

VIETNAMESE - GERMAN UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEER



Lab 2 Report

Student: Le Thanh Hai - 10421016

Student: Trinh The Hao - 10421017

Student: Vu Ngoc Quang - 10421103

1 Exercise 2.1

We download dataset from this page: "<https://archive-beta.ics.uci.edu/dataset/53/iris>"

Explanation:

The Iris dataset is a popular machine learning dataset that contains measurements of various features of three species of iris flowers: Iris setosa, Iris versicolor, and Iris virginica. It contains four features or attributes: sepal length, sepal width, petal length, and petal width. Each sample in the dataset represents an individual flower, and the measurements of these attributes are provided for each flower.

```
1 import numpy as np
2
3 # load iris from numpy
4 iris = np.genfromtxt('iris.data', delimiter=',', encoding=None)
5 iris = np.array([list(x) for x in iris])
6
7 # divide into features and labels
8 X = iris[:, 0:4]
9 y = iris[:, 4]
```

2 Exercise 2.2 & 2.3

Compute and report eight descriptive metrics of IRIS for all the flower types

```
1 max = np.max(X,axis=0) # max
2 min = np.min(X,axis=0) # min
3 mean = np.mean(X,axis=0) # mean
4 std = np.std(X,axis=0) # standard deviation
5 var = np.var(X,axis=0) # variance
6 median = np.median(X,axis=0) # median
7 q1 = np.percentile(X, 25,axis=0) # 25% percentile
8 q2 = np.percentile(X, 75,axis=0) # 75% percentile
```

Explanation:

The maximum value of each feature in the dataset is stored in the max array.

The minimum value of each feature in the dataset is stored in the min array.

The mean value of each feature in the dataset is stored in the mean array.

The standard deviation of each feature in the dataset is stored in the std array.

The variance of each feature in the dataset is stored in the var array.

The median value of each feature in the dataset is stored in the median array.

The 25th percentile value of each feature in the dataset is stored in the q1 array.

The 75th percentile value of each feature in the dataset is stored in the q2 array.

These statistics can provide useful insights into the distribution of the data. For example, the range of values for each feature can be determined from the max and min arrays, while the spread of the data can be measured using the std and var arrays. The median and percentiles can provide information about the central tendency and spread of the data as well.

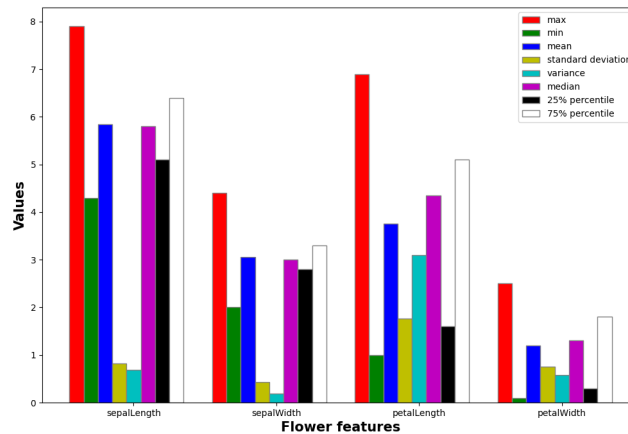


Figure 1: *Features for all flower types*

3 Exercise 2.4 & 2.5

Compute and report eight descriptive metrics of IRIS for each flower type.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 # load iris from numpy
6 iris = np.genfromtxt('iris.data', delimiter=',', dtype=None, encoding=↵
    None)
7 iris = np.array([list(x) for x in iris])
8
9 features = iris[:,0:4]
10 features = features.astype(np.float64)
11
12 setosa = features[0:50]
13 versicolor = features[50:100]
14 virginica = features[100:150]
15
16 labels = ["setosa", "versicolor", "virginica"]
17 flowers = [setosa, versicolor, virginica]
18

```

```
19 def compute(flower, index):
20     #max
21     max = np.max(flower, axis=0)
22     print("Max for all flower type: ", max)
23
24     #min
25     min = np.min(flower, axis=0)
26     print("Min for all flower type: ", min)
27
28     # mean
29     mean = np.mean(flower, axis=0)
30     print("Mean for all flower type: ", mean)
31
32     # standard deviation
33     std = np.std(flower, axis=0)
34     print("Standard deviation for all flower type: ", std)
35
36     # variance
37     var = np.var(flower, axis=0)
38     print("Variance for all flower type: ", var)
39
40     # median
41     median = np.median(flower, axis=0)
42     print("Median for all flower type: ", median)
43
44     # 25% percentile
45     q1 = np.percentile(flower, 25, axis=0)
46     print("25% percentile for all flower type: ", q1)
47
48
49     # 75% percentile
50     q2 = np.percentile(flower, 75, axis=0)
51     print("75% percentile for all flower type: ", q2)
52
53     # set width of bar
54     barWidth = 0.1
55     # plt.subplots(figsize =(12, 8))
56     plt.subplot(1, 3, i+1)
57
58     # set position of bar on X axis
59     br1 = np.arange(len(max))
60     br2 = [x + barWidth for x in br1]
61     br3 = [x + barWidth for x in br2]
62     br4 = [x + barWidth for x in br3]
63     br5 = [x + barWidth for x in br4]
```

```
64     br6 = [x + barWidth for x in br5]
65     br7 = [x + barWidth for x in br6]
66     br8 = [x + barWidth for x in br7]
67
68     # Make the plot
69     plt.bar(br1, max, color='r', width = barWidth,
70             edgecolor='grey', label='max')
71     plt.bar(br2, min, color='g', width = barWidth,
72             edgecolor='grey', label='min')
73     plt.bar(br3, mean, color='b', width = barWidth,
74             edgecolor='grey', label='mean')
75     plt.bar(br4, std, color='y', width = barWidth,
76             edgecolor='grey', label='standard deviation')
77     plt.bar(br5, var, color='c', width = barWidth,
78             edgecolor='grey', label='variance')
79     plt.bar(br6, median, color='m', width = barWidth,
80             edgecolor='grey', label='median')
81     plt.bar(br7, q1, color='k', width = barWidth,
82             edgecolor='grey', label='25% percentile')
83     plt.bar(br8, q2, color='w', width = barWidth,
84             edgecolor='grey', label='75% percentile')
85
86     # Adding Xticks
87     plt.xlabel(labels[i], fontweight='bold', fontsize = 10)
88     plt.ylabel('Values', fontweight='bold', fontsize = 10)
89     plt.xticks([r + 0.4 for r in range(len(max))],
90               ['sepalLength', 'sepalWidth', 'petalLength', 'petalWidth'←
91               ])
92
93     plt.legend()
94
95     fig = plt.figure(figsize=(19,10))
96     SMALL_SIZE = 7
97     plt.rc('font', size=SMALL_SIZE)
98     for i in range(len(flowers)):
99         compute(flowers[i],i)
100     plt.show()
```

Explanation:

Data Loading: The Iris dataset is loaded from the 'iris.data' file using NumPy's `genfromtxt()` function.

Data Preparation: The features are extracted from the dataset and converted to float64 data type.

Data Segmentation: The features are segmented into three arrays representing each flower

species: setosa, versicolor, and virginica.

Statistical Analysis: For each flower species, the following statistical properties are computed:

- + Maximum: The maximum value of each feature.
- + Minimum: The minimum value of each feature.
- + Mean: The average value of each feature.
- + Standard Deviation: The measure of the dispersion of each feature.
- + Variance: The square of the standard deviation, indicating the spread of each feature.
- + Median: The middle value of each feature.
- + 25 Percentile: The value below which 25 of the data falls.
- + 75 Percentile: The value below which 75 of the data falls.

Visualization: Bar plots are created for each flower species, representing the above statistical properties for each feature.

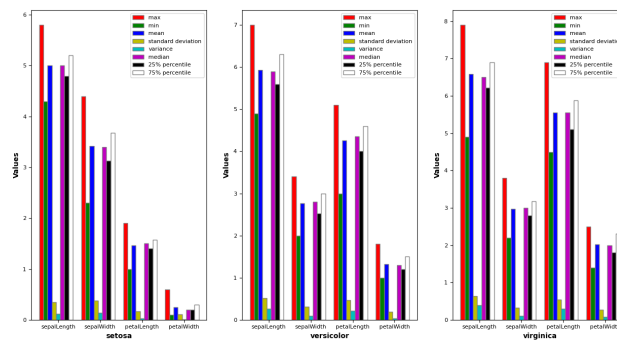


Figure 2: Features for each flower types

4 Exercise 2.6

Compute and report the correlation between each pair-wise feature for all the flower types.

