**Practical: 5**

**Aim: To Implement principal component analysis**

**Code:**

1. **Visualization of iris data set in 3d space**

from sklearn.datasets import load\_iris

import matplotlib.pyplot as plt

import mpl\_toolkits.mplot3d.axes3d as p3

iris\_data = load\_iris()

x = iris\_data.data

y = iris\_data.target

fig = plt.figure(figsize=(8, 6))

ax = p3.Axes3D(fig, elev=-150, azim=110)

ax.set\_xlabel('Sepal length')

ax.set\_ylabel('Sepal width')

ax.set\_zlabel('Petal length')

for name, label in [('Setosa', 0), ('Versicolour', 1), ('Virginica', 2)]:

ax.text3D(x[y == label, 0].mean(),

x[y == label, 1].mean() + 1.5,

x[y == label, 2].mean(), name,

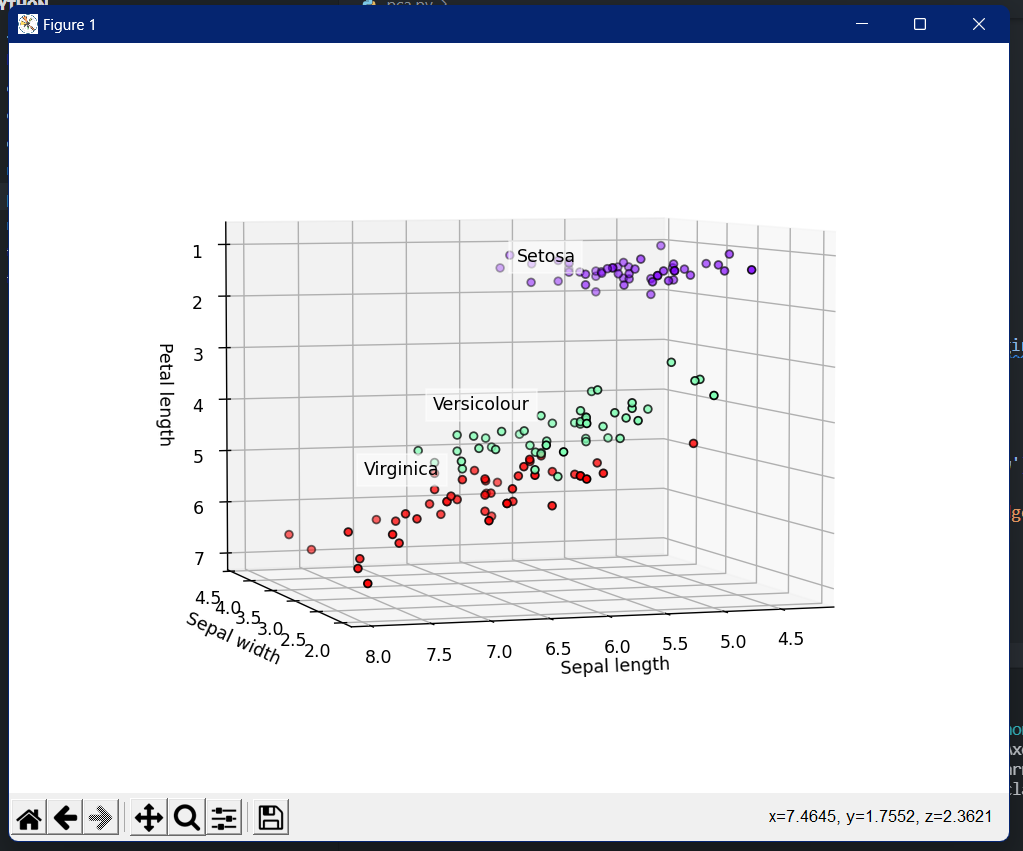
horizontalalignment='center',

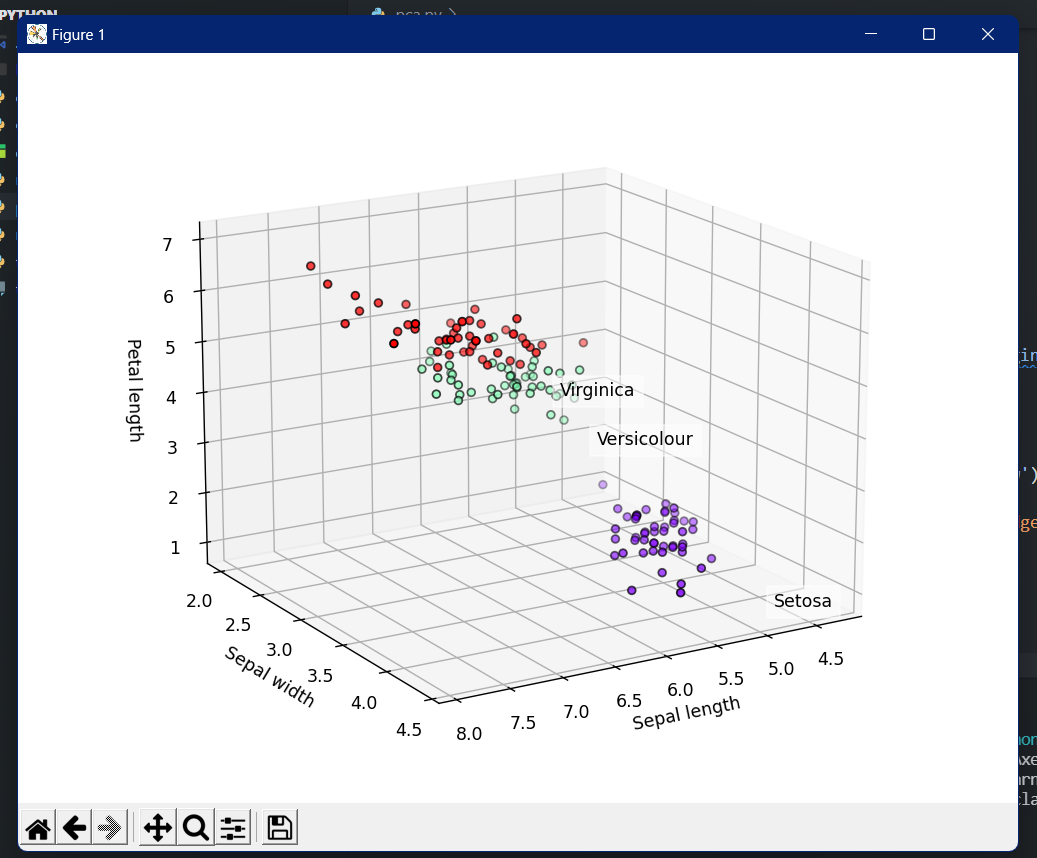
bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))

ax.scatter(x[:, 0], x[:, 1], x[:, 2], c=y, cmap='rainbow', edgecolor='k')

plt.show()

**Output:**

****

****

1. **Applying principal component analysis**

**Code:**

# PCA starts here

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix

from matplotlib.colors import ListedColormap

import numpy as np

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x,

y,

test\_size=0.2,

random\_state=0)

sc = StandardScaler()

# Scale the data

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Apply PCA

pca = PCA(n\_components=2)

X\_train = pca.fit\_transform(X\_train)

X\_test = pca.transform(X\_test)

