Lab 4 Report

Nakkina Vinay (B21AI023)

Question 2

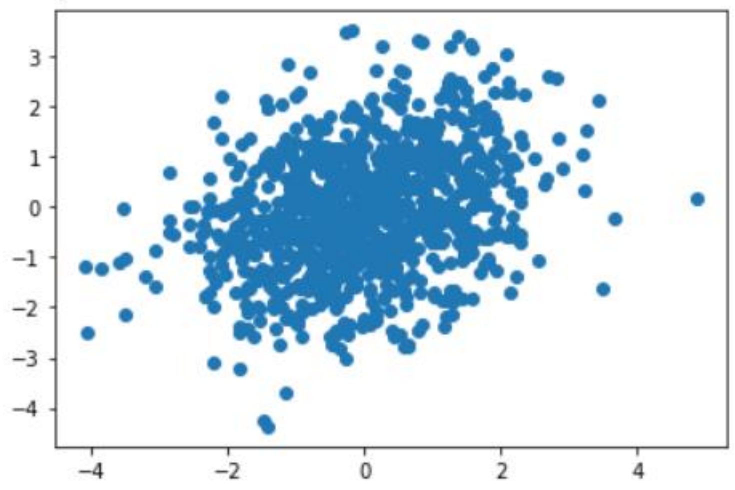
Getting Sample Random points from the Multivariate Normal Distribution

- Mean and Covariance matrix are given in the question
- From the order of covariance matrix, we get the dimensions as 2

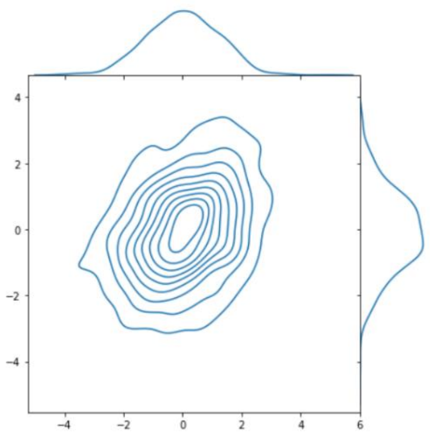
```
Mean is:  [0 0]
Covariance is:
 [[1.5 0.5]
 [0.5 1.5]]
```

- Using the np.random.multivariate_normal() we can generate the random points by using the mean and covariance matrix and the dimensions
- After generating the data points, by plotting the graph we get

```
<matplotlib.collections.PathCollection at 0x7f902374
```



- Plotting the density function



Subpart 1: Covariance Matrix of the sample and Eigenvalues and EigenVectors

- We will calculate the mean of each variable in the sample using np.mean()
- After calculating the mean, subtracting the mean from each data in the sample to get the centred data
- Calculating the covariance matrix between each pair of variables using the formula

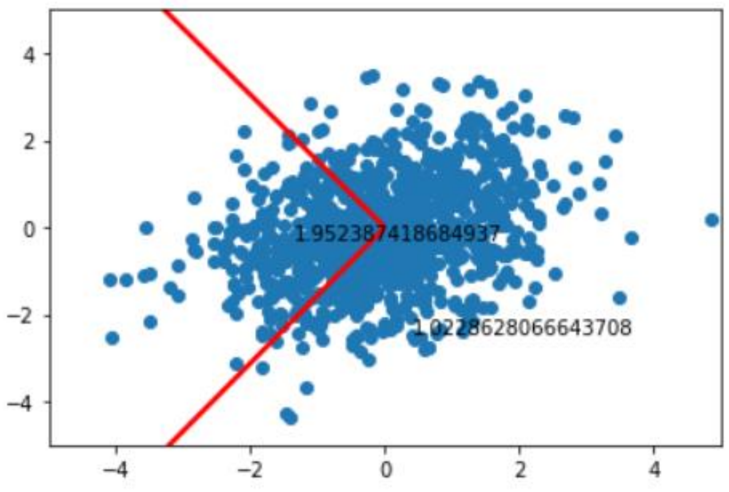
```
np.dot(centred_data.T,centred_data) / (X.shape[0]-1)
```

```
Mean of each variable in the sample:  [ 0.01228702 -0.06333361]

Centred sample:
 [[ 0.39984019 -2.39651682]
 [-1.36492741 -0.2343689 ]
 [-0.88813578 -0.90046028]
 ...
 [ 1.11821426  1.18274065]
 [ 0.16784734 -0.5632746 ]
 [-0.47244561 -1.69886078]]

Covariance Matrix of the sample:
 [[1.48225682 0.4647313 ]
 [0.4647313  1.49299341]]
```

