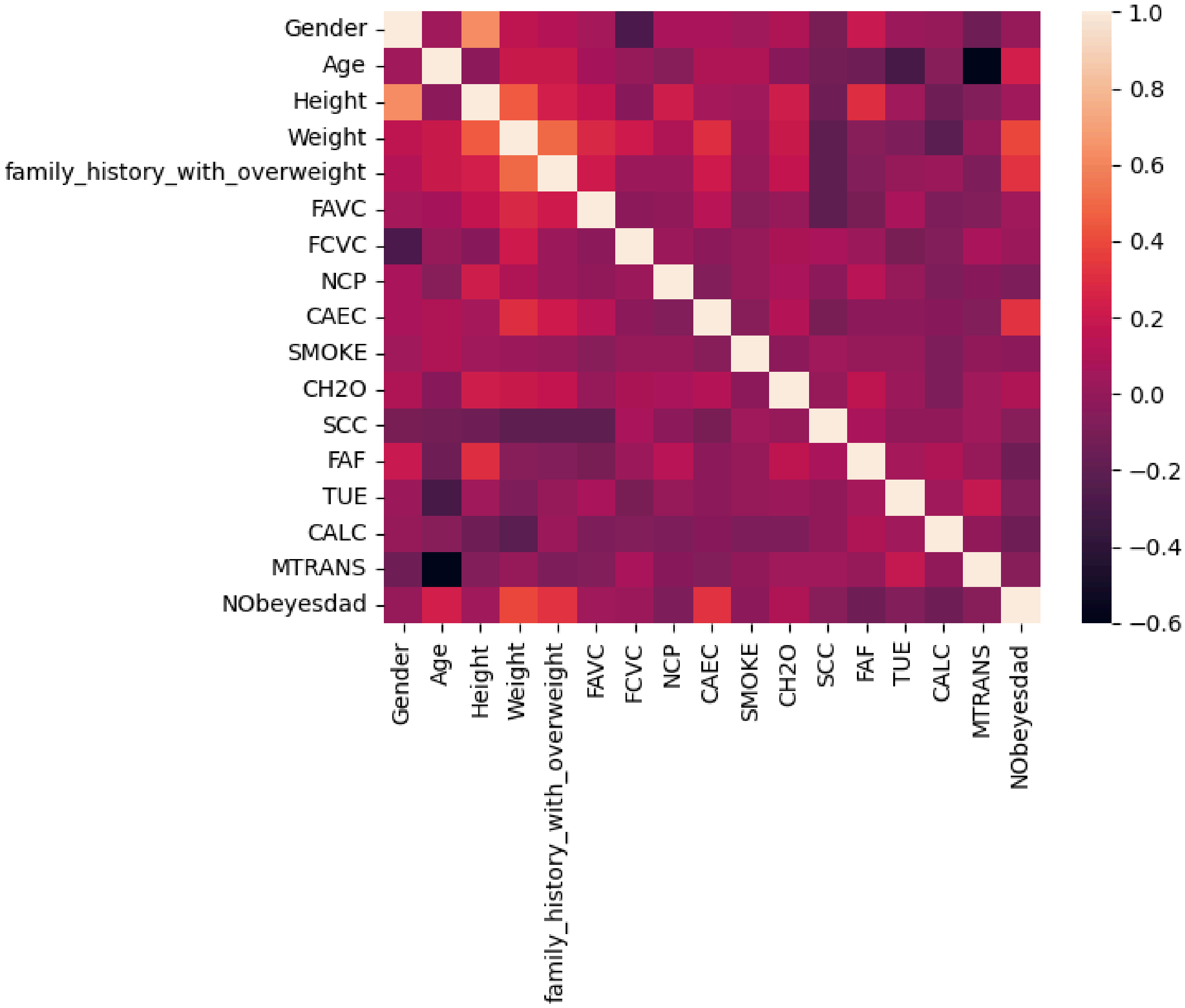
```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['SMOKE']=le.transform(df['SMOKE'])
C:\Users\rohit\AppData\Local\Temp\ipykernel_16104\288179486.py:17: SettingWithCopyWar
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['SCC']=le.transform(df['SCC'])
C:\Users\rohit\AppData\Local\Temp\ipykernel_16104\288179486.py:20: SettingWithCopyWar
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['CALC']=le.transform(df['CALC'])
C:\Users\rohit\AppData\Local\Temp\ipykernel_16104\288179486.py:23: SettingWithCopyWar
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['MTRANS']=le.transform(df['MTRANS'])
C:\Users\rohit\AppData\Local\Temp\ipykernel_16104\288179486.py:26: SettingWithCopyWar
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['NObeyesdad']=le.transform(df['NObeyesdad'])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2085 entries, 0 to 2110
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Gender                          2085 non-null   int32  
 1   Age                             2085 non-null   float64
 2   Height                          2085 non-null   float64
 3   Weight                          2085 non-null   float64
 4   family_history_with_overweight  2085 non-null   int32  
 5   FAVC                            2085 non-null   int32  
 6   FCVC                            2085 non-null   float64
 7   NCP                             2085 non-null   float64
 8   CAEC                            2085 non-null   int32  
 9   SMOKE                           2085 non-null   int32  
 10  CH2O                            2085 non-null   float64
 11  SCC                             2085 non-null   int32  
 12  FAF                             2085 non-null   float64
 13  TUE                             2085 non-null   float64
 14  CALC                            2085 non-null   int32  
 15  MTRANS                          2085 non-null   int32  
 16  NObeyesdad                      2085 non-null   int32  
dtypes: float64(8), int32(9)
memory usage: 219.9 KB
```

```
sns.heatmap(df.corr())
```

```
<Axes: >
```



Training and Testing

```
X=df.drop('NObeyesdad',axis=1)
y=df['NObeyesdad']
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

```
print("X_train = ",X_train.shape)
print("X_test = ",X_test.shape)
print("y_train = ",y_train.shape)
print("y_test = ",y_test.shape)
```

```
X_train =  (1668, 16)
X_test =  (417, 16)
```

```
     y_train =  (1668,)
     y_test =  (417,)
```

Model 1 Linear Regression

```
from sklearn.linear_model import LinearRegression
li=LinearRegression()
li
```

