Project 1 Report

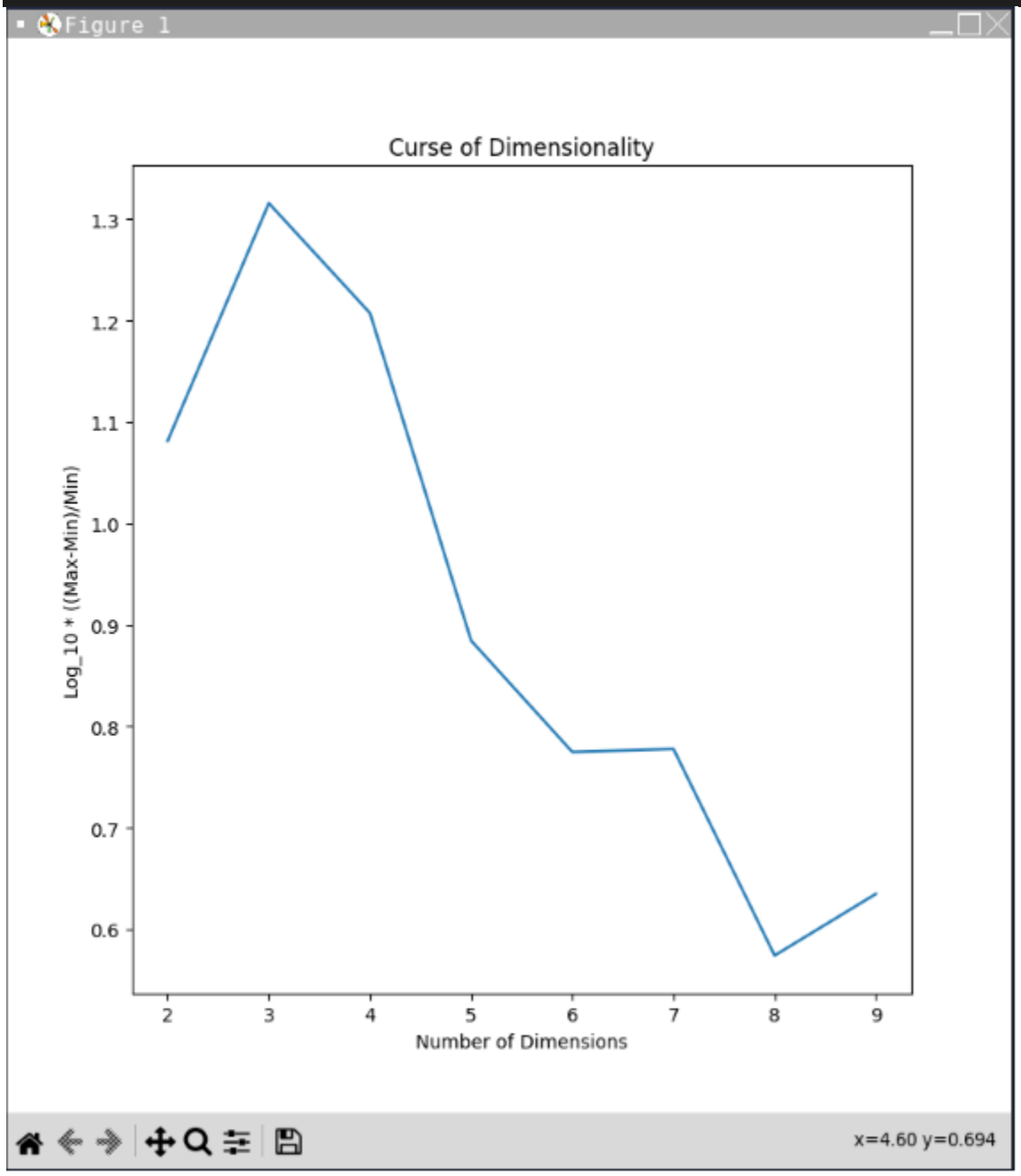
Grant Davis
CS458
09/27/2022

P1-1. Curse of Dimensionality
(a) Generate 1000 points following a uniform distribution under a given dimension, and then compute difference between max and min distance between any pair of points. Hint: Refer to the tutorial "Introduction to Numpy and Pandas" on how to generate random points.
(b) Repeat (a) for different dimensions from 2 to 50.