- After calculating the covariance matrix we can find the eigenvalues and eigen vectors using the eig function imported from numpy.linalg library

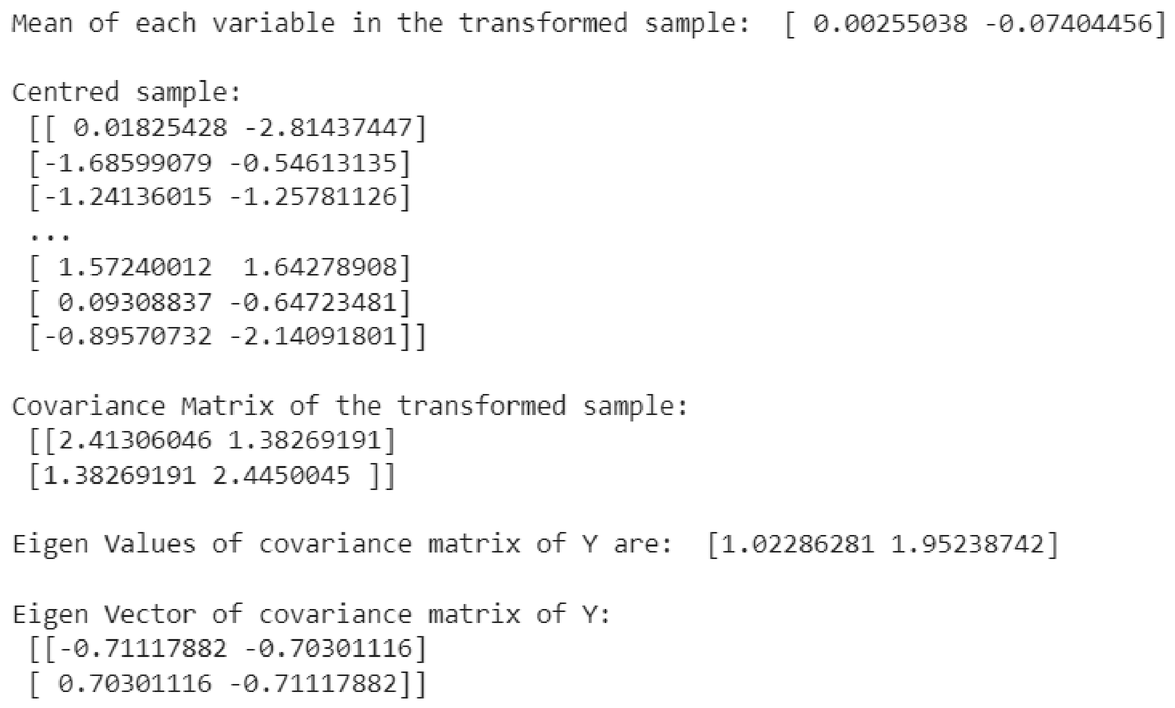
```
Eigen Values are:  [1.02286281 1.95238742]

Eigen Vector:
 [[-0.71117882 -0.70301116]
 [ 0.70301116 -0.71117882]]
```