```
1 import itertools
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # load iris from numpy
6 iris = np.genfromtxt('iris.data', delimiter=',', dtype=None, encoding=
    None)
7 iris = np.array([list(x) for x in iris])
8
```

```

9  flower_features = ["sepalLength", "sepalWidth", "petalLength", "↵
    petalWidth"]
10
11  features = iris[:,0:4]
12  features = features.astype(np.float64)
13
14  correlation = {}
15
16  for col_a, col_b in itertools.combinations([0,1,2,3], 2):
17      x = features[:,col_a]
18      y = features[:,col_b]
19
20      xy = np.mean(x*y)
21      mean_x = np.mean(x)
22      mean_y = np.mean(y)
23      var_x = np.var(x)
24      var_y = np.var(y)
25      corr = (xy-mean_x*mean_y)/ np.sqrt(var_x*var_y)
26
27      correlation[f"{flower_features[col_a]}_{flower_features[col_b]}"] ↵
        = corr
28
29  print(correlation)

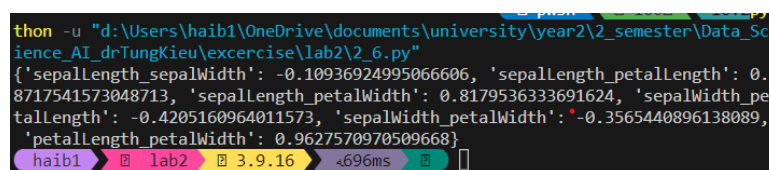
```

Explanation:

The Iris dataset is a classic dataset in the field of machine learning and consists of 150 observations of Iris flowers, with four features measured for each: sepal length, sepal width, petal length, and petal width. The dataset is loaded from a CSV file using the NumPy library.

We use the itertools library to create all possible combinations of the four features and calculate the Pearson correlation coefficient between each pair of features. The Pearson correlation coefficient measures the linear correlation between two variables and ranges from -1 to 1, with a value of 1 indicating a perfect positive correlation, 0 indicating no correlation, and -1 indicating a perfect negative correlation.

We have analyzed the correlation between the different features of the Iris flower dataset using Python and the NumPy and Matplotlib libraries. We found that there are strong positive correlations between petal length and petal width, as well as between sepal length and petal length and petal width.



```

thon -u "d:\Users\haib1\OneDrive\documents\university\year2\2_semester\Data_Sc
ience_AI_drTungKieu\excercise\lab2\2_6.py"
{'sepalLength_sepalWidth': -0.10936924995066606, 'sepalLength_petalLength': 0.
8717541573048713, 'sepalLength_petalWidth': 0.8179536333691624, 'sepalWidth_p
etalLength': -0.4205160964011573, 'sepalWidth_petalWidth': -0.3565440896138089,
'petalLength_petalWidth': 0.9627570970509668}
haib1 lab2 3.9.16 4696ms

```

Figure 3: *Correlation for all flower types*

5 Exercise 2.7

Compute and report the correlation between each pair-wise feature for each flower type.

```
1 import itertools
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # load iris from numpy
6 iris = np.genfromtxt('iris.data', delimiter=',', dtype=None, encoding=↵
    None)
7 iris = np.array([list(x) for x in iris])
8
9 flower_features = ["sepalLength", "sepalWidth", "petalLength", "↵
    petalWidth"]
10
11 features = iris[:,0:4]
12 features = features.astype(np.float64)
13
14 setosa = features[0:50]
15 versicolor = features[50:100]
16 virginica = features[100:150]
17 labels = ["setosa", "versicolor", "virginica"]
18 flowers = [setosa, versicolor, virginica]
19
20 def plot_correlation(flower, index):
21     correlation = {}
22
23     for col_a, col_b in itertools.combinations([0,1,2,3], 2):
24         x = flower[col_a]
25         y = flower[col_b]
26
27         xy = np.mean(x*y)
28         mean_x = np.mean(x)
29         mean_y = np.mean(y)
30         var_x = np.var(x)
31         var_y = np.var(y)
32         corr = (xy-mean_x*mean_y)/ np.sqrt(var_x*var_y)
33
34         correlation[f"{flower_features[col_a]}_{flower_features[col_b]↵
            }]"] = corr
35
36     print(f"Correlation for {labels[index]}: ")
37     print(correlation)
38
```



```
39 for i in range(len(flowers)):  
40     plot_correlation(flowers[i],i)  
41     print("\n")
```

Explanation:

We use the itertools library to create all possible combinations of the four features and calculate the Pearson correlation coefficient between each pair of features for each species of Iris flower. The Pearson correlation coefficient measures the linear correlation between two variables and ranges from -1 to 1, with a value of 1 indicating a perfect positive correlation, 0 indicating no correlation, and -1 indicating a perfect negative correlation.

We have analyzed the correlation between the different features of the Iris flower dataset by species using Python and the NumPy and Matplotlib libraries. We found that the correlation coefficients vary between the different species, suggesting that the relationships between the features may be different for each species.

```
Correlation for setosa:  
{'sepalLength_sepalWidth': 0.99599866124026, 'sepalLength_petalLength': 0.9999  
739110463373, 'sepalLength_petalWidth': 0.9981684517756673, 'sepalWidth_petal  
length': 0.9966070856700843, 'sepalWidth_petalWidth': 0.9973966302370953, 'peta  
lLength_petalWidth': 0.9983334792136432}  
  
Correlation for versicolor:  
{'sepalLength_sepalWidth': 0.9994520573117284, 'sepalLength_petalLength': 0.99  
8140485010611, 'sepalLength_petalWidth': 0.9923513066677933, 'sepalWidth_petal  
Length': 0.9978189869581393, 'sepalWidth_petalWidth': 0.990941502333502, 'peta  
lLength_petalWidth': 0.9976355106625119}  
  
Correlation for virginica:  
{'sepalLength_sepalWidth': 0.9960075939721932, 'sepalLength_petalLength': 0.98  
94618015130858, 'sepalLength_petalWidth': 0.9967851116719527, 'sepalWidth_peta  
lLength': 0.9983022949582452, 'sepalWidth_petalWidth': 0.9991996168621611, 'pe  
talLength_petalWidth': 0.9957952875865307}
```

Figure 4: *Correlation for each flower type*

6 Exercise 2.8

Draw and report the histogram chart for each feature for all the flower types.

```
1 import numpy as np  
2 import matplotlib.pyplot as plt  
3  
4 # load iris from numpy  
5 iris = np.genfromtxt('iris.data', delimiter=',', dtype=None, encoding=↵  
    None)  
6 iris = np.array([list(x) for x in iris])  
7  
8 color = ['pink', 'purple', 'yellow', 'orange']  
9 flower_features = ["sepalLength", "sepalWidth", "petalLength", "↵  
    petalWidth"]
```

```

10 features = iris[:,0:4]
11 features = features.astype(np.float64)
12
13 # histogram chart for each feature for all the flower
14 fig = plt.figure(figsize=(15,8))
15 for i in range(4):
16     plt.subplot(2, 2, i+1)
17     plt.hist(features[:,i], bins=20, color=color[i], edgecolor='black', ←
        , alpha=0.5)
18     plt.xlabel(f"{flower_features[i]} cm")
19     plt.ylabel("Frequency")
20
21     plt.title(f"histogram of {flower_features[i]} for all flower types ←
        ")
22     plt.tight_layout()
23
24 plt.show()

```

Explanation:

We use the Matplotlib library to create a histogram chart for each feature of the Iris flower dataset. A histogram is a chart that shows the distribution of a dataset by dividing it into a set of bins and counting the number of observations that fall into each bin.

We can see that the distributions of the different features vary. Sepal length and sepal width have a relatively normal distribution, while petal length and petal width have a bimodal distribution. This suggests that there may be different subgroups or clusters within the Iris flower dataset based on these features.

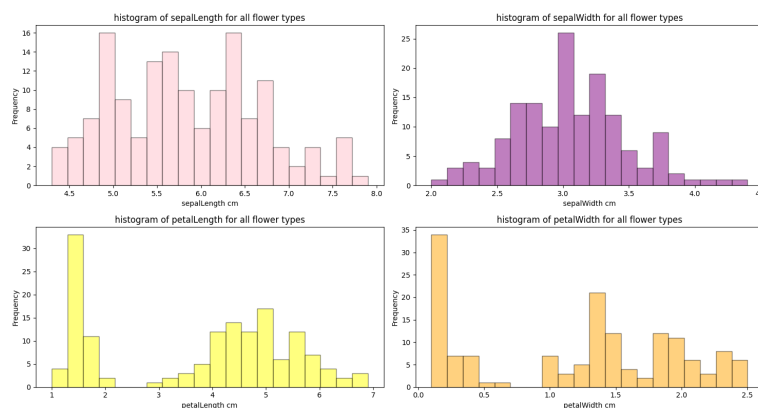


Figure 5: *Histogram for all flower types*

7 Exercise 2.9

Draw and report the histogram chart for each feature for each flower type.