```
▾ LinearRegression ⓘ ?
LinearRegression()
```

```
li.fit(X_train,y_train)
y_pred=li.predict(X_test)
accuracy=li.score(X_test,y_test)
accuracy
```

```
0.2179840640752998
```

```
from sklearn.metrics import mean_absolute_error,mean_squared_error

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE):", mae)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
Mean Absolute Error (MAE): 1.4145600564157936
Mean Squared Error (MSE): 2.902726370515661
Root Mean Squared Error (RMSE): 1.703738938486663
```

Model 2 Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr
```

```
▾ LogisticRegression ⓘ ?
LogisticRegression()
```

```
lr.fit(X_train,y_train)
y_pred=lr.predict(X_test)
accuracy=lr.score(X_test,y_test)
accuracy
```

```
c:\Python310\lib\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarn
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
0.7170263788968825
```

```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score

print(mean_squared_error(y_test,y_pred))
print(mean_absolute_error(y_test,y_pred))
print(r2_score(y_test,y_pred))
```

```
2.431654676258993
0.709832134292566
0.34489425981873123
```

Model 3 SVM

```
from sklearn.svm import SVC
svm_classifier=SVC(probability=True)
svm_classifier
```

```
▾ SVC
SVC(probability=True)
```

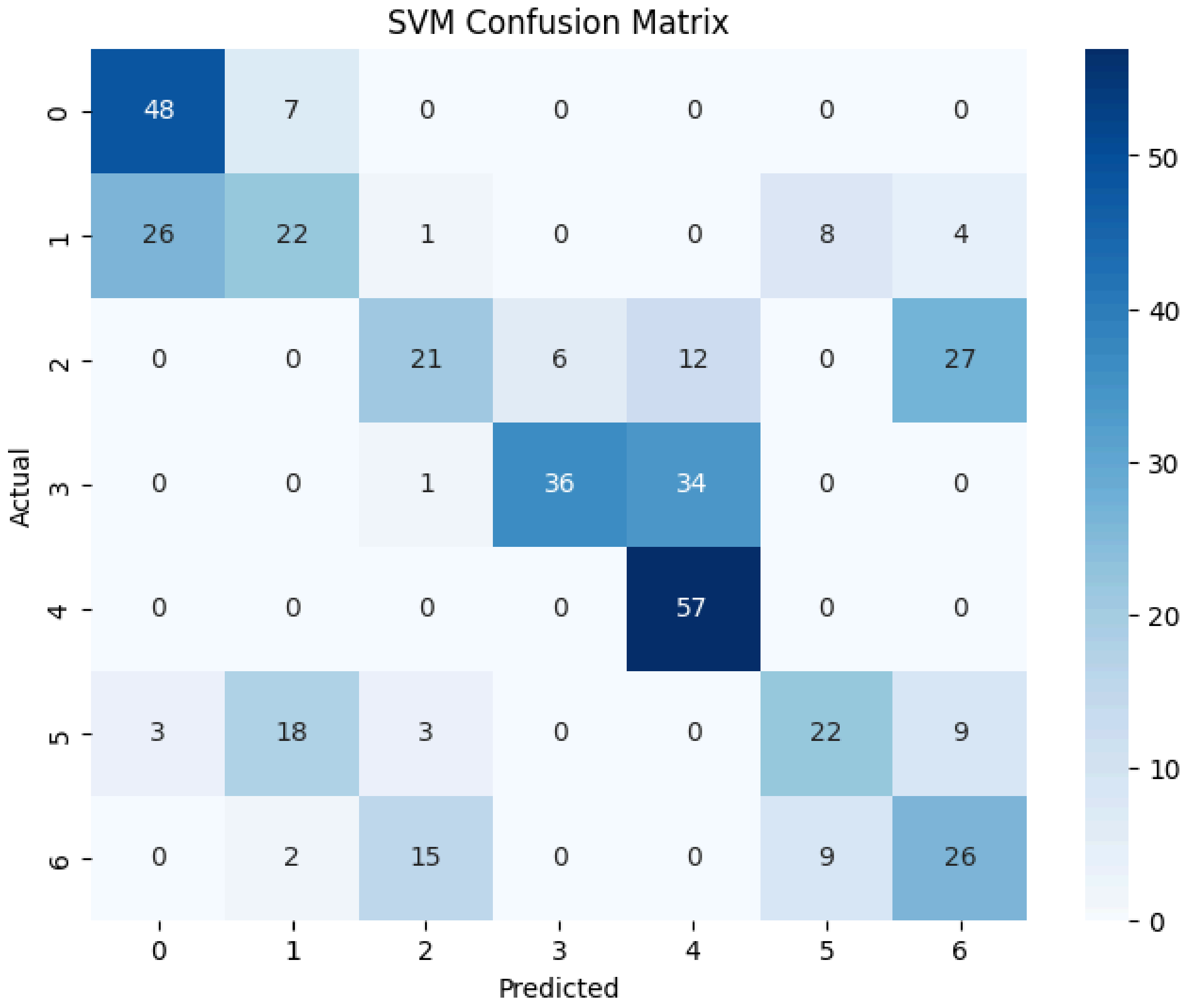
```
from sklearn.metrics import accuracy_score

svm_classifier.fit(X_train,y_train)
y_pred=svm_classifier.predict(X_test)
accuracy=accuracy_score(y_test,y_pred)
accuracy
```

```
0.5563549160671463
```

```
#Confusion matrix
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred)
print("SVM Confusion Matrix: \n",cm)
```

```
SVM Confusion Matrix:
 [[48  7  0  0  0  0  0]
 [26 22  1  0  0  8  4]
 [ 0  0 21  6 12  0 27]
 [ 0  0  1 36 34  0  0]
 [ 0  0  0  0 57  0  0]
 [ 3 18  3  0  0 22  9]
 [ 0  2 15  0  0  9 26]]
```

```
#Plot Confusion Matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('SVM Confusion Matrix')
plt.show()
```



```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.62      0.87      0.73        55
           1       0.45      0.36      0.40        61
           2       0.51      0.32      0.39        66
           3       0.86      0.51      0.64        71
           4       0.55      1.00      0.71        57
           5       0.56      0.40      0.47        55
           6       0.39      0.50      0.44        52

    accuracy                           0.56       417
   macro avg       0.56      0.57      0.54       417
weighted avg       0.57      0.56      0.54       417
```