- Plotting the eigenvalues and eigen vectors superimposed on the dataset



Subpart 2: Performing the transformation and finding the covariance matrix

- First we will calculate the negative square root of the covariance matrix using fractional_matrix_power() imported from scipy.linalg
- We will dot the X data points with the calculated negative square root of the covariance matrix
- We will calculate the covariance matrix of the transferred data points the same as that we have done for calculating the covariance matrix for the normal data points
- After calculating the covariance matrix of the transferred matrix, we will find the eigenvalues and eigen vectors



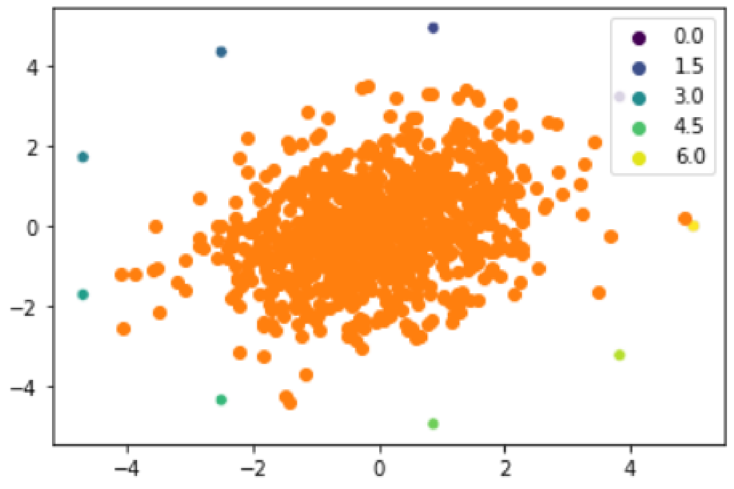
```
Mean of each variable in the transformed sample:  [ 0.00255038 -0.07404456]

Centred sample:
 [[ 0.01825428 -2.81437447]
 [-1.68599079 -0.54613135]
 [-1.24136015 -1.25781126]
 ...
 [ 1.57240012  1.64278908]
 [ 0.09308837 -0.64723481]
 [-0.89570732 -2.14091801]]

Covariance Matrix of the transformed sample:
 [[2.41306046 1.38269191]
 [1.38269191 2.4450045 ]]

Eigen Values of covariance matrix of Y are:  [1.02286281 1.95238742]

Eigen Vector of covariance matrix of Y:
 [[-0.71117882 -0.70301116]
 [ 0.70301116 -0.71117882]]
```

- The purpose of transformation is it can be used to rotate or rescale the data in order to align the principal components of the data with the axes of the plot. This can simplify the visualization of the data by reducing the dimensionality of the data and highlighting the most important features or patterns in data. It also helps in normalizing the data or to standardize the variables.
- Covariance matrix obtained after the transformation has higher values than the previous one.

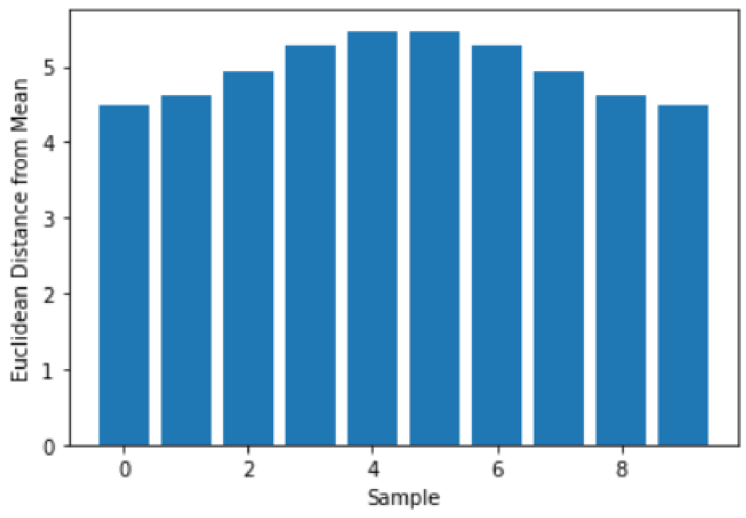
Subpart 3: Uniformly sample the 10 points on the curve x2+y2=25