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.spatial import ConvexHull
from scipy.spatial.distance import cdist
from progressbar import progressbar

NUM_POINTS = 50
MAX_ITERATIONS = 10

##Functions
#Find Min/Max points Function
def findMinMax(p):
  #Extraction of points forming the convex hull
  hull = ConvexHull(p)
  hullpoints = p[hull.vertices,:]
  hdist = cdist(hullpoints, hullpoints, metric='euclidean')

  #Min/Max points
  maxpair = np.unravel_index(hdist.argmax(), hdist.shape)
  i,j = np.where(hdist==np.min(hdist[np.nonzero(hdist)]))

  pairPoints = np.array([(np.linalg.norm(hullpoints[i][0] -
hullpoints[i][1])),(np.linalg.norm(hullpoints[maxpair[0]] -
hullpoints[maxpair[1]]))])
```

```
  logDiff =
np.log10((pairPoints[1]-pairPoints[0])/pairPoints[0])
  print(logDiff)
  logDiffs.append(logDiff)

#Curse of Dimensionality Graph Function
def curseGraph():
  x = np.arange(2,MAX_ITERATIONS)
  y = logDiffs
  plt.title("Curse of Dimensionality")
  plt.xlabel("Number of Dimensions")
  plt.ylabel("Log_10 * ((Max-Min)/Min)")
  plt.plot(x,y)
  plt.show()

logDiffs = []

##Run Curse of Dimensionality
for x in range(2,MAX_ITERATIONS):
  print(f"({x}/{MAX_ITERATIONS}): ", end='')
  findMinMax(np.random.rand(NUM_POINTS,x))

curseGraph()
(2/10): 1.0814941798021713

(3/10): 1.3156547989618175

(4/10): 1.2073302289966508

(5/10): 0.8844795227092559

(6/10): 0.7747844527933487

(7/10): 0.7777922517130524
```

```
(8/10): 0.5741849693594667

(9/10): 0.6348518949926998
```



When generating a large number of points with more than 10 dimensions the time to complete the process became exponentially longer. I could show the Curse of dimensionality with fewer points for a similar number of dimensions.

P1-2. The Iris Dataset

(a) Data Visualization. Duplicate the following figure using scatter plot.

```
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
import numpy as np
import sys
import subprocess
subprocess.check_call([sys.executable, '-m', 'pip', 'install',
'sklearn'])
from sklearn import datasets
from sklearn.cluster import KMeans

NUM_PLOTS = 4

#Generate 16 Iris plots
def plots():
  for v in range(NUM_PLOTS):
    for h in range(NUM_PLOTS):
      if (v != h):
        axs[v][h].scatter(x[:, h], x[:, v], s=7, c=y,
cmap=plt.cm.brg, edgecolor='k', linewidth=0.5)
      else:
        axs[v][h].text(0.5, 0.5, TitleList[v],
horizontalalignment='center', verticalalignment='center',
clip_on=True)
        axs[v][h].xaxis.set_visible(False)
        axs[v][h].yaxis.set_visible(False)

#import iris dataset
iris = datasets.load_iris()
x = iris.data
y = iris.target

MarkerList = ['s', 'o', 'd']
TitleList = ["Sepal Length", "Sepal Width", "Petal Length",
"Petal Width"]
```

```
#Figure 1
fig, axs = plt.subplots(NUM_PLOTS, NUM_PLOTS)
fig.suptitle("Iris Data (blue=setosa, red=versicolor,
green=virginica)")
#Figure 2
fig2, axs2 = plt.subplots()

plots()
plt.show()
```

(b) Find the best discretization for the petal length and the petal width that can best separate the Iris data and plot a figure similar to the figure in slide 54 in Chapter 2. For each flower type, list in a table how many data samples are correctly separated and how many are not correctly separated.