```
1 # color of chart for each features
2 color = ['pink', 'purple', 'yellow', 'orange']
3 flower_features = ["sepalLength", "sepalWidth", "petalLength", "↵
    petalWidth"] # 4 features
4
5 features = iris[:,0:4] # eliminate the label column
6 features = features.astype(np.float64) # convert to float 64 datatype
7
8 # divide the iris into 3 types of flower
9 setosa = features[0:50]
10 versicolor = features[50:100]
11 virginica = features[100:150]
12
13 labels = ["setosa", "versicolor", "virginica"] # 3 flowers
14 flowers = [setosa, versicolor, virginica] # list of flowers
15
16 # histogram chart for each feature for all the flower
17 def histogram(flower, index):
18     # create new figure and set a size
19     fig = plt.figure(figsize=(15,8))
20
21     # plot each features into a chart
22     for i in range(4):
23         plt.subplot(2, 2, i+1)
24         plt.hist(flower[:,i], bins=20, color=color[i], edgecolor='↵
            black', alpha=0.5)
25         plt.xlabel(f"{flower_features[i]} cm")
26         plt.ylabel("Frequency")
27
28         plt.title(f"histogram of {flower_features[i]} for {labels[↵
            index]}")
29         plt.tight_layout()
30
31     plt.show()
32
33 for i in range(len(flowers)):
34     histogram(flowers[i], i)
```

Explanation:

We can see that the distributions of the different features vary between the different species of Iris flower. For example, setosa flowers have narrower sepal length and width, and shorter petal length and width compared to versicolor and virginica flowers. Virginica flowers have the longest sepal and petal lengths, while versicolor flowers have the widest petal widths.

We have analyzed the distribution of the different features of the Iris flower dataset by species

using Python and the NumPy and Matplotlib libraries. We found that the distributions of the features vary between the different species of Iris flower, suggesting that these features may be useful in distinguishing between different species of Iris flowers.

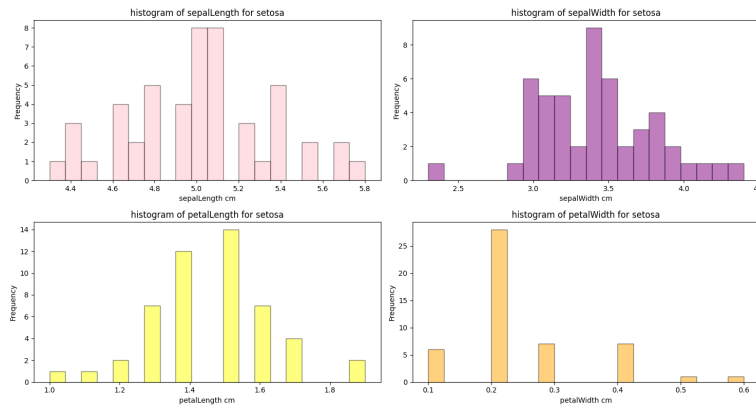


Figure 6: *Histogram chart for setosa*

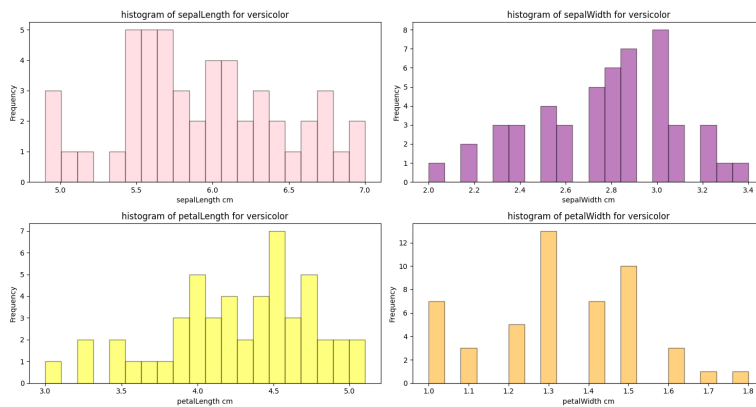


Figure 7: *Histogram chart for versicolor*

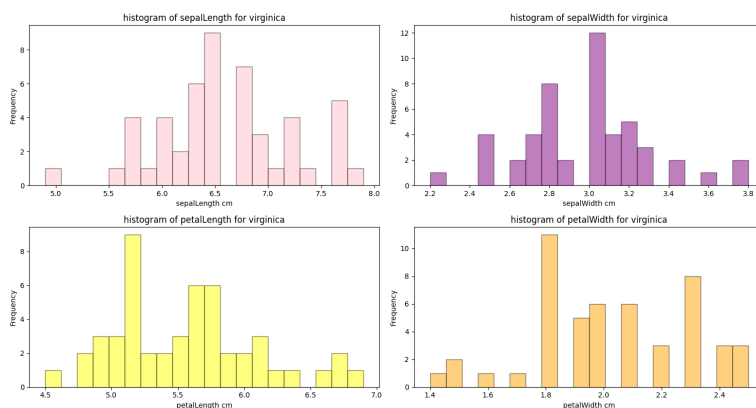


Figure 8: *Histogram chart for virginica*

8 Exercise 2.10

Draw and report the box chart for each feature of all the flower types.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # load iris from numpy
5 iris = np.genfromtxt('iris.data', delimiter=',', dtype=None, encoding=↵
    None)
6 iris = np.array([list(x) for x in iris])
7
8 flower_features = ["sepalLength", "sepalWidth", "petalLength", "↵
    petalWidth"]
9 features = iris[:,0:4]
10 features = features.astype(np.float64)
11
12 # box chart for each feature for all the flower
13 data = []
14 for i in range(4):
15     data.append(features[:,i])
16
17 # make the boxplot become colorful and pretty
18 plt.figure(figsize=(15,8))
19 plt.boxplot(data, labels=flower_features, patch_artist=True, ↵
    medianprops={'linewidth':2})
20 plt.xlabel("Features")
21 plt.ylabel("cm")
22
23 plt.title("Boxplot of Iris Flower Features")
24 plt.show()
```

Explanation:

We use the Matplotlib library to create a boxplot chart for each feature of the Iris flower dataset. A boxplot is a chart that displays the distribution of a dataset by showing the median, quartiles, and outliers of the data.

We can see that the distributions of the different features vary. Sepal length and sepal width have a relatively similar interquartile range (IQR) and median, while petal length and petal width have a wider range and higher median. Additionally, there are some outliers present in the data, particularly in the petal length and petal width features.

We have analyzed the distribution of the different features of the Iris flower dataset using boxplots in Python and the NumPy and Matplotlib libraries. We found that the distributions of the features vary, with sepal length and sepal width having a relatively similar distribution, while petal length and petal width have a wider range and higher median.

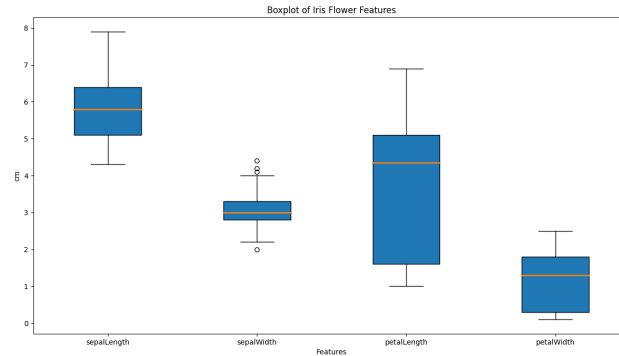


Figure 9: *Box chart for all flower types*

9 Exercise 2.11

Draw and report the box chart for each feature of each flower type.