- By using the np.linspace() on the angle we can make 10 uniformly sample points on the curve
- After making 10 equal angles, we multiply it with radius and the cos and sin to get the X and Y points
- After getting the 10 points, using scatter plot and hue to plot the graph and get different colours for the points

```
<matplotlib.collections.PathCollection at 0x7f901dc52670:
```



- Calculating the mean of the data points
- Calculating the Euclidean distance of each point from the mean and plotting the bar graphs



- Calculating the covariance matrix of the data points by the same method done before. We get the covariance matrix as

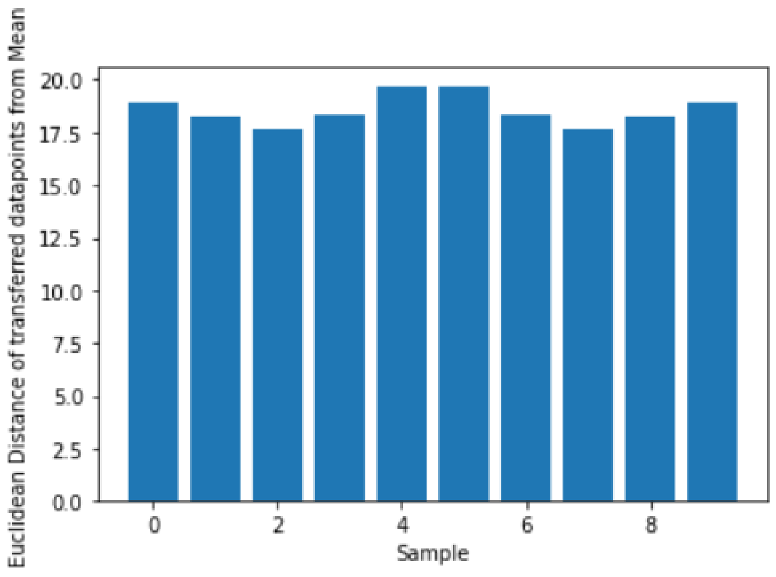
```
Covariance matrix of data points
 [[1.50000000e+01 5.23153433e-16]
 [5.23153433e-16 1.25000000e+01]]
```

Subpart 4: Transformation of data points and calculating covariance matrix

- Calculating the Square root of the covariance matrix of the data points
- Performing the transformation for getting the transferred data points
- After getting the transferred data points we will find the covariance matrix of the transferred data by the same way

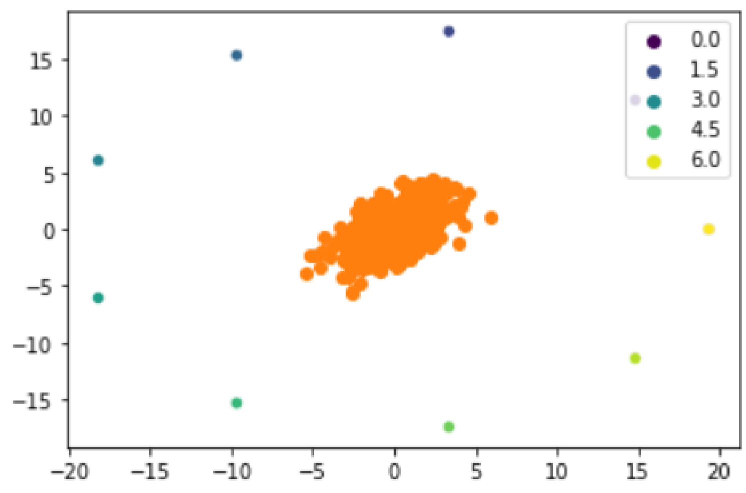
```
Mean of each variable in the transformed sample:  [ 1.93649167e+00 -9.30577513e-16]

Centred Sample:
 [[ 1.74284251e+01  1.28365318e-15]
 [ 1.28978952e+01  1.13629869e+01]
 [ 1.42619083e+00  1.74091060e+01]
 [-1.16189500e+01  1.53093109e+01]
 [-2.01335610e+01  6.04611907e+00]
 [-2.01335610e+01 -6.04611907e+00]
 [-1.16189500e+01 -1.53093109e+01]
 [ 1.42619083e+00 -1.74091060e+01]
 [ 1.28978952e+01 -1.13629869e+01]
 [ 1.74284251e+01 -3.04612710e-15]]

Covariance Matrix of the transformed dataponts:
 [[2.25000000e+02 1.40680056e-14]
 [1.40680056e-14 1.56250000e+02]]
```