```
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
import numpy as np
import sys
import subprocess
subprocess.check_call([sys.executable, '-m', 'pip', 'install',
'sklearn'])
from sklearn import datasets
from sklearn.cluster import KMeans

NUM_PLOTS = 4

#Generate 16 Iris plots
def plots():
  for v in range(NUM_PLOTS):
    for h in range(NUM_PLOTS):
      if (v != h):
        axs[v][h].scatter(x[:, h], x[:, v], s=7, c=y,
cmap=plt.cm.brg, edgecolor='k', linewidth=0.5)
      else:
```

```
        axs[v][h].text(0.5, 0.5, TitleList[v],
horizontalalignment='center', verticalalignment='center',
clip_on=True)
        axs[v][h].xaxis.set_visible(False)
        axs[v][h].yaxis.set_visible(False)

def discretization():
  #Scatter var
  IrisLabels = ['Setosa', 'Versicolor', 'Virginica']
  for i in range(np.prod(y.shape)):
    axs2.scatter(x[i, 2], x[i, 3], marker=MarkerList[int(i/50)],
c='k', s=10.0)
  for l in range(len(IrisLabels)):
    axs2.scatter([],[],color='k', label=IrisLabels[l],
marker=MarkerList[l])
  axs2.legend()

  #Delete unnecessary var
  ClusterArray = np.delete(x, [0,1], 1)

  #Generate clusters
  axs2.set_xlabel("Petal Length (cm)")
  axs2.set_ylabel("Petal Width (cm)")
  km = KMeans(n_clusters=3)
  km.fit(ClusterArray)

  #Find centroids and plot them
  km_cntr = km.cluster_centers_
  axs2.scatter(km_cntr[:, 0], km_cntr[:, 1], c='red', s=15.0)

  #Create rectangles
  CentroidRectangleLengths = np.empty((0,2))
  for i in range(0,101,50):
    try:
```

```
      val_min_x, val_max_x =
np.min(np.delete(ClusterArray[i:(i+50)], 1, 1), axis=0),
np.max(np.delete(ClusterArray[i:(i+50)], 1, 1), axis=0)
      val_min_y, val_max_y =
np.min(np.delete(ClusterArray[i:(i+50)], 0, 1), axis=0),
np.max(np.delete(ClusterArray[i:(i+50)], 0, 1), axis=0)
      x_len, y_len = val_max_x-val_min_x, val_max_y-val_min_y
      CentroidRectangleLengths =
np.append(CentroidRectangleLengths, [np.concatenate((x_len,
y_len))], axis=0)
    except ValueError:
      pass

  #Plot rectangles
  index=0
  sortedCenters = np.sort(km_cntr, axis=0)
  for c in (sortedCenters):
    width, height = CentroidRectangleLengths[index][0],
CentroidRectangleLengths[index][1]
    axs2.add_patch(Rectangle(
      xy=(c[0]-width/2, c[1]-height/2), width=width,
height=height, linewidth=1, color='black', fill=False))
    index += 1

#import iris dataset
iris = datasets.load_iris()
x = iris.data
y = iris.target

MarkerList = ['s', 'o', 'd']
TitleList = ["Sepal Length", "Sepal Width", "Petal Length",
"Petal Width"]

