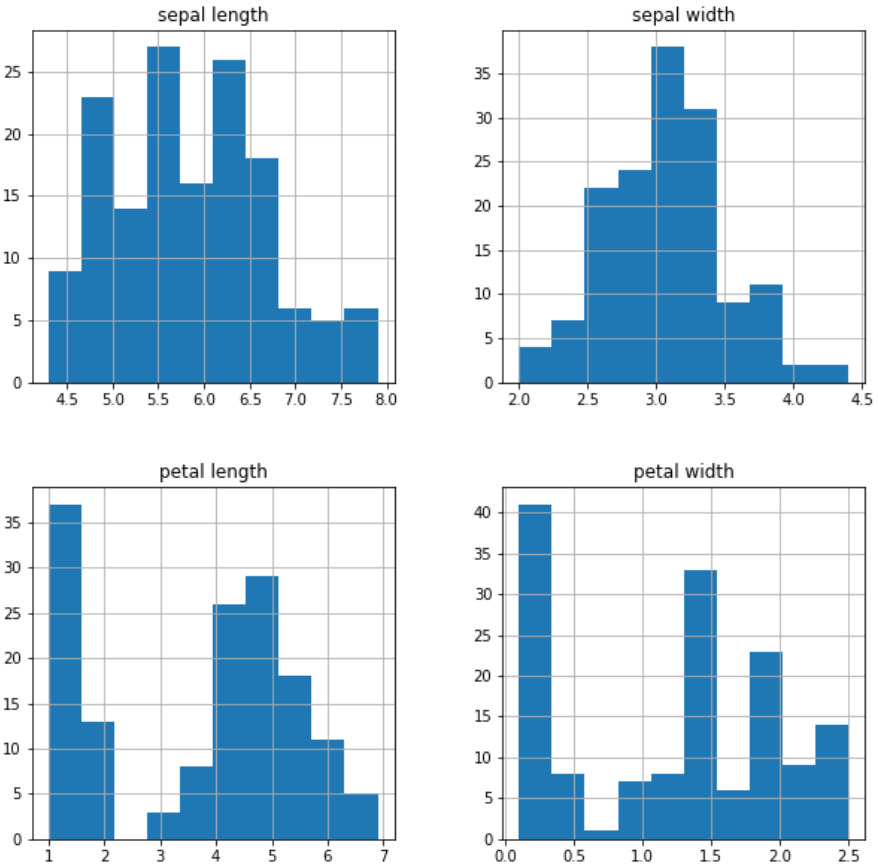
Pattern Recognition & Machine Learning
Lab-4 Assignment

Name: Tanish Pagaria
Roll No.: B21AI040

Question-1

Preprocessing and exploratory analysis of the data

The iris dataset did not have any NaN or NULL entries. Furthermore, because the data points were so close in range with each other, normalization was unnecessary. The labels were categorical and, hence, encoded.



Histogram showing the distribution of data points in each feature of the dataset

The dataset was then split into the training set and the testing set in the ratio 70:30 using the `sklearn.model_selection.train_test_split()` function.

Task-1

Implementation of Gaussian Bayes classifier from scratch

We considered the three cases while building the Gaussian Bayes Classifier from scratch and the `__init__` constructor of our `GaussianBayesClassifier()` class handled the type of case the classifier would be dealing with.

For the `train()` method of the class, we stored class-wise means, covariance matrices, and priors to be used later while making predictions.

We defined a discriminant function, `gi()`, for prediction and comparison. We removed the unnecessary part of the discriminant function equation, which is common for all classes.

Finally, the argmax of the `gi(x)` values was used to determine the most likely class for the tested set.

3 variants of the classifier

- **Case-1:** $\Sigma_i = \sigma^2 I$ (I stands for the identity matrix)
- **Case-2:** $\Sigma_i = \Sigma$ (covariance of all classes are identical but arbitrary)
- **Case-3:** $\Sigma_i = actual\ covariance$

The constructor of the class handled the three cases.