- Calculating the Euclidean distance of transferred data points and plotting the bar graph of the data points from the mean



- Plotting the points in Q along with data points in Y

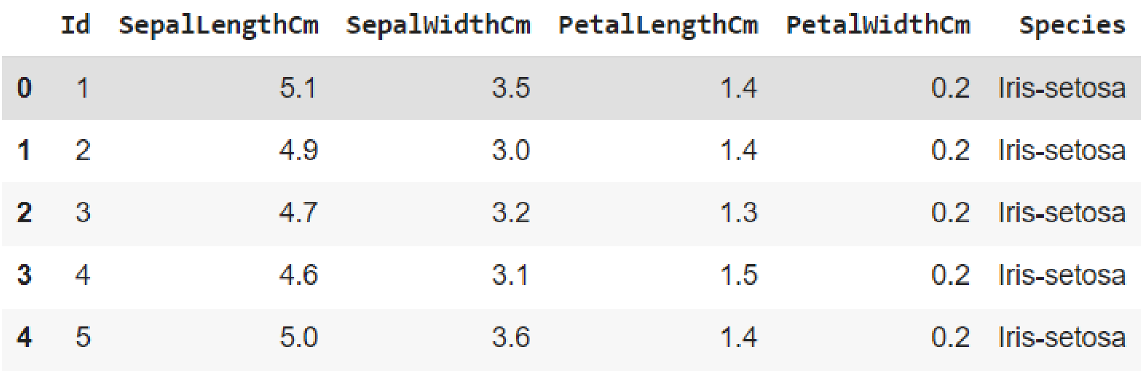
```
<matplotlib.collections.PathCollection at 0x7f90234520d0>
```



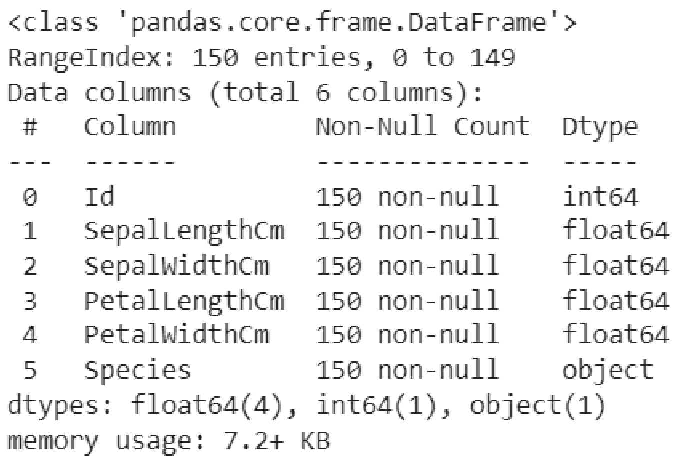
- The graph of Euclidean distance of data points from the mean before was first increasing and decreasing but the graph of euclidean distance of transferred data points from the mean was first decreasing, and then increasing and then decreasing and increasing

Question 1

- Getting the dataset from the read_csv() and doing the head() and info() on the dataset