explained\_variance = pca.explained\_variance\_ratio\_

# Apply Logistic Regression

classifier = LogisticRegression(random\_state=0)

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

# Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred)

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(

np.arange(start=X\_set[:, 0].min() - 1,

stop=X\_set[:, 0].max() + 1,

step=0.01),

np.arange(start=X\_set[:, 1].min() - 1,

stop=X\_set[:, 1].max() + 1,

step=0.01))

# Plot the decision boundary

plt.contourf(X1,

X2,

classifier.predict(np.array([X1.ravel(),

X2.ravel()]).T).reshape(X1.shape),

alpha=0.75,

cmap=ListedColormap(('white', 'yellow', 'pink')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

# Plot the training points

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0],

X\_set[y\_set == j, 1],

c=ListedColormap(('red', 'green', 'blue'))(i),

label=j)

plt.title('Logistic Regression (Training set)')

plt.xlabel('PC1')

plt.ylabel('PC2')

plt.legend()

# show scatter plot

plt.show()

# Test set visualization

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(

np.arange(start=X\_set[:, 0].min() - 1,

stop=X\_set[:, 0].max() + 1,

step=0.01),

np.arange(start=X\_set[:, 1].min() - 1,

stop=X\_set[:, 1].max() + 1,

step=0.01))

plt.contourf(X1,

X2,

classifier.predict(np.array([X1.ravel(),

X2.ravel()]).T).reshape(X1.shape),

alpha=0.75,

cmap=ListedColormap(('yellow', 'white', 'aquamarine')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0],