#Figure 1
fig, axs = plt.subplots(NUM_PLOTS, NUM_PLOTS)
```

```
fig.suptitle("Iris Data (blue=setosa, red=versicolor,
green=virginica)")
#Figure 2
fig2, axs2 = plt.subplots()

plots()
discretization()
plt.show()
```

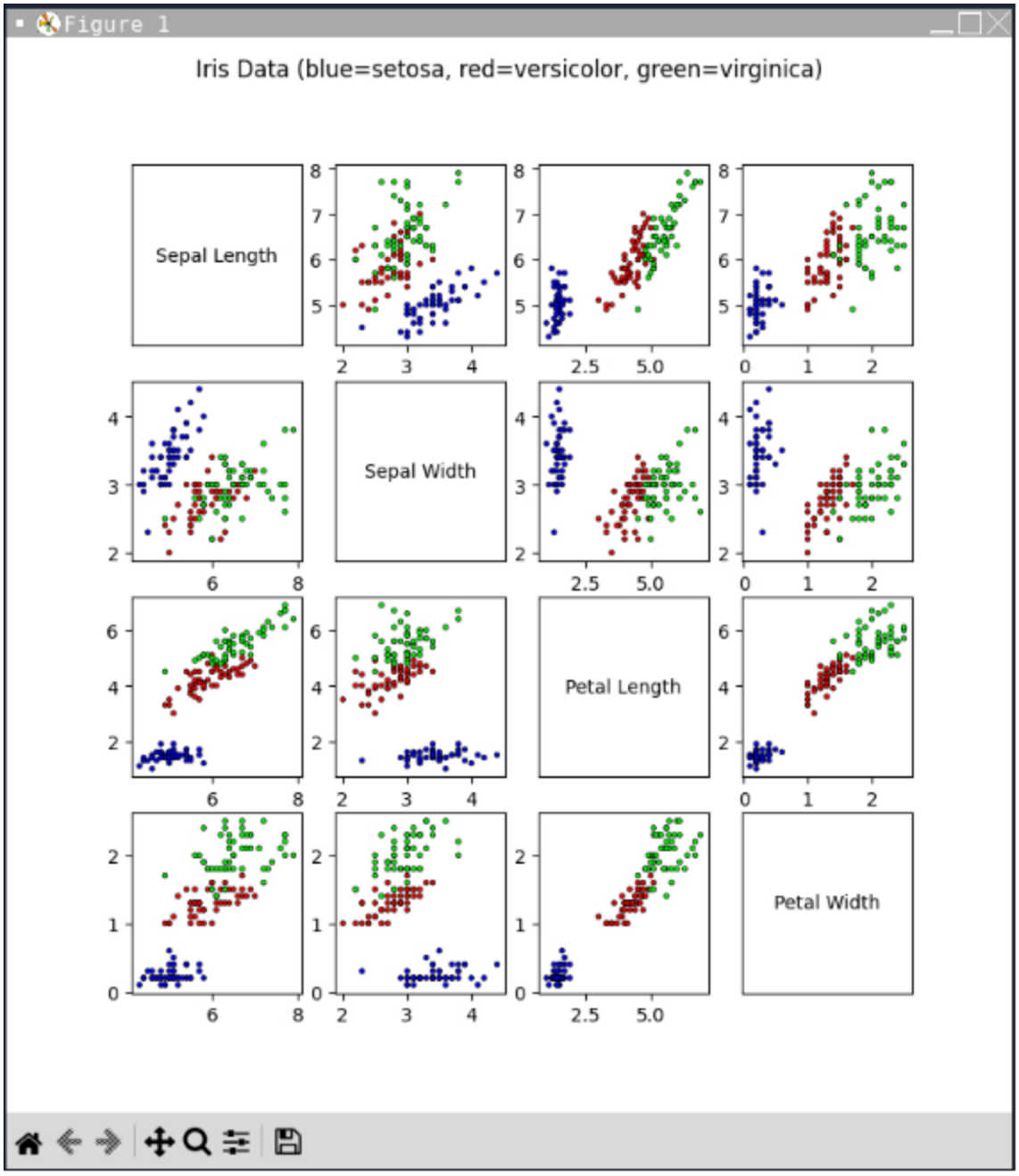
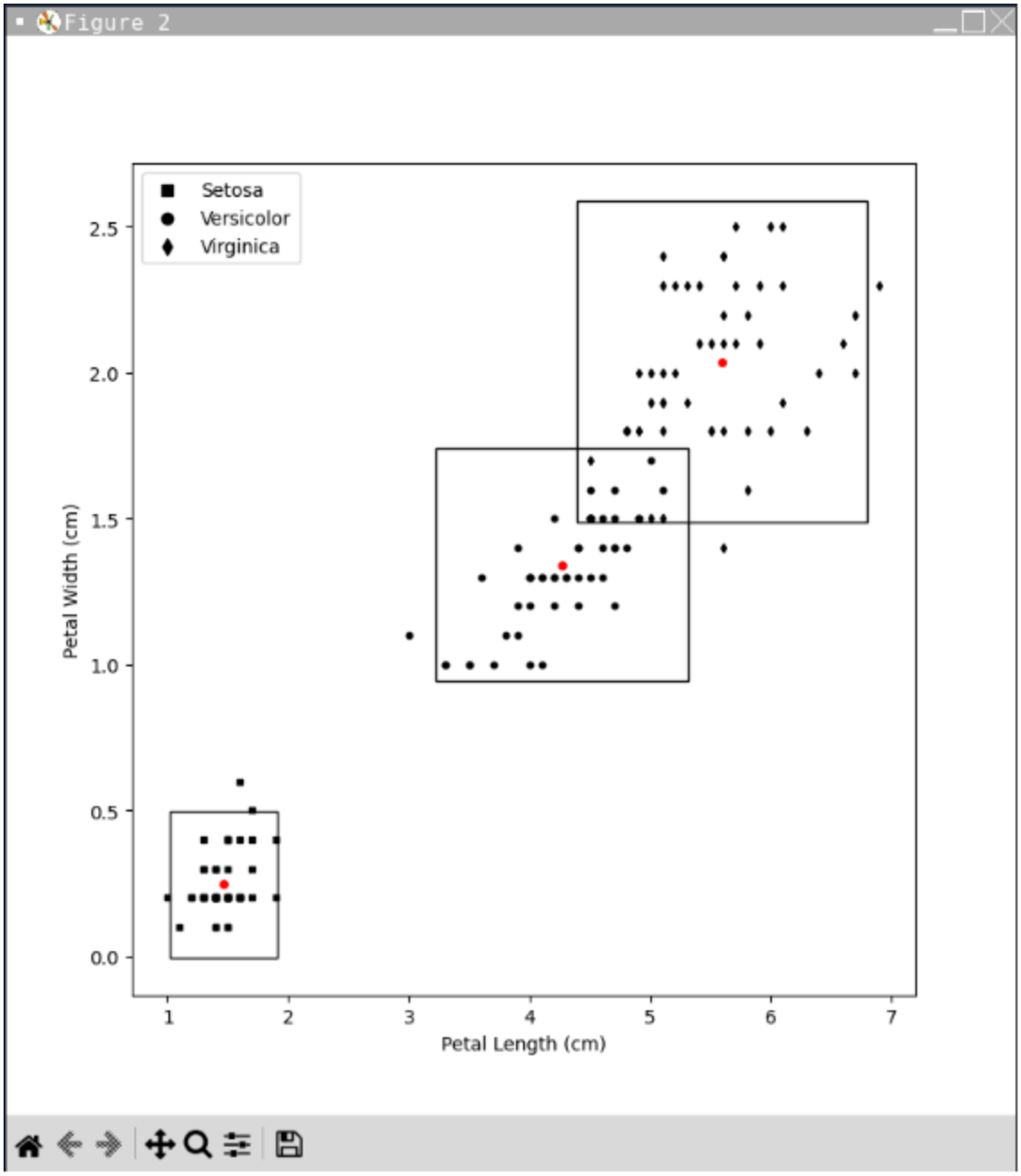


Figure 1
Iris Data (blue=setosa, red=versicolor, green=virginica)
Sepal Length
Sepal Width
Petal Length
Petal Width



Each cluster is plotted naively as each flower has 50 points each. The markers are spaced by 50.
The flower types that are correctly spaced are 50/50 for each group. The rectangles, which some overlap, are not correct.
Setosa had 48 correct and 2 incorrect, Versicolor had 49 correct and 1 incorrect, and Virginica had 48 correct and 2 incorrect.

P1-3. Principal Component Analysis for The Iris Dataset

(a) Use the Iris dataset and plot all the samples in a figure using Sepal Length and Sepal Width, i.e., xlabel('Sepal length') and ylabel('Sepal width').