(Assumptions: in Case-1 and Case-2, the covariance matrices were taken based on the overall covariance matrix of all the features of the training dataset and adjusted accordingly to satisfy the cases.)

Task-2

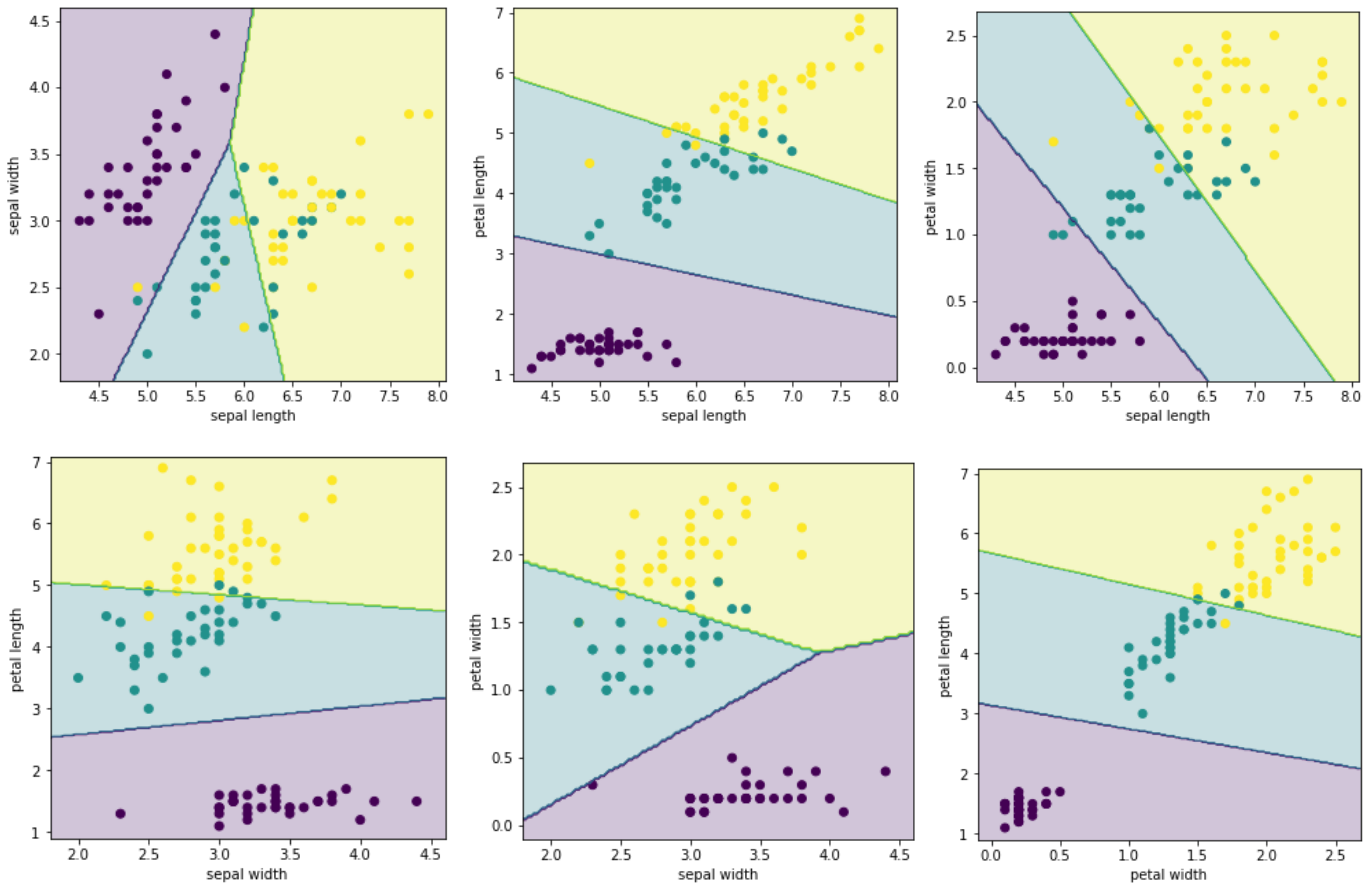
As mentioned previously, the Gaussian Bayes Classifier class has the following methods:

- `train()`: takes in the training data and trains the model
- `test()`: takes in the testing data and outputs the predictions and accuracy
- `predict()`: takes in a single data point and outputs the predicted class
- `plot_decision_boundary()`: takes the input of the training data and plots the decision boundary of the model (considering only 2 features)

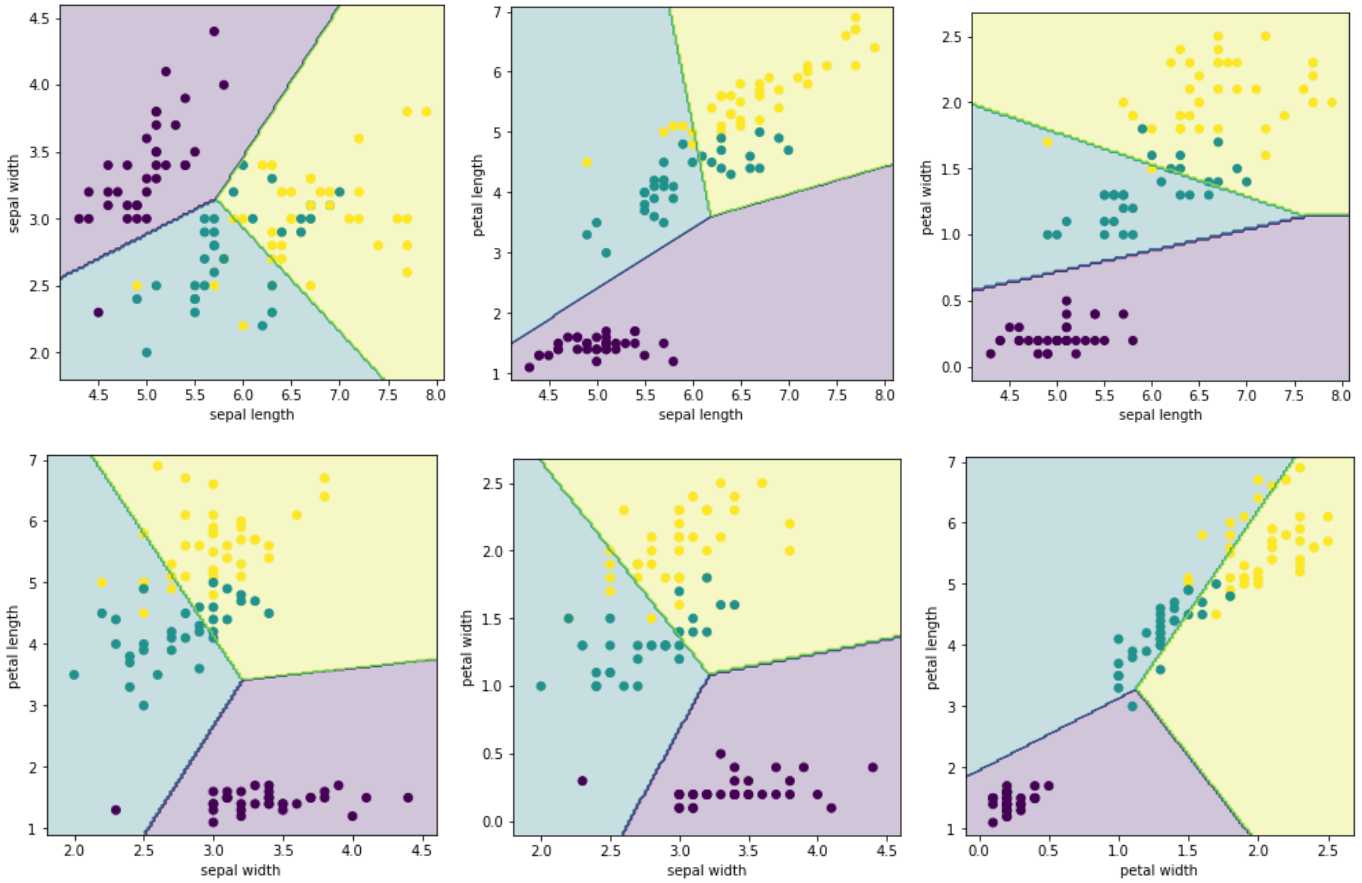
Other methods apart from the ones mentioned above are also present that aid in other parts of the classifier class, like the `gi()` method and the `accuracy()` method.