Explanation:

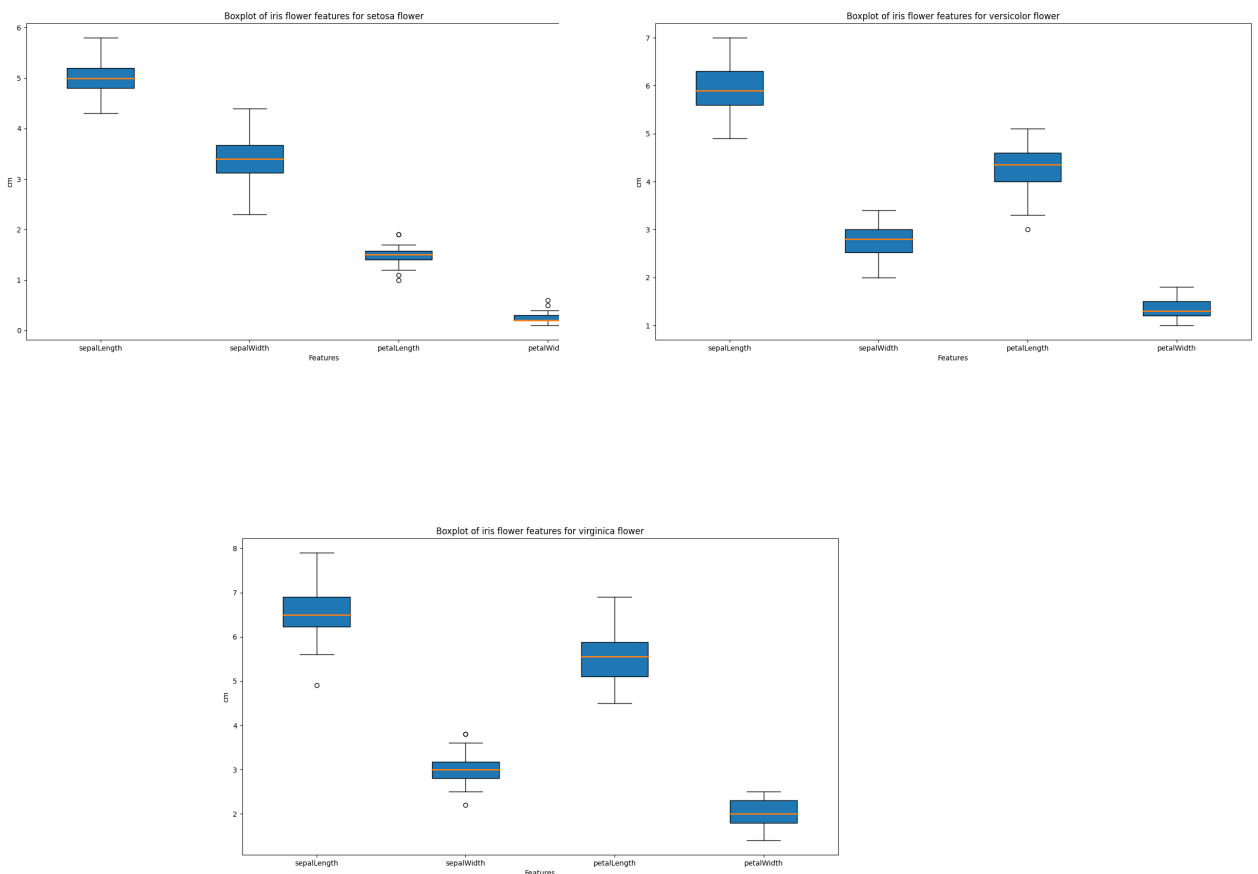
In summary, this code loads the Iris dataset, divides it into subsets based on flower types, and creates boxplots to visualize the distribution of each feature for each flower type.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # load iris from numpy
5 iris = np.genfromtxt('iris.data', delimiter=',', dtype=None, encoding=↵
    None)
6 iris = np.array([list(x) for x in iris])
7
8 flower_features = ["sepalLength", "sepalWidth", "petalLength", "↵
    petalWidth"]
9 features = iris[:,0:4]
10 features = features.astype(np.float64)
11 setosa = features[0:50]
12 versicolor = features[50:100]
13 virginica = features[100:150]
14
15 labels = ["setosa", "versicolor", "virginica"]
16 flowers = [setosa, versicolor, virginica]
17
18 def box_chart(flower, index):
19     # box chart for each feature for all the flower
20     data = []
21     for i in range(4):
22         data.append(flower[:,i])
23
```

```

24 plt.figure(figsize=(15,8))
25 plt.boxplot(data, labels=flower_features, patch_artist=True, ←
    medianprops={'linewidth':2})
26 plt.xlabel("Features")
27 plt.ylabel("cm")
28
29 plt.title("Boxplot of iris flower features for " + labels[index] + ←
    " flower")
30 plt.show()
31
32 for i in range(3):
33     box_chart(flowers[i], i)

```



10 Exercise 2.12

Draw and report the distribution chart for each feature of all the flower types.

Explanation: In summary, this code loads the Iris dataset, visualizes the distribution of each feature using histograms and KDE plots, and displays the resulting plots in a 2x2 grid. The colors, labels, and titles are customized to represent the features and their distributions for all flower types.

```

1
2 # histogram chart for each feature for all the flower
3 fig, axes = plt.subplots(2, 2, figsize=(15, 8))
4 for i, ax in enumerate(axes.flat):
5     sns.histplot(features[:, i], bins=20, color=color[i], edgecolor='↵
        black', alpha=0.5, ax=ax)
6     sns.kdeplot(features[:, i], color='blue', fill=True, alpha=0.3, ax↵
        =ax)
7     ax.set_xlabel(f"{flower_features[i]} cm")
8     ax.set_ylabel("Frequency")
9     ax.set_title(f"Distribution of {flower_features[i]} for all flower↵
        types")
10    ax.legend()
11 plt.tight_layout()
12 plt.show()

```

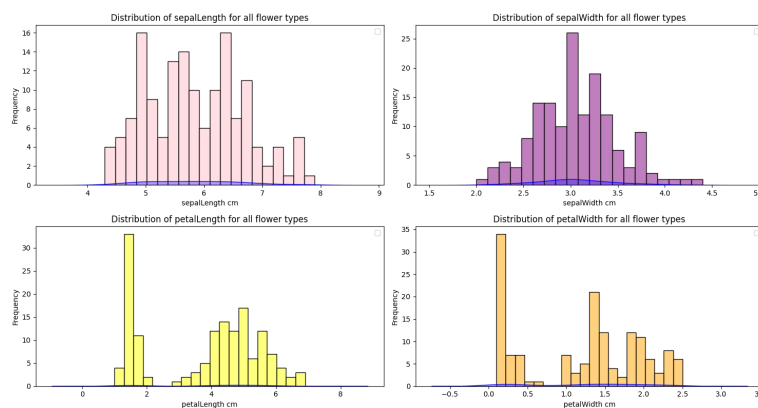


Figure 10: *Distribution chart for all flower types*

11 Exercise 2.13

Draw and report the distribution chart for each feature of each flower type.

Explanation:

In summary, this code loads the Iris dataset, divides it into subsets based on flower types, and creates histograms and KDE plots to visualize the distribution of each feature for each flower type. The colors, labels, and titles are customized to represent the features and their distributions for each flower type.