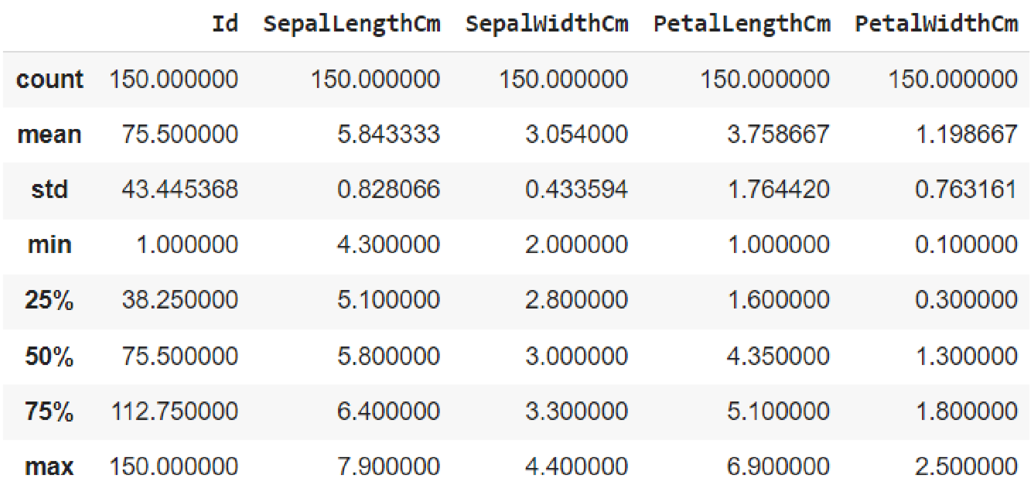


	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa



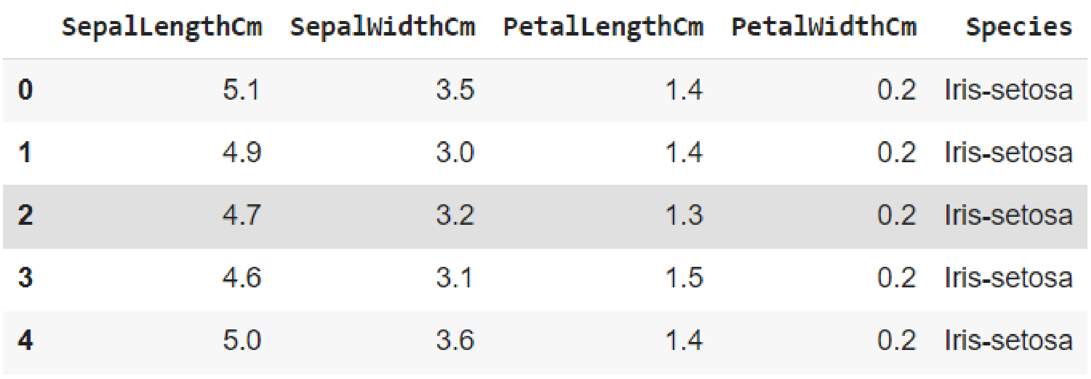
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

- Performing the describe function for the dataset



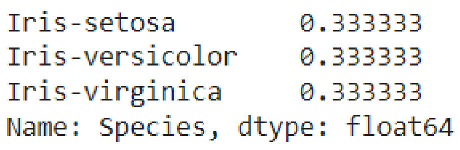
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

- Checking the empty values and we found that there are no empty values
- Since there is no use for the Id column, we can drop it
- After dropping, doing head() on the dataset