Task-3

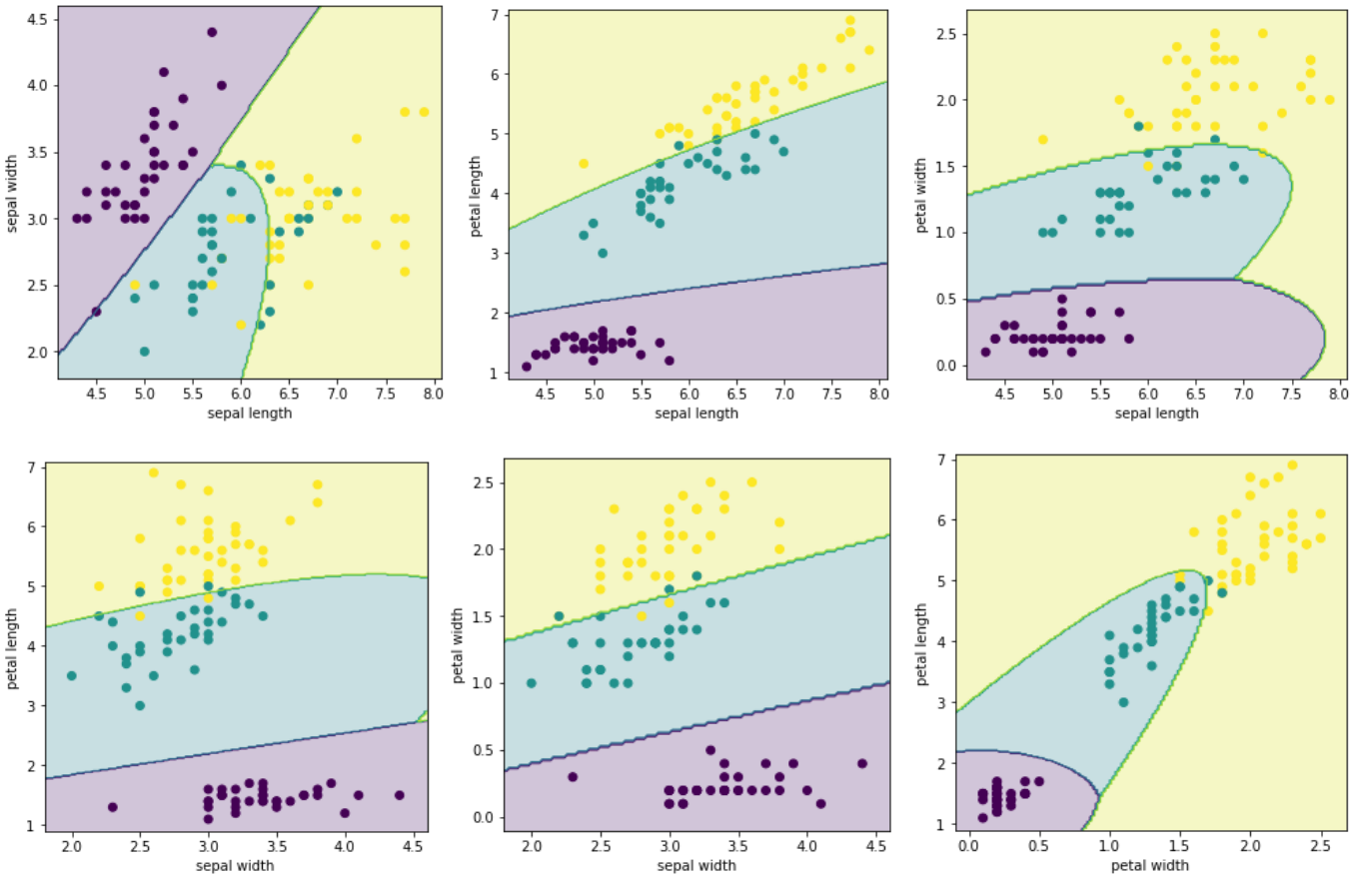
Plotting the decision boundary for each case on the training dataset