```

1 setosa = features[0:50]
2 versicolor = features[50:100]
3 virginica = features[100:150]
4 labels = ["setosa", "versicolor", "virginica"]

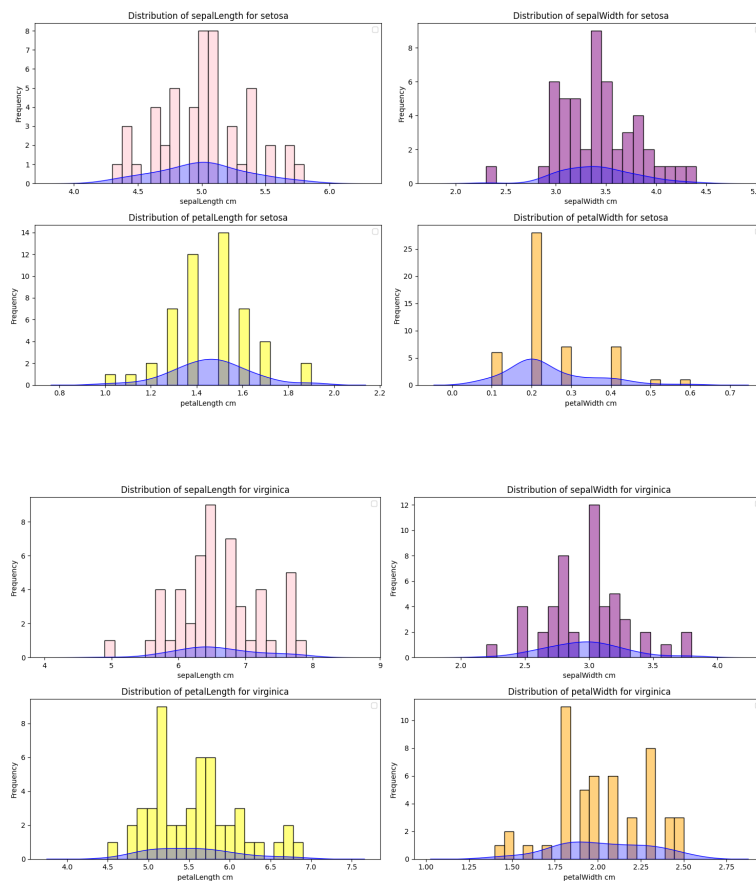
```

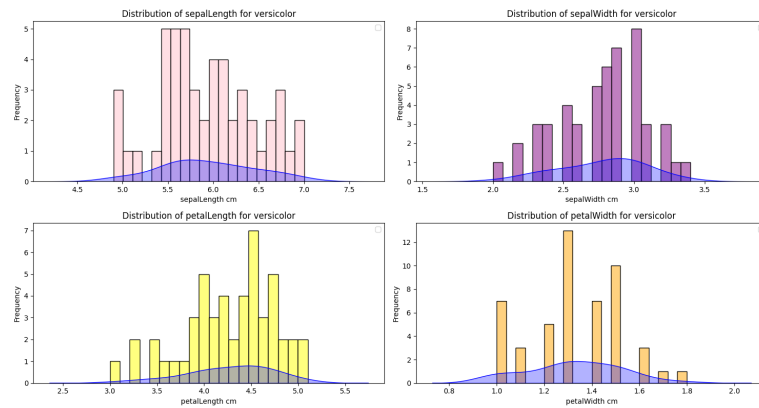


```

5 flowers = [setosa, versicolor, virginica]
6
7 def histogram(flower, index):
8     # histogram chart for each feature for all the flower
9     fig, axes = plt.subplots(2, 2, figsize=(15, 8))
10    for i, ax in enumerate(axes.flat):
11        sns.histplot(flower[:, i], bins=20, color=color[i], edgecolor='black', alpha=0.5, ax=ax)
12        sns.kdeplot(flower[:, i], color='blue', fill=True, alpha=0.3, ax=ax)
13        ax.set_xlabel(f"{flower_features[i]} cm")
14        ax.set_ylabel("Frequency")
15        ax.set_title(f"Distribution of {flower_features[i]} for {labels[index]}")
16        ax.legend()
17    plt.tight_layout()
18
19    plt.show()
20
21 for i in range(len(flowers)):
22     histogram(flowers[i], i)

```





12 Exercise 2.14

Draw and report the scatter chart for each pair-wise feature of all the flower types.

Explanation:

In summary, this code loads the Iris dataset, creates scatter plots to visualize the relationship between pairs of features, and displays the resulting plots in a 2x3 grid. Each scatter plot represents a combination of two features, and the colors, titles, and axis labels are customized based on the selected features.

```
1 import itertools
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # load iris from numpy
6 iris = np.genfromtxt('iris.data', delimiter=',', dtype=None, encoding=↵
    None)
7 iris = np.array([list(x) for x in iris])
8
9 flower_features = ["sepalLength", "sepalWidth", "petalLength", "↵
    petalWidth"]
10
11 features = iris[:,0:4]
12 features = features.astype(np.float64)
13 color = ['purple', 'green', 'brown', 'orange', 'red', 'pink']
14
15 index = 0
16 # Scatter plot for each column
17 fig = plt.figure(figsize=(15,8))
18 for col_a, col_b in itertools.combinations([0,1,2,3], 2):
19     x = features[:,col_a]
20     y = features[:,col_b]
21     plt.subplot(2,3,index+1)
22     plt.scatter(x,y,color=color[index])
23     plt.title(f"Scatter plot of {flower_features[col_a]} vs {↵
```

```

    flower_features[col_b])")
24 plt.xlabel(flower_features[col_a],color="blue")
25 plt.ylabel(flower_features[col_b],color="blue")
26 plt.tight_layout()
27
28 index += 1
29 plt.show()

```

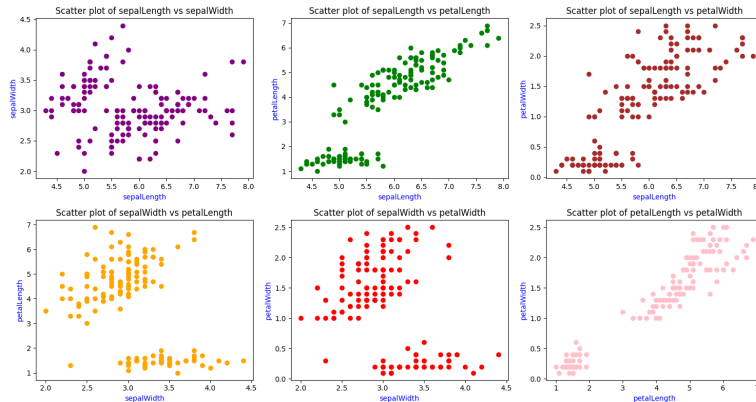


Figure 11: Scatter chart for each pair- wise feature of all the flower types

13 Exercise 2.15

Draw and report the scatter chart for each pair-wise feature of each flower type.

Explanation:

In summary, this code loads the Iris dataset, creates scatter plots to visualize the relationship between pairs of features for each flower type, and displays the resulting plots. Each scatter plot represents a combination of two features, and the colors, titles, and axis labels are customized based on the selected features and the flower type.

```

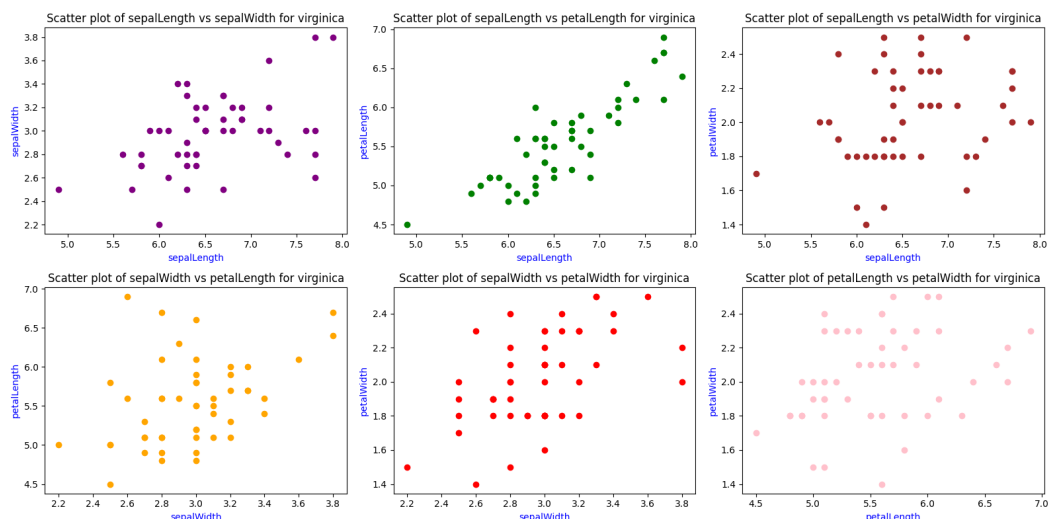
1 import itertools
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # load iris from numpy
6 iris = np.genfromtxt('iris.data', delimiter=',', dtype=None, encoding=↵
    None)
7 iris = np.array([list(x) for x in iris])
8
9 flower_features = ["sepalLength", "sepalWidth", "petalLength", "↵
    petalWidth"]
10
11 features = iris[:,0:4]

```