X\_set[y\_set == j, 1],

c=ListedColormap(('red', 'green', 'blue'))(i),

label=j)

# title for scatter plot

plt.title('Logistic Regression (Test set)')

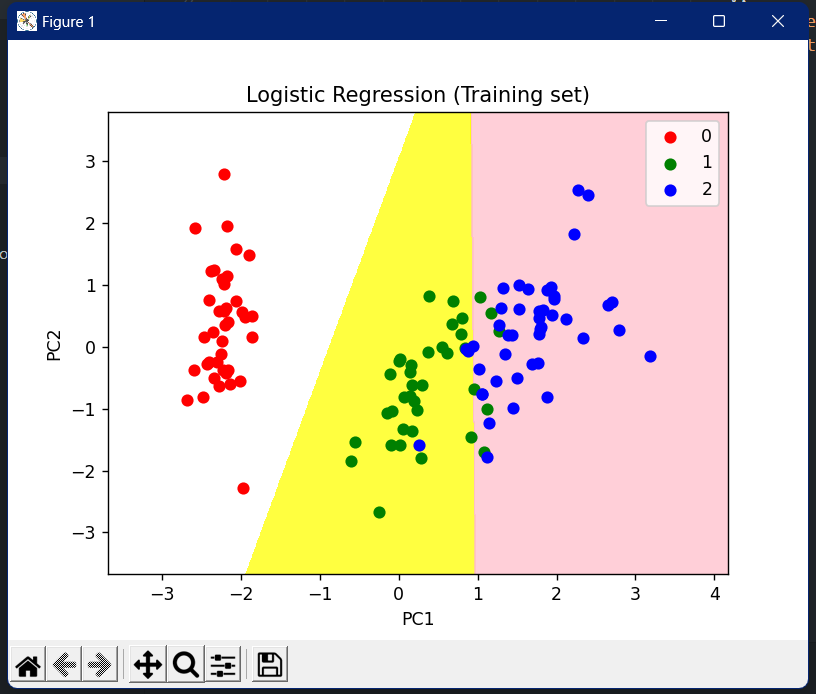
plt.xlabel('PC1')

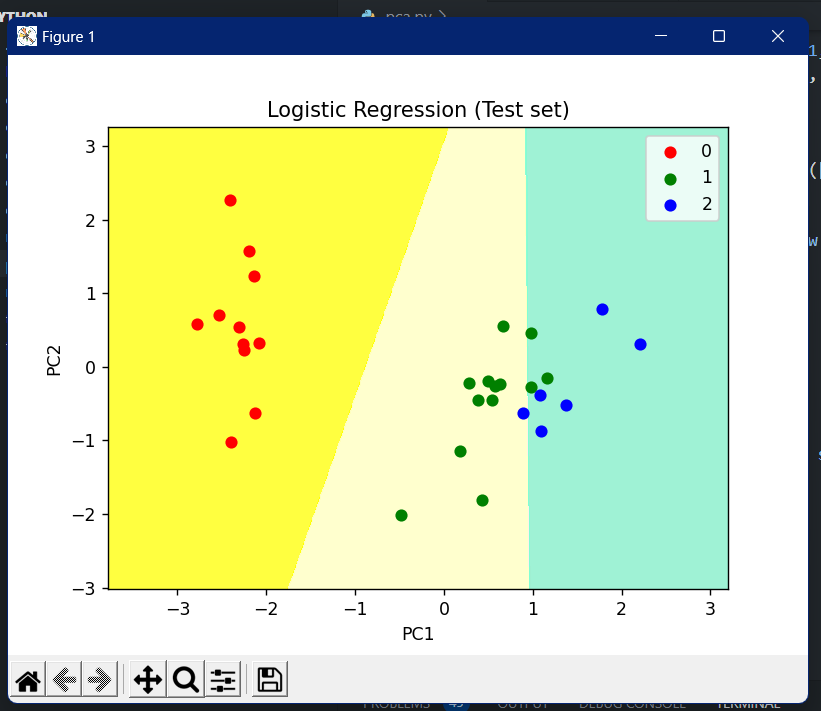
plt.ylabel('PC2')

plt.legend()

plt.show()

**# Ouput**

****

****