Decision Boundary for Case-1



Decision Boundary for Case-2



Decision Boundary for Case-3

Comparision among the 3 models

As is visible from the decision boundary plots shown above, the model performs the best in case-3. We are also aware that the discriminant function is quadratic in case-3 (also observable from the plots), and hence, in this case, the classifier predicts the classes better than in any other case.
In case-1 and case-2, the decision boundary plots are linear (as expected). The classifier gives the least performance in case-2.

Performance Comparision Results: *Case-3 > Case-1 > Case-2*

Task-4

Performing 5-fold cross-validation on the training dataset

It was performed using the built-in sklearn.model_selection.KFold() function. We defined the following to execute the same based on the three different cases.

```
def cross_validation(X, y, case):
    kf = KFold(n_splits=5, random_state=0, shuffle=True)
    model = GaussianBayesClassifier(case=case)

    accuracy_scores = []
    for train_index , test_index in kf.split(X):
        train_index = train_index.tolist()
        test_index = test_index.tolist()
        train_X, test_X = X.loc[train_index], X.loc[test_index]
        train_y, test_y = y[train_index], y[test_index]

        model.train(train_X, train_y)
        accuracy_scores.append(model.accuracy(test_X , test_y))

    print(f'For Case-{case}:')
    print('Accuracy Scores:', accuracy_scores)
    accuracy_scores = np.array(accuracy_scores)
    print('Mean Accuracy Score:', accuracy_scores.mean())
    print()
```

Accuracy Report

The following accuracy scores were obtained upon the 5-fold cross-validation:

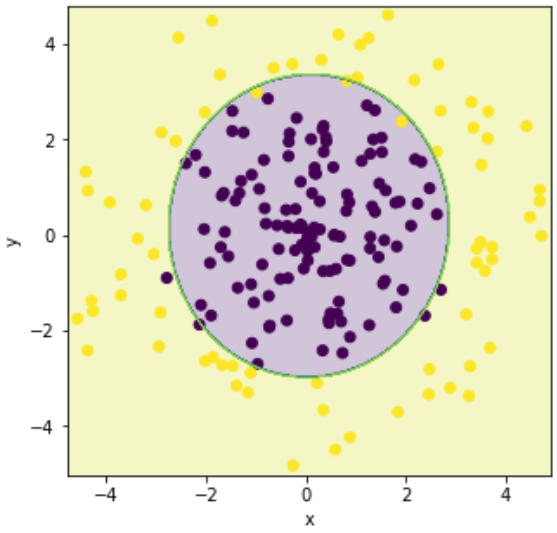
```
For Case-1:
Accuracy Scores: [0.9, 0.8, 0.9666666666666667, 0.9333333333333333, 0.9666666666666667]
Mean Accuracy Score: 0.9133333333333334

For Case-2:
Accuracy Scores: [0.8666666666666667, 0.7333333333333333, 0.9666666666666667, 0.9, 0.8666666666666667]
Mean Accuracy Score: 0.8666666666666668

For Case-3:
Accuracy Scores: [1.0, 0.9333333333333333, 0.9666666666666667, 1.0, 0.9333333333333333]
Mean Accuracy Score: 0.9666666666666666
```