```

12 features = features.astype(np.float64)
13 color = ['purple', 'green', 'brown', 'orange', 'red', 'pink']
14
15 setosa = features[0:50]
16 versicolor = features[50:100]
17 virginica = features[100:150]
18
19 labels = ["setosa", "versicolor", "virginica"]
20 flowers = [setosa, versicolor, virginica]
21
22 def plot_histogram(flower, index):
23     i = 0
24     # Scatter plot for each column
25     fig = plt.figure(figsize=(16,8))
26     for col_a, col_b in itertools.combinations([0,1,2,3], 2):
27         x = flower[:,col_a]
28         y = flower[:,col_b]
29         plt.subplot(2,3,i+1)
30         plt.scatter(x,y,color=color[i])
31         plt.title(f"Scatter plot of {flower_features[col_a]} vs {↵
                flower_features[col_b]} for {labels[index]}")
32         plt.xlabel(flower_features[col_a],color="blue")
33         plt.ylabel(flower_features[col_b],color="blue")
34         plt.tight_layout()
35
36         i += 1
37     plt.show()
38
39 for i in range(len(flowers)):
40     plot_histogram(flowers[i], i)

```





14 Exercise 2.16

Draw and report the bubble chart for each pair-wise feature of all the flower types.

Explanation:

```
1 import itertools
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # load iris from numpy
6 iris = np.genfromtxt('iris.data', delimiter=',', dtype=None, encoding=↵
    None)
7 iris = np.array([list(x) for x in iris])
8
9 flower_features = ["sepalLength", "sepalWidth", "petalLength", "↵
    petalWidth"]
10
11 features = iris[:,0:4]
12 features = features.astype(np.float64)
13 color = ['purple', 'green', 'brown', 'orange', 'red', 'pink']
14
15 index = 0
16 # Scatter plot for each column
17 fig = plt.figure(figsize=(15,8))
18 for col_a, col_b in itertools.combinations([0,1,2,3], 2):
19     x = features[:,col_a]
20     y = features[:,col_b]
21     plt.subplot(2,3,index+1)
22     #bubble plot
23     plt.scatter(x,y,s=features[:,3]*100,color=color[index],alpha=0.5)
24     plt.title(f"Bubble plot of {flower_features[col_a]} vs {↵
        flower_features[col_b]}")
25     plt.xlabel(flower_features[col_a],color="blue")
26     plt.ylabel(flower_features[col_b],color="blue")
27     plt.tight_layout()
28
29     index += 1
30 plt.show()
```

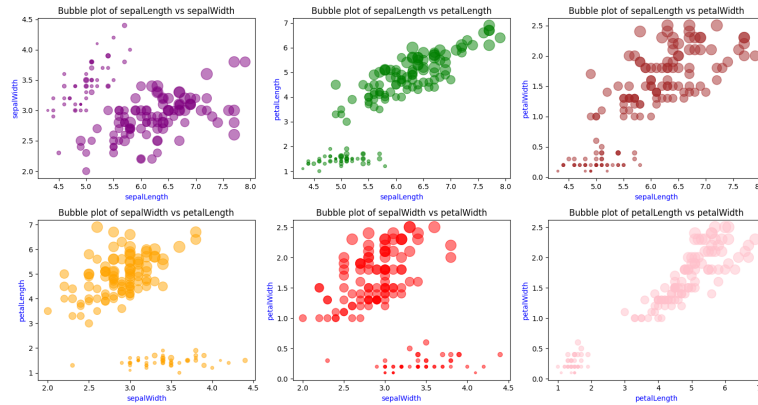


Figure 12: bubble chart for each pair wise feature of all the flower types

15 Exercise 2.17

Draw and report the bubble chart for each pair-wise feature of each flower type.

Explanation:

In summary, this code generates bubble plots instead of scatter plots to visualize the relationship between pairs of features for each flower type in the Iris dataset. The size of the bubbles is determined by the petal width, creating varying bubble sizes. The colors, titles, and axis labels are customized based on the selected features and the flower type.

```

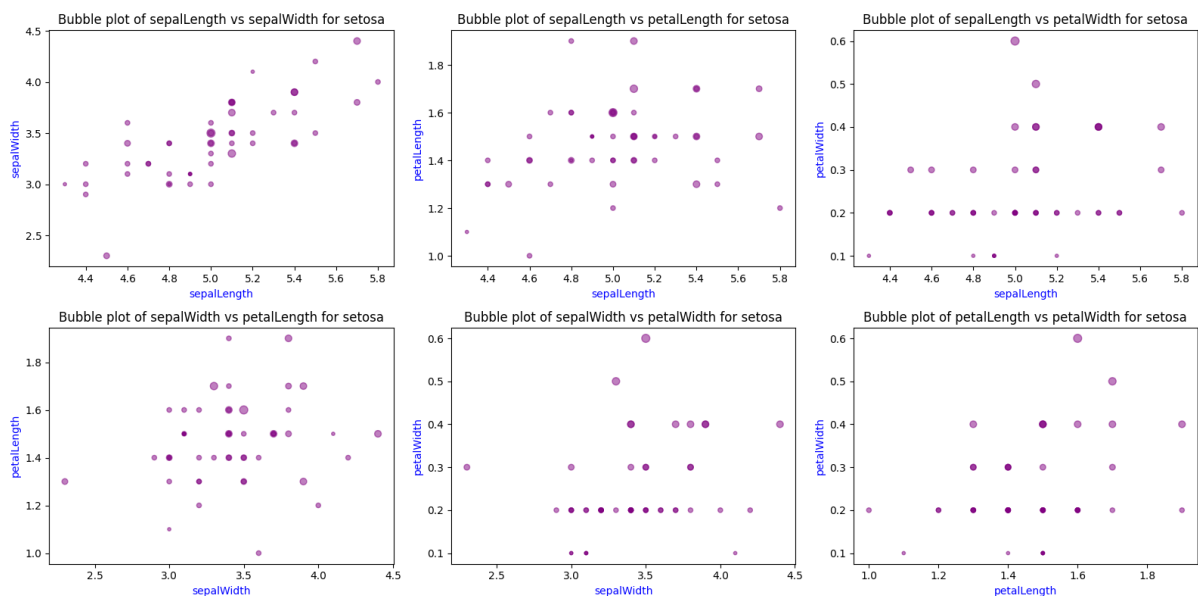
1 import itertools
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # load iris from numpy
6 iris = np.genfromtxt('iris.data', delimiter=',', dtype=None, encoding=↵
    None)
7 iris = np.array([list(x) for x in iris])
8
9 flower_features = ["sepalLength", "sepalWidth", "petalLength", "↵
    petalWidth"]
10
11 features = iris[:,0:4]
12 features = features.astype(np.float64)
13 color = ['purple', 'green', 'brown', 'orange', 'red', 'pink']
14
15 setosa = features[0:50]
16 versicolor = features[50:100]
17 virginica = features[100:150]
18
19 labels = ["setosa", "versicolor", "virginica"]
20 flowers = [setosa, versicolor, virginica]
21

```