Model 4 Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

dt_classifier=DecisionTreeClassifier()
dt_classifier
```

```
▾ DecisionTreeClassifier ⓘ ?
DecisionTreeClassifier()
```

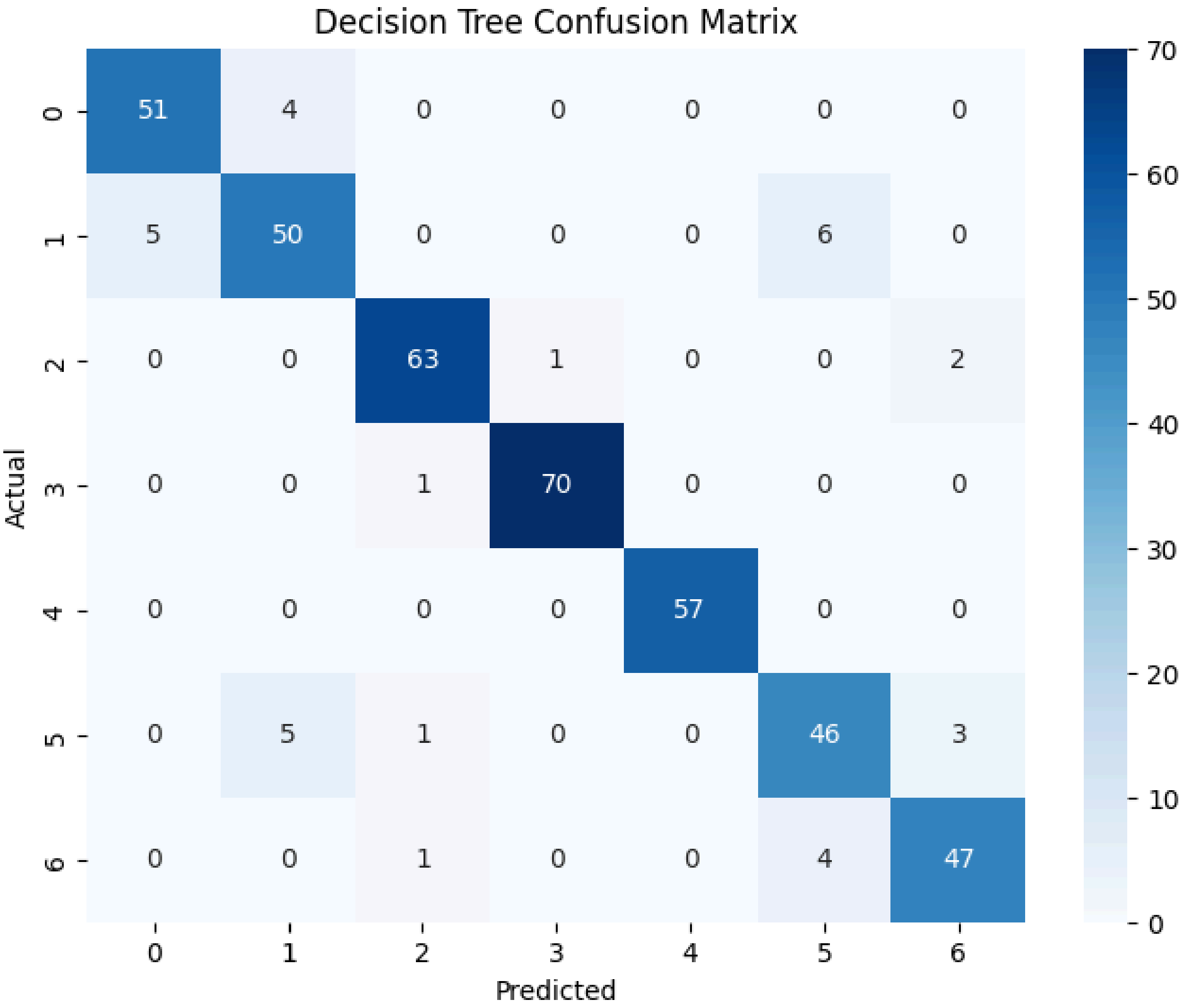
```
dt_classifier.fit(X_train,y_train)
y_pred=dt_classifier.predict(X_test)
accuracy=accuracy_score(y_test,y_pred)
accuracy
```

```
0.920863309352518
```

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred)
print("Decision Tree Confusion Matrix: \n",cm)
```

```
Decision Tree Confusion Matrix: 
 [[51  4  0  0  0  0  0]
 [ 5 50  0  0  0  6  0]
 [ 0  0 63  1  0  0  2]
 [ 0  0  1 70  0  0  0]
 [ 0  0  0  0 57  0  0]
 [ 0  5  1  0  0 46  3]
 [ 0  0  1  0  0  4 47]]
```

```
#Plot Confusion Matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Decision Tree Confusion Matrix')
plt.show()
```



```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.93      0.92        55
           1       0.85      0.82      0.83        61
           2       0.95      0.95      0.95        66
           3       0.99      0.99      0.99        71
           4       1.00      1.00      1.00        57
           5       0.82      0.84      0.83        55
           6       0.90      0.90      0.90        52

    accuracy                           0.92       417
   macro avg       0.92      0.92      0.92       417
weighted avg       0.92      0.92      0.92       417
```

Model 5 Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rf_classifier=RandomForestClassifier()
rf_classifier
```