Generalizability of the model

From the accuracy scores obtained above, we observe that the least accuracy was obtained in case-2. The model is well-generalized for case-3, as is evident from the high accuracy scores obtained in the cross-validation. Also, for case-3, the model is not overfitting the dataset.
However, in case-1 and case-2, the model cannot be called well-generalized. And, the model in case-2 is even less fitting the data in comparison with when in case-1.

Task-5

We created a synthetic dataset of points from a circular distribution $x^2 + y^2 = 25$ using the random module and the trigonometric identity $sin^2\theta + cos^2\theta = 1$.
(200 random points were taken from the distribution to create the dataset.)
The euclidean distances of the data points from the origin were compared for segregation of the points into 2 classes:-
Class = 1 → points having distance <= 3
Class = 2 → points having distance > 3 & distance <= 5



Decision Boundary for the Gaussian Bayes Classifier trained on the synthetic dataset

QUESTION-2

Given,

$$\text{covariance matrix } \Sigma = [\,[\tfrac{3}{2}, \tfrac{1}{2}], [\tfrac{1}{2}, \tfrac{3}{2}]\,]$$

$$\mu = [0, 0]$$

Sampling random points from the multivariate normal distribution

Performed using `numpy.random.multivariate_normal()`. The set of data points was saved as X. (We took 500 data points randomly in this case, the number could have been varied according to requirements.)

Calculating the covariance matrix of the sample X

We defined a function from scratch to calculate the covariance matrix of the sample X.

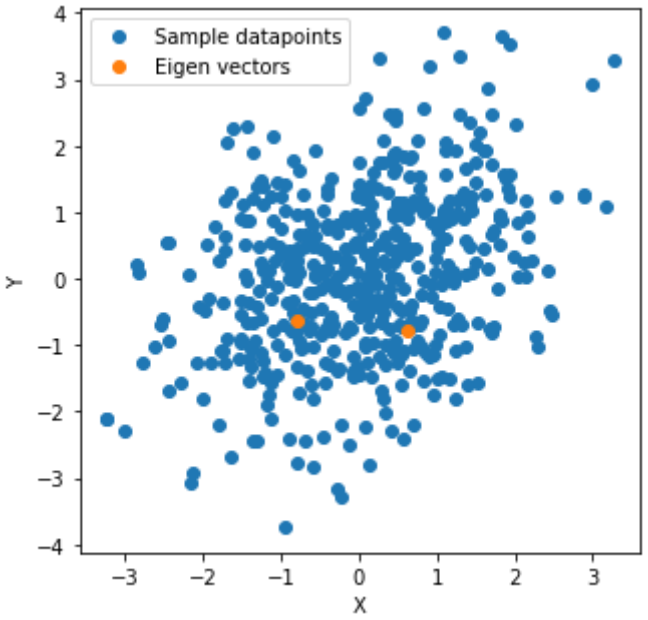
```
def covariance(x, y):
    n = len(x)
    xmean = np.mean(x)
    ymean = np.mean(y)
    return np.sum(np.multiply((x-xmean),(y-ymean)))/(n-1)
```

```
def covariance_matrix(x,y):
    return np.array([[covariance(x,x), covariance(x,y)], [covariance(y,x), covariance(y,y)]])
```

The covariance matrix was saved as Σ_S.

Finding eigenvalues and vectors of Σ_S and plotting it superimposed on the datapoints X

Eigenvectors and eigenvalues of the covariance matrix were calculated directly using `numpy.linalg.eig()`.