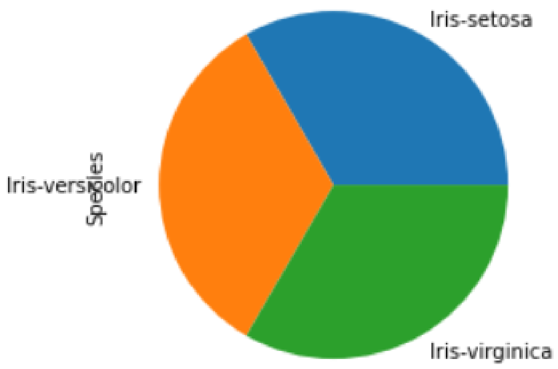


	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

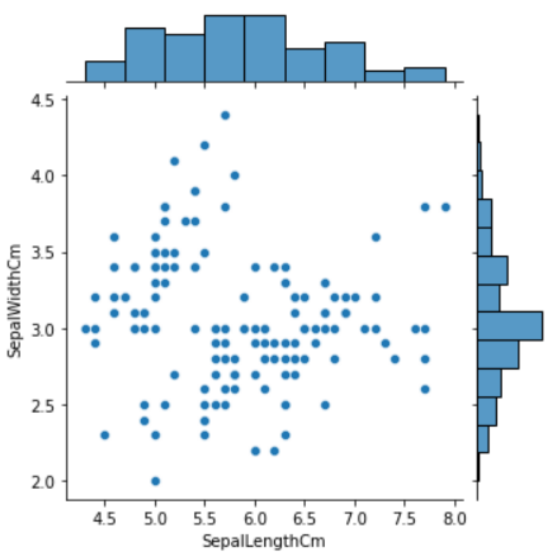
- Since there are no empty values and all the values are integers, there is no need to perform any preprocessing and visualising the classes and plotting the pie chart

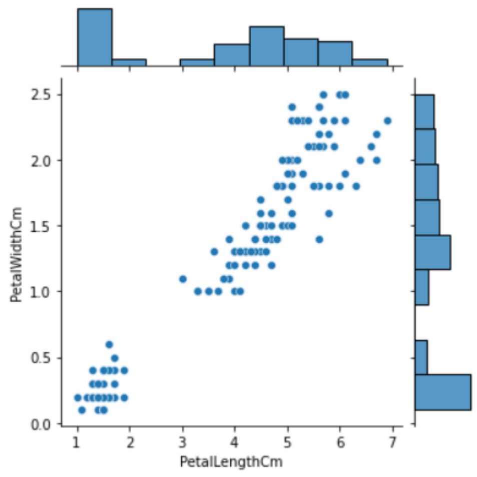


```
Iris-setosa        0.333333
Iris-versicolor    0.333333
Iris-virginica     0.333333
Name: Species, dtype: float64
```