```
▾ RandomForestClassifier ⓘ ?
RandomForestClassifier()
```

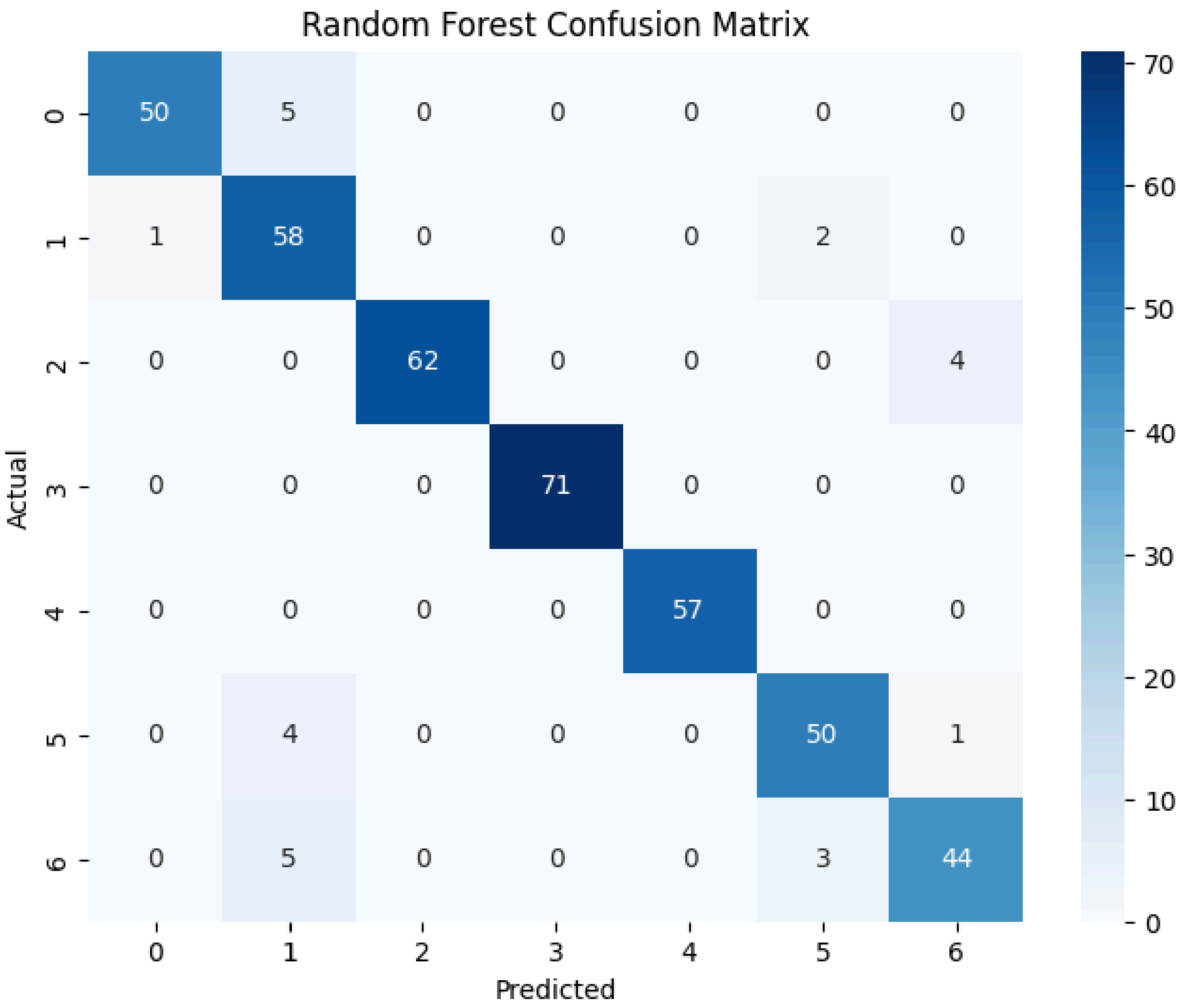
```
rf_classifier.fit(X_train,y_train)
y_pred=rf_classifier.predict(X_test)
accuracy=accuracy_score(y_test,y_pred)
accuracy
```

```
0.9400479616306955
```

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred)
print("Random Forest Confusion Matrix: \n",cm)
```

```
Random Forest Confusion Matrix:
 [[50  5  0  0  0  0  0]
 [ 1 58  0  0  0  2  0]
 [ 0  0 62  0  0  0  4]
 [ 0  0  0 71  0  0  0]
 [ 0  0  0  0 57  0  0]
 [ 0  4  0  0  0 50  1]
 [ 0  5  0  0  0  3 44]]
```

```
#Plot Confusion Matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Random Forest Confusion Matrix')
plt.show()
```



```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.91      0.94        55
           1       0.81      0.95      0.87        61
           2       1.00      0.94      0.97        66
           3       1.00      1.00      1.00        71
           4       1.00      1.00      1.00        57
           5       0.91      0.91      0.91        55
           6       0.90      0.85      0.87        52

    accuracy                           0.94       417
   macro avg       0.94      0.94      0.94       417
weighted avg       0.94      0.94      0.94       417
```

Model 6 K Neighbors

```
from sklearn.neighbors import KNeighborsClassifier

knn_classifier=KNeighborsClassifier()
knn_classifier
```

```
▾ KNeighborsClassifier ⓘ ?
KNeighborsClassifier()
```