Plotting the eigenvectors of Σ_S on the data points X

Performing transformation $Y=\Sigma_S^{(-1/2)}X$ on data points X and calculating covariance matrix of the transformed data points

Fractional power was performed using the `scipy.linalg.fractional_matrix_power()` function. Then, the covariance matrix was calculated using the implemented covariance matrix function.

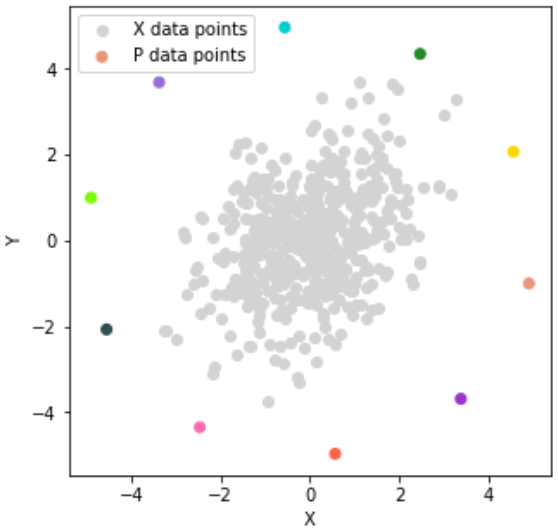
Observations:-

- The obtained covariance matrix is approximately equal to the identity matrix.

- The purpose of the transformation was to simplify the data points so that while calculating the discriminant function for normal density, we can use the simplified covariance matrix expression that acts almost as an identity matrix, so we can ease the calculation part.

Uniformly sampling 10 points on the curve $x^2 + y^2 = 25$

To take the points on the circle uniformly, we took a random angle in the interval [0, 2π) and then took 9 other points, each at an angle of 36° from the previous one, starting from the point we took in the beginning, each point being at a distance of 5 units from the origin (0,0).



Plot of data points in P along with the data points in X

Plotting Euclidean distance of each point from μ using barplot

The euclidean distance of each point was determined by defining a function based on the formula shown below.

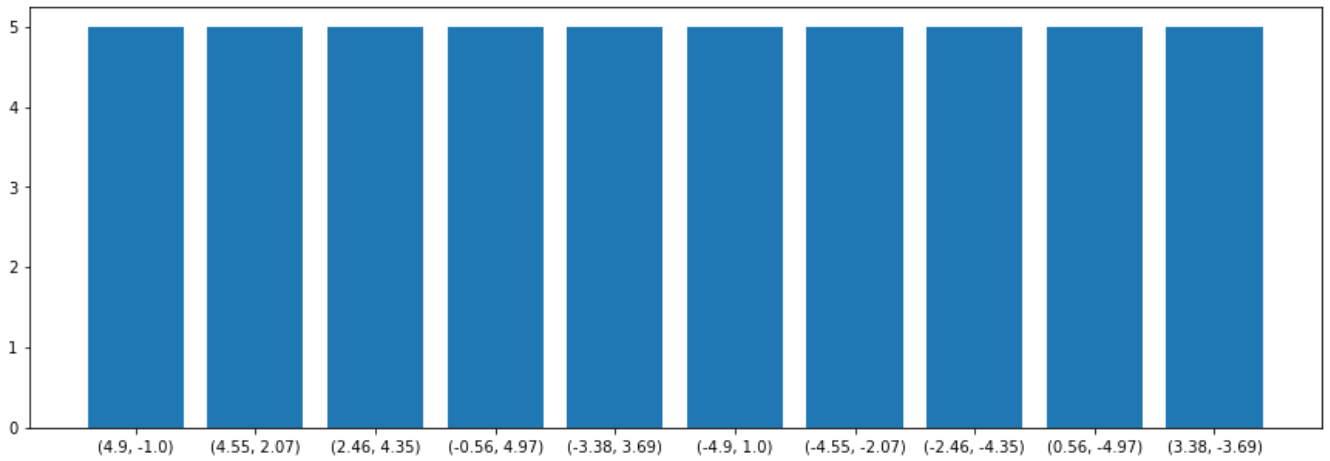
$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