```

22 def plot_histogram(flower, index):
23     i = 0
24     # Scatter plot for each column
25     fig = plt.figure(figsize=(16,8))
26     for col_a, col_b in itertools.combinations([0,1,2,3], 2):
27         x = flower[:,col_a]
28         y = flower[:,col_b]
29         plt.subplot(2,3,i+1)
30         plt.scatter(x,y,s=flower[:,3]*100,color=color[index],alpha=
31                     =0.5)
32         plt.title(f"Bubble plot of {flower_features[col_a]} vs {flower_features[col_b]} for {labels[index]}")
33         plt.xlabel(flower_features[col_a],color="blue")
34         plt.ylabel(flower_features[col_b],color="blue")
35         plt.tight_layout()
36         i += 1
37     plt.show()
38
39 for i in range(len(flowers)):
40     plot_histogram(flowers[i], i)

```

Figure 13: *Features for setosa*

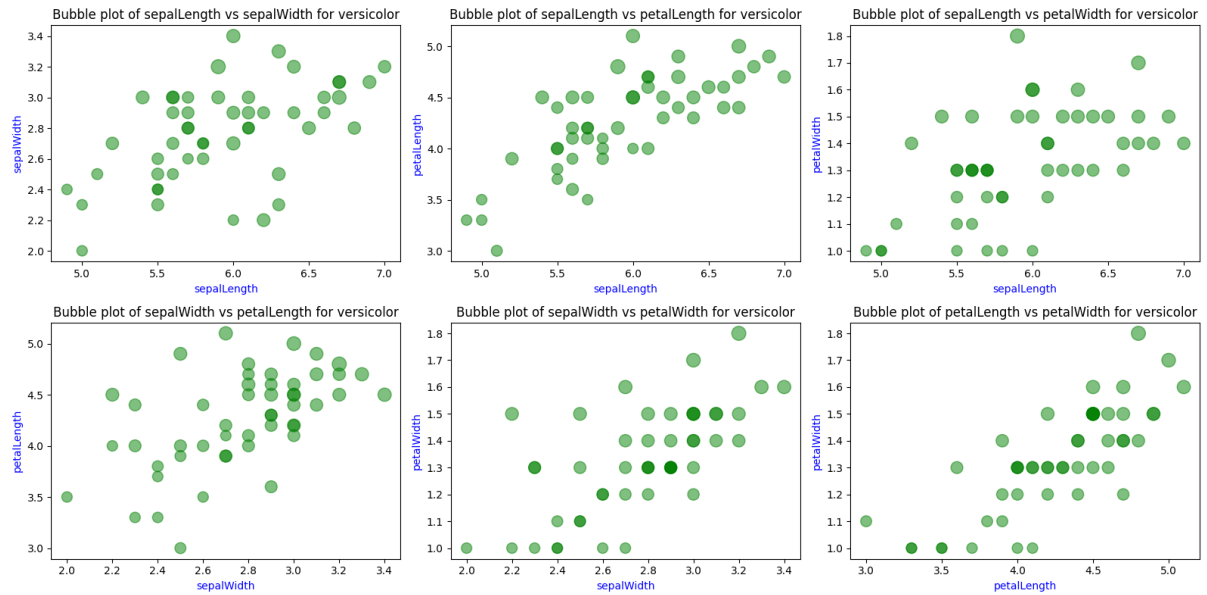


Figure 14: *Features for versicolor*

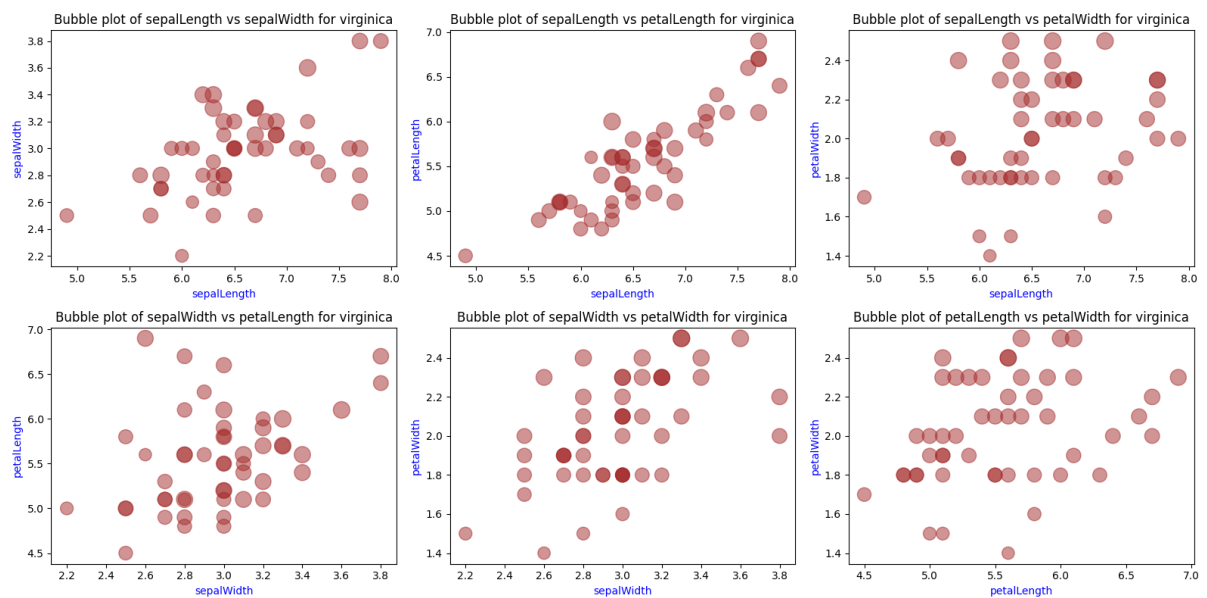


Figure 15: *Features for virginica*

16 Exercise 2.18

Draw and report the density chart for each pair-wise feature of all the flower types.

Explanation:

In summary, this code generates a density plot for the Iris dataset using seaborn's `kdeplot()` function. The plot shows the density distribution of the Iris species based on a specific length variable. The x-axis represents the length, the y-axis represents the density, and the plot is titled "Iris Species Density".

```
1 import matplotlib.pyplot as plt
2 import seaborn as sns
3
4 iris = sns.load_dataset("iris")
5
6 plt.figure(figsize=(10, 5))
7 plt.title("Iris Species Density")
8 plt.xlabel("Length")
9 plt.ylabel("Density")
10 sns.kdeplot(data=iris, fill=True, alpha=0.5, linewidth=0.5)
11 plt.show()
```

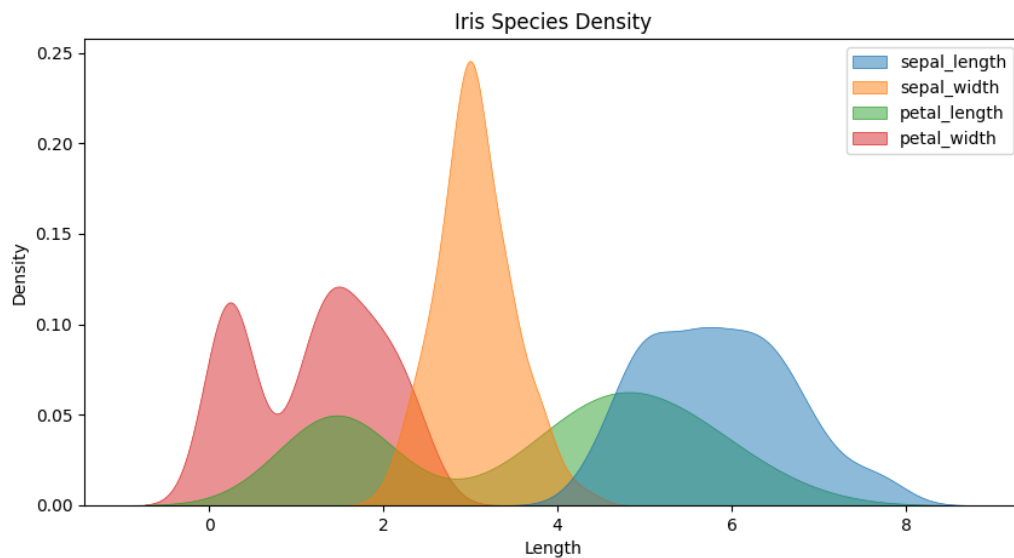


Figure 16: *Features for each flower types*

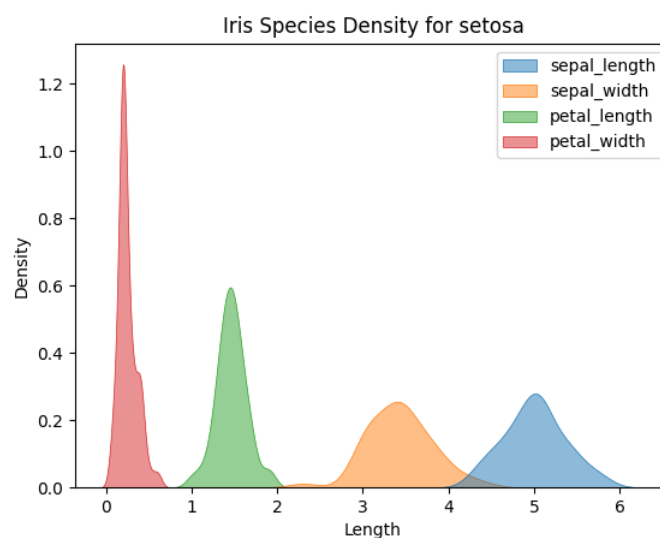
17 Exercise 2.19

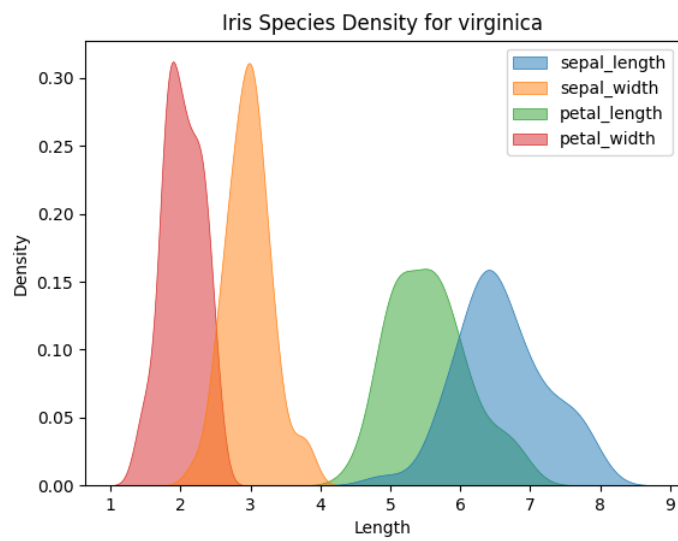
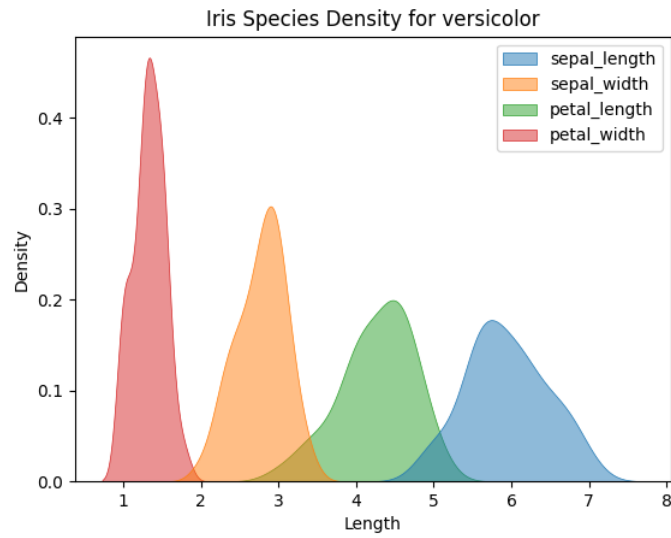
Draw and report the density chart for each pair-wise feature of each flower type.

Explanation:

In summary, this code creates separate density plots for each species of Iris flowers in the Iris dataset using seaborn's `kdeplot()` function. Each density plot represents the density distribution of a specific species based on a length variable. The plots are customized with species-specific titles, x-axis label ("Length"), and y-axis label ("Density").