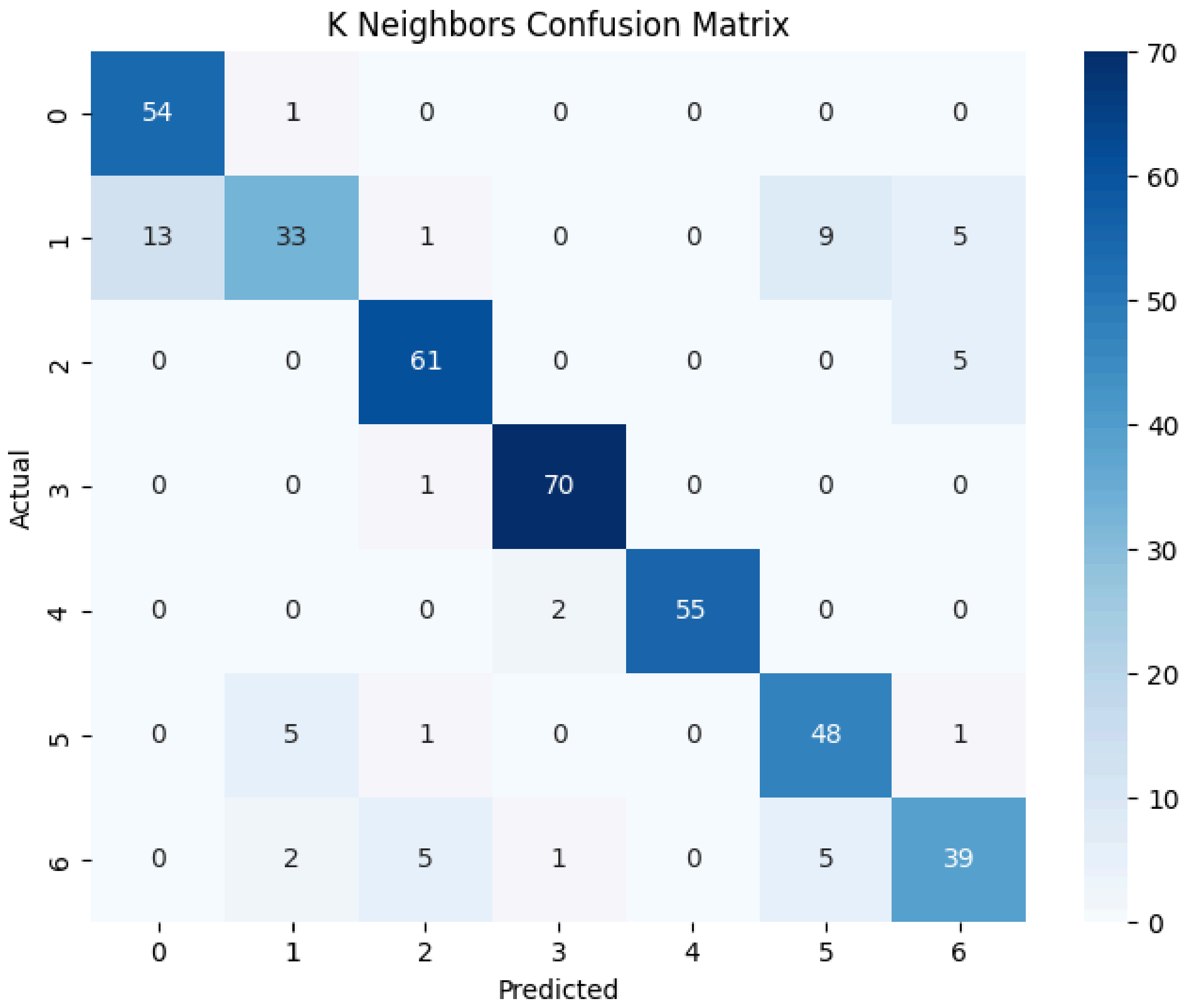
```
knn_classifier.fit(X_train,y_train)
y_pred=knn_classifier.predict(X_test)
accuracy=accuracy_score(y_test,y_pred)
accuracy
```

```
0.8633093525179856
```

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred)
print("K Neighbors Confusion Matrix: \n",cm)
```

```
K Neighbors Confusion Matrix: 
 [[54  1  0  0  0  0  0]
 [13 33  1  0  0  9  5]
 [ 0  0 61  0  0  0  5]
 [ 0  0  1 70  0  0  0]
 [ 0  0  0  2 55  0  0]
 [ 0  5  1  0  0 48  1]
 [ 0  2  5  1  0  5 39]]
```

```
#Plotting Confusion Matrix

plt.figure(figsize=(8,6))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('K Neighbors Confusion Matrix')
plt.show()
```



```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.98      0.89        55
           1       0.80      0.54      0.65        61
           2       0.88      0.92      0.90        66
           3       0.96      0.99      0.97        71
           4       1.00      0.96      0.98        57
           5       0.77      0.87      0.82        55
           6       0.78      0.75      0.76        52

    accuracy                           0.86       417
   macro avg       0.86      0.86      0.85       417
weighted avg       0.86      0.86      0.86       417
```

Model 7 PLA

```
from sklearn.linear_model import Perceptron

pla_classifier=Perceptron(max_iter=1000,tol=1e-3,random_state=42)
pla_classifier
```

```
▾      Perceptron      ⓘ ?
Perceptron(random_state=42)
```

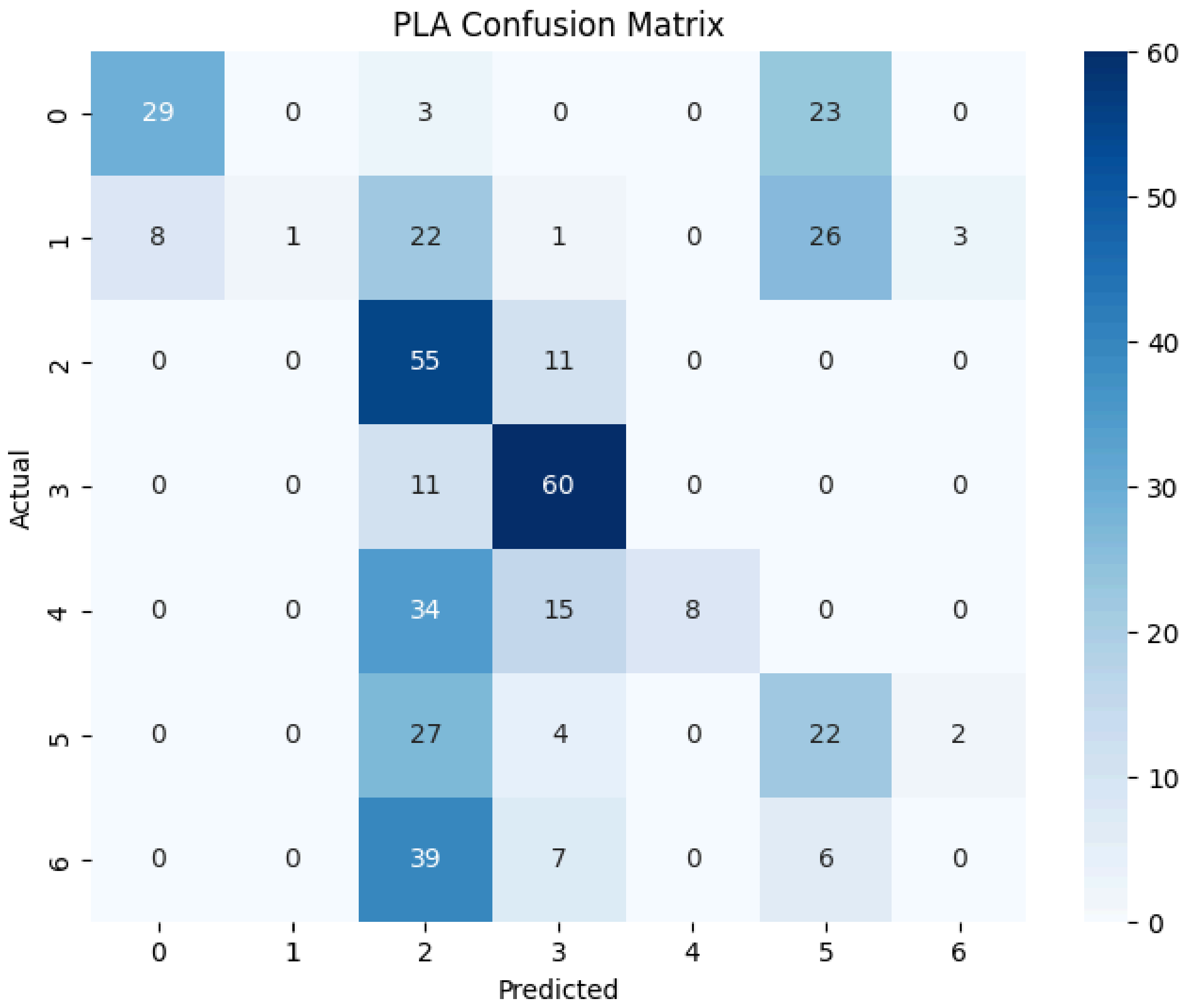
```
pla_classifier.fit(X_train,y_train)
y_pred=pla_classifier.predict(X_test)
accuracy=accuracy_score(y_test,y_pred)
accuracy
```