```
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
import numpy as np
import sys
import subprocess
subprocess.check_call([sys.executable, '-m', 'pip', 'install',
'sklearn'])
from sklearn.decomposition import PCA
from sklearn import datasets

##Functions
#No Decomposition
def basePlot():
  axs.scatter(x[:, 0], x[:, 1], c=y, cmap=plt.cm.brg)
  axs.set_xlabel("Sepal Length (cm)")
  axs.set_ylabel("Sepal width (cm)")

#Import Iris Dataset
iris = datasets.load_iris()
x = iris.data
y = iris.target
IrisLabels = ['Versicolor', 'Setosa', 'Virginica']
PlotColors = ['r', 'b', 'g']

fig, axs = plt.subplots()
fig2, axs2 = plt.subplots()

basePlot()

leg = []
for l in range(len(IrisLabels)):
  leg.append(Rectangle((0,0),1,1,fc=PlotColors[l]))
axs.legend(leg, IrisLabels)
```

```
axs2.legend(leg, IrisLabels)
plt.show()
```

(b)The Iris dataset has 4 attributes (sepal length, sepal width, petal length, and petal width). Use PCA to reduce the dimension of the dataset from 4 to 2. Plot all the samples after the dimensionality reduction in a 2D figure. Compare this figure with the figure in (a) and discuss whether you can better separate the data samples after the dimensionality reduction.

```
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
import numpy as np
import sys
import subprocess
subprocess.check_call([sys.executable, '-m', 'pip', 'install',
'sklearn'])
from sklearn.decomposition import PCA
from sklearn import datasets

##Functions
#Decompostion
def PCAPlot():
  pca_iris = PCA(3)
  pca_iris.fit(x)
  decomp_x = pca_iris.transform(x)
  axs2.scatter(decomp_x[:,0], decomp_x[:,1], c=y,
cmap=plt.cm.brg)
  axs2.set_xlabel("Sepal Length (cm)")
  axs2.set_ylabel("Sepal width (cm)")

#No Decomposition
def basePlot():
  axs.scatter(x[:, 0], x[:, 1], c=y, cmap=plt.cm.brg)
  axs.set_xlabel("Sepal Length (cm)")
  axs.set_ylabel("Sepal width (cm)")
```

```
#Import Iris Dataset
iris = datasets.load_iris()
x = iris.data
y = iris.target
IrisLabels = ['Versicolor', 'Setosa', 'Virginica']
PlotColors = ['r', 'b', 'g']

fig, axs = plt.subplots()
fig2, axs2 = plt.subplots()

basePlot()
PCAPlot()