```
1 setosa = iris[iris["species"] == "setosa"]
2 versicolor = iris[iris["species"] == "versicolor"]
3 virginica = iris[iris["species"] == "virginica"]
4
5 labels = ["setosa", "versicolor", "virginica"]
6 flowers = [setosa, versicolor, virginica]
7
8 def plot(flower, index):
9     sns.kdeplot(data=flower, fill=True, alpha=0.5, linewidth=0.5)
10    plt.title(f"Iris Species Density for {labels[index]}")
11    plt.xlabel("Length")
12    plt.ylabel("Density")
13    plt.show()
14
15 for i in range(len(flowers)):
16     plot(flowers[i], i)
```





18 Exercise 2.20

Draw and report the parallel chart for all the flower types.

Explanation:

In summary, this code uses the Iris dataset to create a parallel coordinates plot. Each line in the plot represents a data point, and the vertical axes represent different variables or features. The lines are colored based on the species they belong to, allowing for visual comparison of how different species differ in their feature values.

```
1 # libraries
2 import pandas
3 import matplotlib.pyplot as plt
4 from pandas.plotting import parallel_coordinates
5
6 # Take the iris dataset
7 import seaborn as sns
8 data = sns.load_dataset('iris')
9
10 # Make the plot
11 parallel_coordinates(data, 'species', colormap=plt.get_cmap("Set2"))
12 plt.show()
```

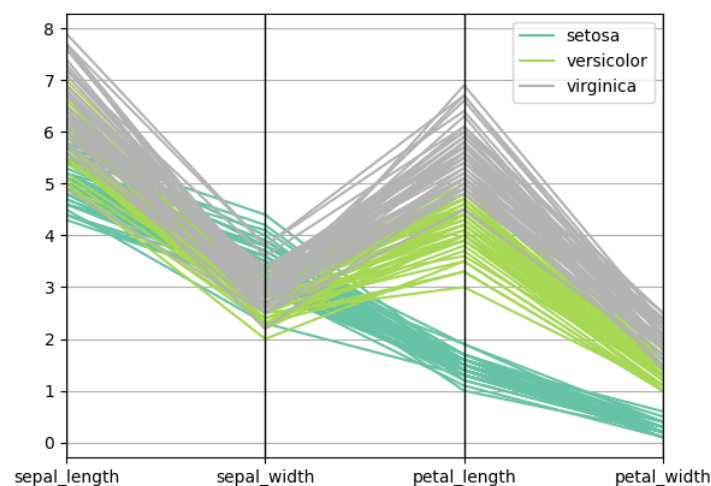


Figure 17: Density chart for each pair wise feature of each flower type

19 Exercise 2.21

Draw and report the deviation chart for all the flower types.

Explanation:

In summary, this code calculates the mean and standard deviation for each feature of the Iris flower dataset and creates a deviation chart using error bars to visualize the deviation of feature values for different flower types.

```
1 import itertools
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # load iris from numpy
6 iris = np.genfromtxt('iris.data', delimiter=',', dtype=None, encoding=↵
    None)
7 iris = np.array([list(x) for x in iris])
8
9 features = iris[:, 0:4]
10 features = features.astype(np.float64)
11
12 # Calculate the mean and standard deviation for each feature
13 mean_values = np.mean(features, axis=0)
14 std_values = np.std(features, axis=0)
15
16 # Set the labels for the x-axis
17 labels = ['sepalLength', 'sepalWidth', 'petalLength', 'petalWidth']
18
19 # Plot the deviation chart
20 plt.figure(figsize=(10, 6))
21 plt.errorbar(labels, mean_values, yerr=std_values, fmt='o', capsize=5)
22
23 # Set the title and labels
24 plt.title('Deviation Chart for Iris Flower Types')
25 plt.xlabel('Features')
26 plt.ylabel('Values')
27
28 # Show the plot
29 plt.show()
```

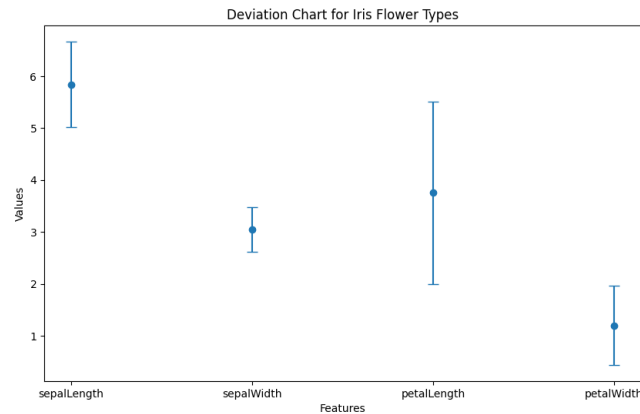


Figure 18: *deviation chart for all the flower types*

20 Exercise 2.22

Draw and report the Andrews curves chart for all the flower types.

Explanation:

The Andrews Curves plot is created using the `andrews_curves()` function. This function takes the dataset data and the column name 'species' as inputs. The colormap parameter is set to `plt.get_cmap("Set2")` to specify the color scheme for the curves.

```
1 from pandas.plotting import andrews_curves
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4 data = sns.load_dataset('iris')
5 # Create an Andrews curves plot
6 plt.title('Andrews curves for Iris Flower Types')
7 andrews_curves(data, 'species', colormap=plt.get_cmap("Set2"))
8 plt.show()
```

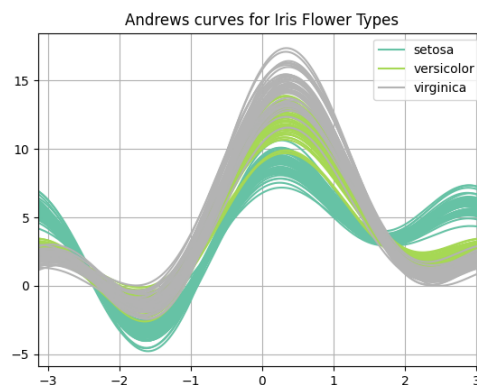


Figure 19: *Andrews Curves chart for all the flower types*