```
0.4196642685851319
```

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred)
print("PLA Matrix: \n",cm)
```

```
PLA Matrix: 
 [[29  0  3  0  0 23  0]
 [ 8  1 22  1  0 26  3]
 [ 0  0 55 11  0  0  0]
 [ 0  0 11 60  0  0  0]
 [ 0  0 34 15  8  0  0]
 [ 0  0 27  4  0 22  2]
 [ 0  0 39  7  0  6  0]]
```

```
#Plotting Confusion Matrix

plt.figure(figsize=(8,6))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('PLA Confusion Matrix')
plt.show()
```



```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.53      0.63        55
           1       1.00      0.02      0.03        61
           2       0.29      0.83      0.43        66
           3       0.61      0.85      0.71        71
           4       1.00      0.14      0.25        57
           5       0.29      0.40      0.33        55
           6       0.00      0.00      0.00        52

    accuracy                           0.42       417
   macro avg       0.57      0.39      0.34       417
weighted avg       0.57      0.42      0.35       417
```

Model 8 MLP

```
from sklearn.neural_network import MLPClassifier

mlp_classifier=MLPClassifier(hidden_layer_sizes=(100,),max_iter=300,random_state=42)
mlp_classifier
```

```
MLPClassifier

MLPClassifier(max_iter=300, random_state=42)
```

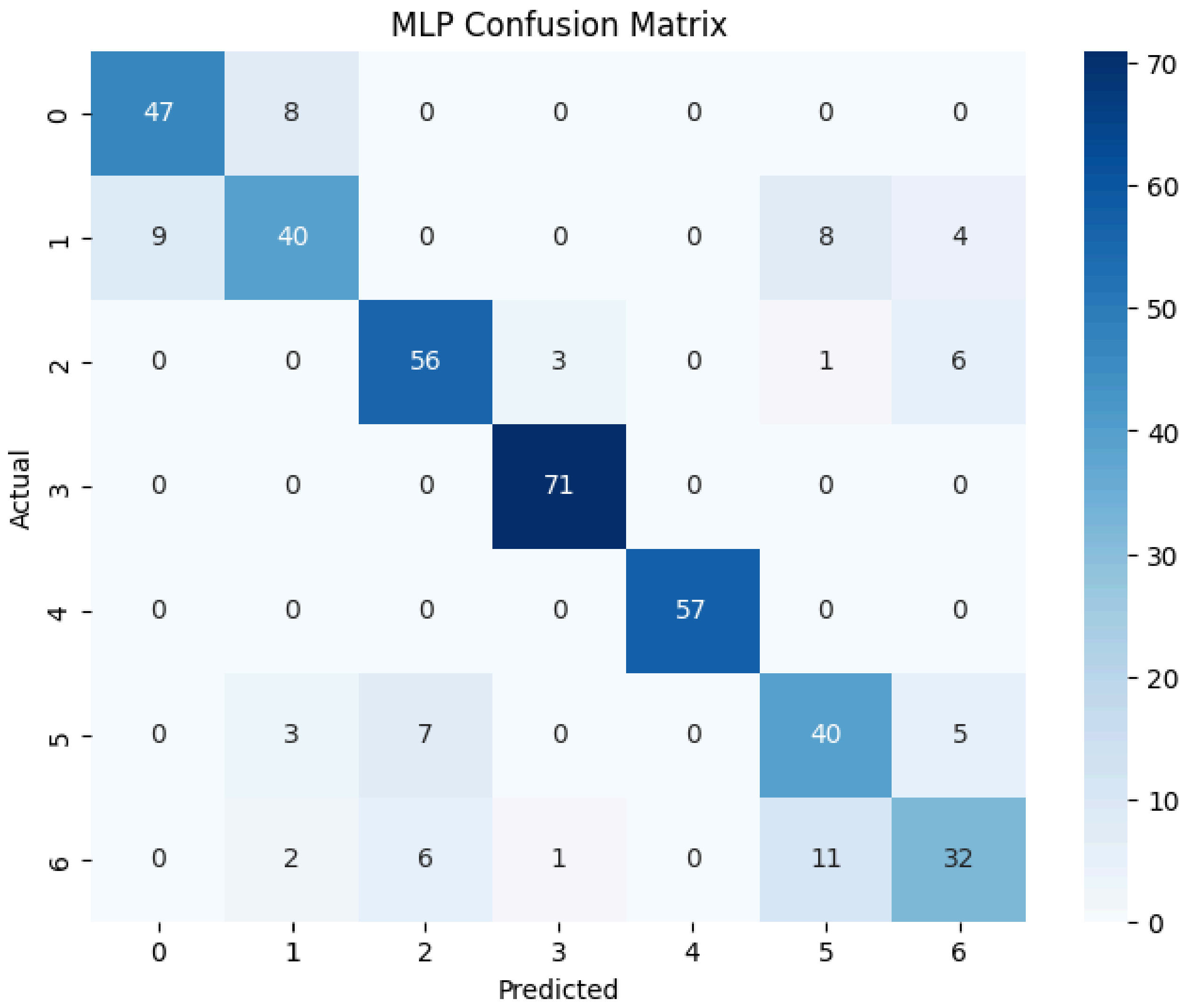
```
mlp_classifier.fit(X_train,y_train)
y_pred=mlp_classifier.predict(X_test)
accuracy=accuracy_score(y_test,y_pred)
accuracy
```

```
c:\Python310\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691:
  warnings.warn(
0.8225419664268585
```