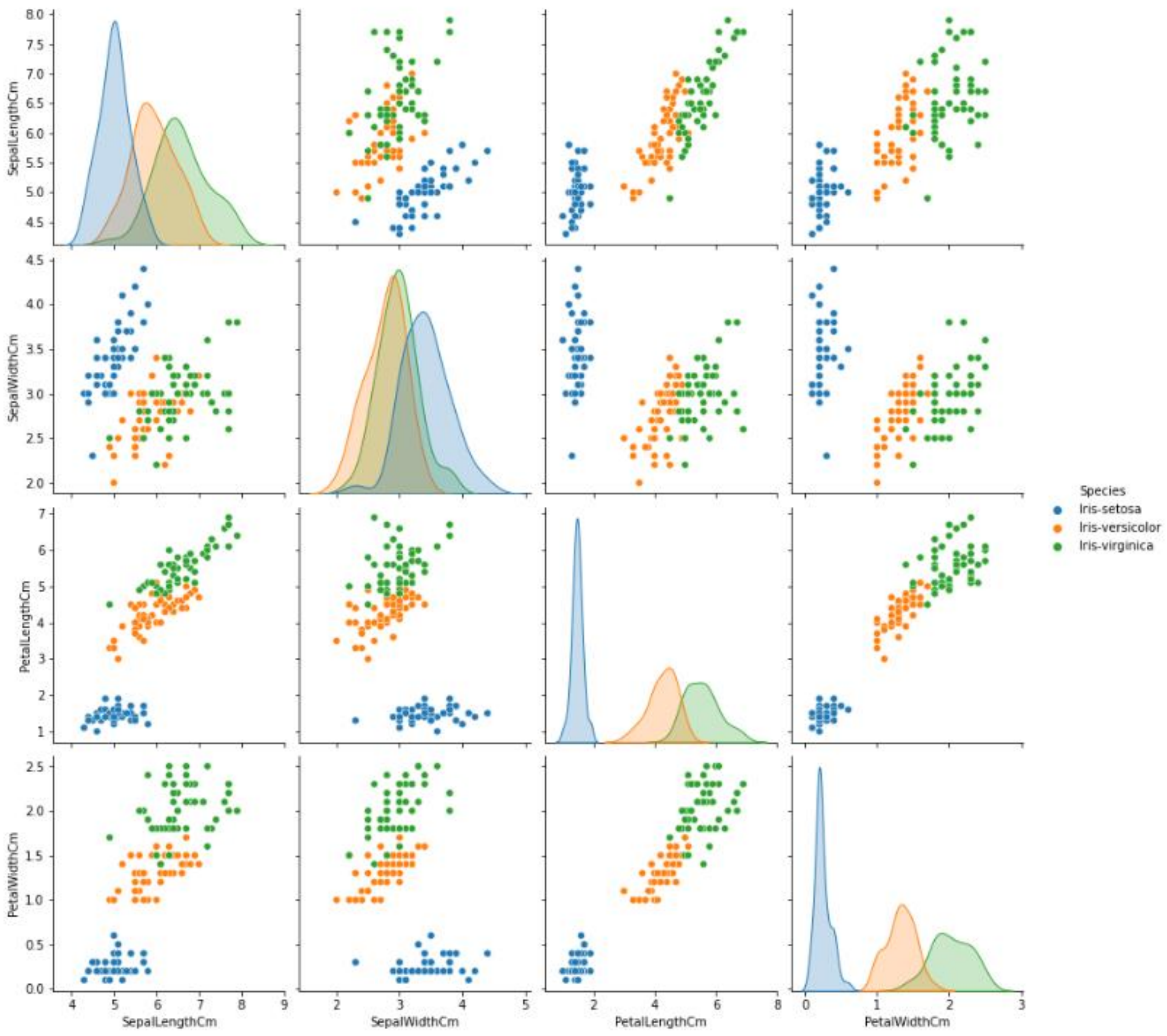


- Plotting bivariate scatterplots and univariate histograms between SepalWidth and SepalLength and plotting bivariate scatterplots and univariate histograms between PetalWidth and PetalLength





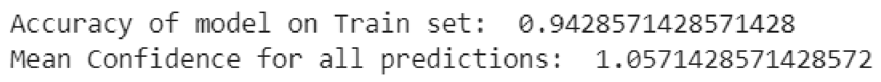
- Performing exploratory analysis on the dataset



- Relacing the values of species Iris-setosa with 0, Iris-versicolor with 1, and Iris-virginica with 2
- Splitting the dataset into train and test with 70:30 ratio

Implementing Gaussian Bayes Classification from Scratch

- A class was declared to implement the necessary function and to implement the Gaussian Bayes Classification from scratch
- The class has the following methods
    - Constructor __init__ used to define all the variables used in the class
    - fit() used for training the model and calculating the means and standard deviation for the classes
    - gauss() takes the standard deviation and mean and retuned the calculated values of the exponential part
    - predict() helps in predicting the class for the test set
    - train() calculates the accuracy of the predicted values with the actual values and returns the accuracy
- Using the gb variable and calling the class
- Implementing the class and printing the accuracy of the model and mean confidence for all predictions



```
Accuracy of model on Train set:  0.9428571428571428
Mean Confidence for all predictions:  1.0571428571428572
```

Implementing 5 fold Cross-Validation

- Using the kfold splitting the dataset into five parts
- By looping through each fold calculating the accuracy of the fold by the GaussianBayesClassifier class and storing them in an array
- Printing the calculated accuracies