leg = []
for l in range(len(IrisLabels)):
  leg.append(Rectangle((0,0),1,1,fc=PlotColors[l]))
axs.legend(leg, IrisLabels)
axs2.legend(leg, IrisLabels)
plt.show()
```

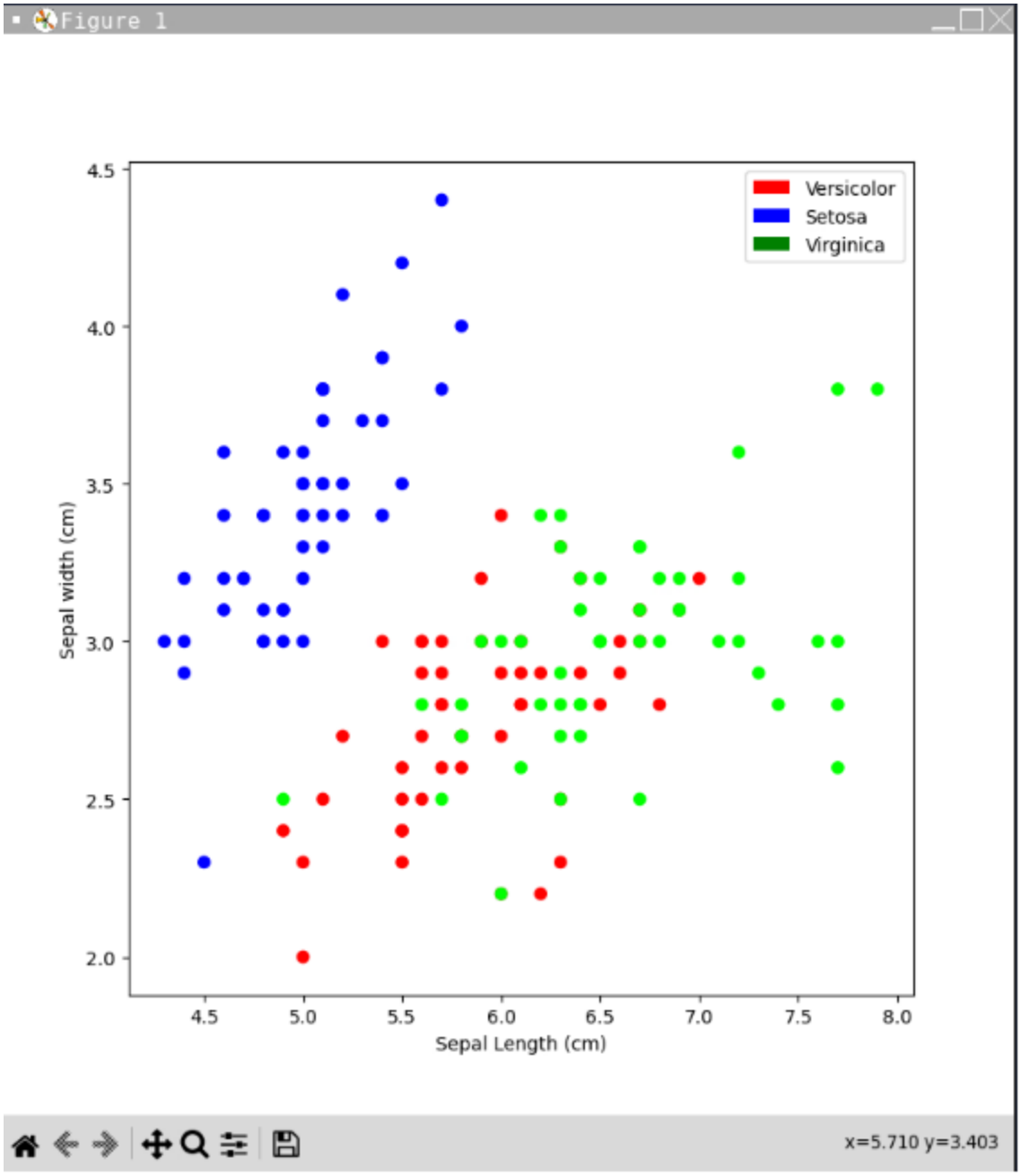
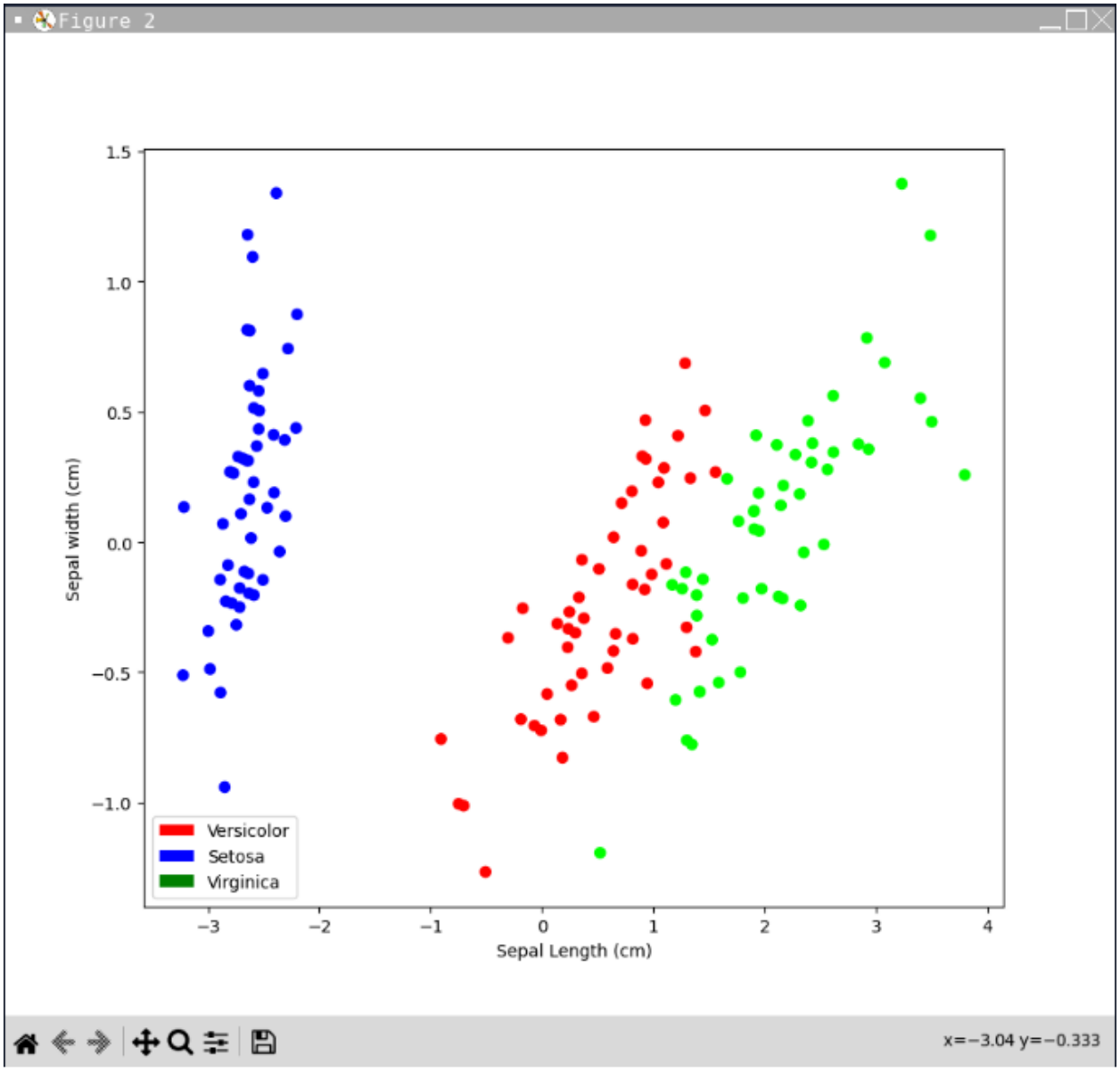


Figure 1
Versicolor
Setosa
Virginica
Sepal width (cm)
Sepal Length (cm)
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
x=5.710 y=3.403



After dimensionality reduction, the data samples are easily separated. Veriscolor and Virginica flower samples are still about the same, while Setosa flower has a distinct separation compared to the others.