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred)
print("MLP Matrix: \n",cm)
```

```
MLP Matrix: 
 [[47  8  0  0  0  0  0]
 [ 9 40  0  0  0  8  4]
 [ 0  0 56  3  0  1  6]
 [ 0  0  0 71  0  0  0]
 [ 0  0  0  0 57  0  0]
 [ 0  3  7  0  0 40  5]
 [ 0  2  6  1  0 11 32]]
```

```
#Plotting Confusion Matrix

plt.figure(figsize=(8,6))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('MLP Confusion Matrix')
plt.show()
```



	0	1	2	3	4	5	6
0	47	8	0	0	0	0	0
1	9	40	0	0	0	8	4
2	0	0	56	3	0	1	6
3	0	0	0	71	0	0	0
4	0	0	0	0	57	0	0
5	0	3	7	0	0	40	5
6	0	2	6	1	0	11	32

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.85      0.85        55
           1       0.75      0.66      0.70        61
           2       0.81      0.85      0.83        66
           3       0.95      1.00      0.97        71
           4       1.00      1.00      1.00        57
           5       0.67      0.73      0.70        55
           6       0.68      0.62      0.65        52

    accuracy                           0.82       417
   macro avg       0.81      0.81      0.81       417
weighted avg       0.82      0.82      0.82       417
```

Model 9 Naive Bayes

```
from sklearn.naive_bayes import GaussianNB

nb_classifier=GaussianNB()
nb_classifier
```

```
▾  GaussianNB ⓘ ?

GaussianNB()
```

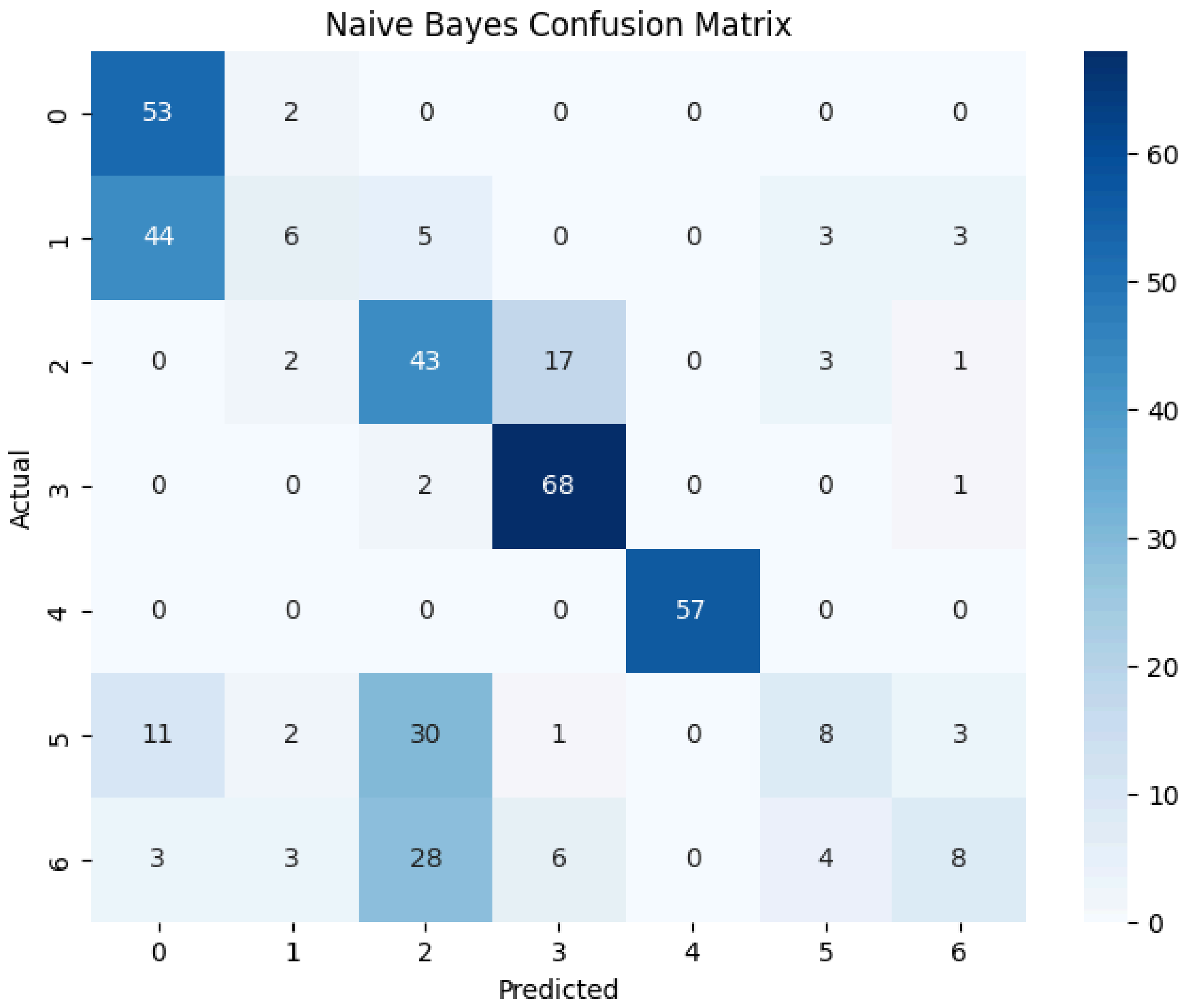
```
nb_classifier.fit(X_train,y_train)
y_pred=nb_classifier.predict(X_test)
accuracy=accuracy_score(y_test,y_pred)
accuracy
```

```
0.5827338129496403
```

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred)
print("Naive Bayes Matrix: \n",cm)
```

```
Naive Bayes Matrix: 
 [[53  2  0  0  0  0  0]
 [44  6  5  0  0  3  3]
 [ 0  2 43 17  0  3  1]
 [ 0  0  2 68  0  0  1]
 [ 0  0  0  0 57  0  0]
 [11  2 30  1  0  8  3]
 [ 3  3 28  6  0  4  8]]
```

```
#Plotting Confusion Matrix

plt.figure(figsize=(8,6))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Naive Bayes Confusion Matrix')
plt.show()
```



```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.48      0.96      0.64        55
           1       0.40      0.10      0.16        61
           2       0.40      0.65      0.49        66
           3       0.74      0.96      0.83        71
           4       1.00      1.00      1.00        57
           5       0.44      0.15      0.22        55
           6       0.50      0.15      0.24        52

    accuracy                           0.58       417
   macro avg       0.57      0.57      0.51       417
weighted avg       0.57      0.58      0.52       417
```

Model 10 KMeans Clustering

```
from sklearn.cluster import KMeans

kmeans=KMeans(n_clusters=3,random_state=42)
kmeans
```

```
▾ KMeans
KMeans(n_clusters=3, random_state=42)
```

```
kmeans.fit(X)
y_pred=kmeans.predict(X)

from sklearn.metrics import silhouette_score
silhouette_avg=silhouette_score(X,y_pred)
print("silhouette_score: ",silhouette_avg)
```

```
silhouette_score:  0.5050222435923701
```

```
plt.figure(figsize=(8,6))
plt.scatter(X.iloc[:,0],X.iloc[:,1],c=y_pred,s=50,cmap='viridis')
centers=kmeans.cluster_centers_
plt.scatter(centers[:,0],centers[:,1],c='red',s=200,alpha=0.75)
plt.title('KMeans Clustering')      #alpha=0.75 sets the transparency of the cluster cent
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Model 11 Dimensionality Reduction using PCA

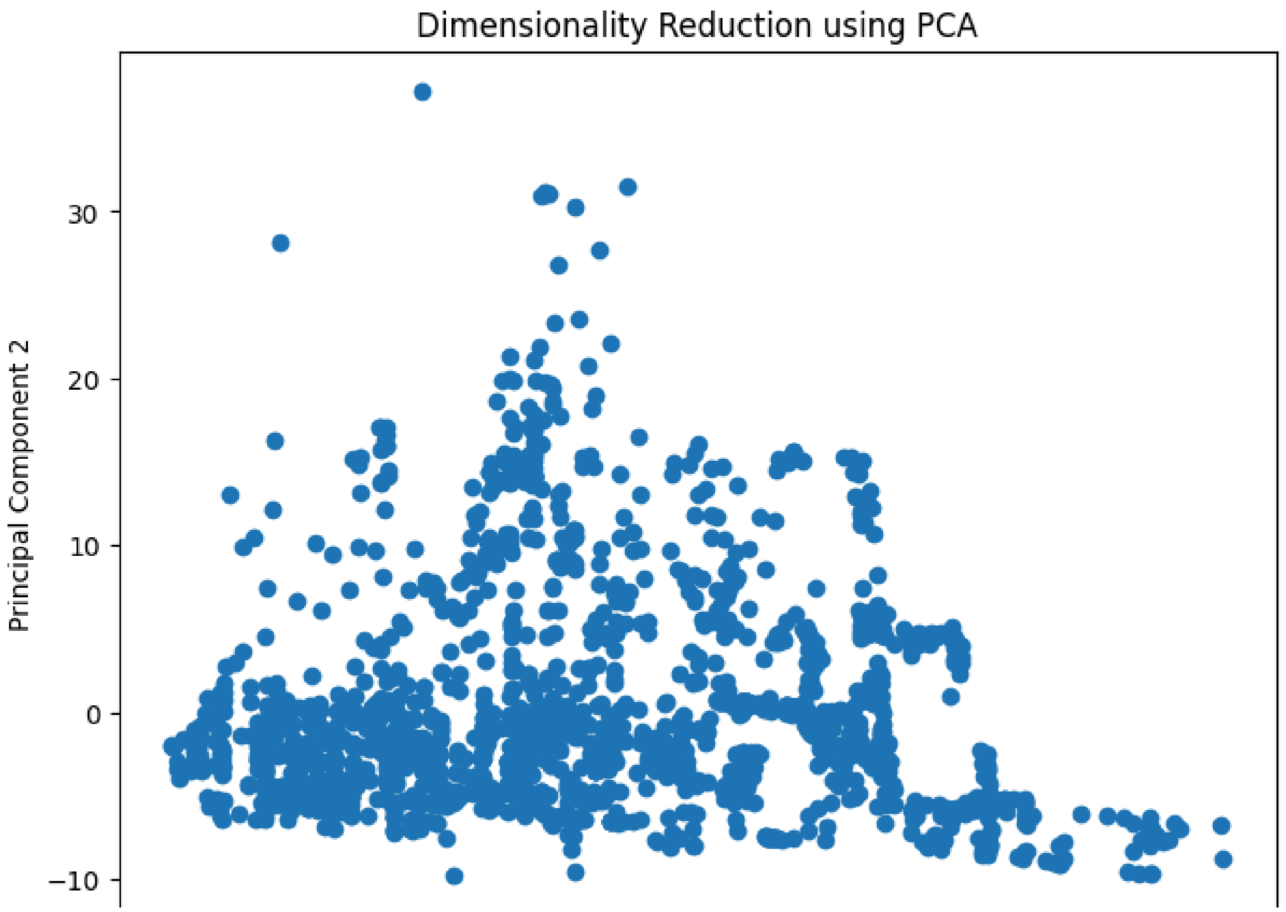
```
from sklearn.decomposition import PCA
pca=PCA(n_components=2)
pca
```

```
▾ PCA
PCA(n_components=2)
```

```
X_pca=pca.fit_transform(X)
```

```
plt.figure(figsize=(8,6))
plt.scatter(X_pca[:,0],X_pca[:,1],cmap='viridis')
plt.title('Dimensionality Reduction using PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

```
C:\Users\rohit\AppData\Local\Temp\ipykernel_16104\2795715780.py:2: UserWarning: No da
  plt.scatter(X_pca[:,0],X_pca[:,1],cmap='viridis')
```


Dimensionality Reduction using PCA
Principal Component 2