p, q = two points in Euclidean n-space

q_i, p_i = Euclidean vectors, starting from the origin of the space (initial point)

n = n-space

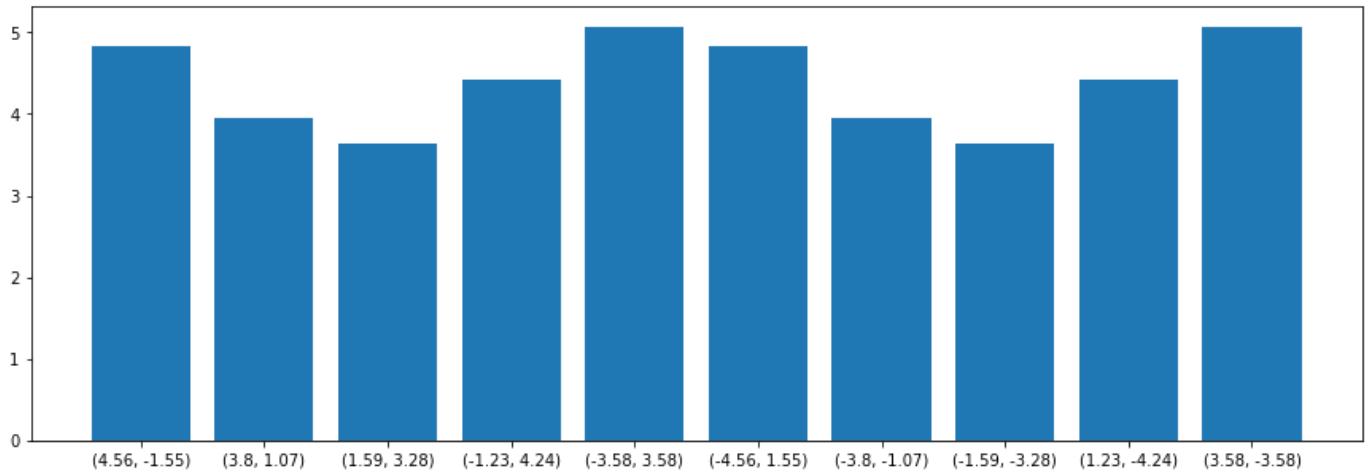
The euclidean distance of each point was 5 (as expected from the fact that they all lie on a circle of radius 5). The bar plot was plotted using `matplotlib.pyplot.bar()`.



Bar plot representing the euclidean distances of the points in P from μ

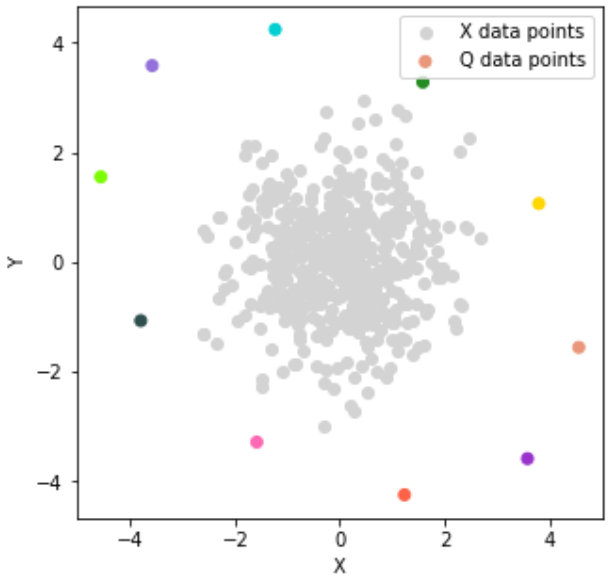
Performing the transformation $Q=\Sigma_S^{(-1/2)}X$ on the datapoints P

The transformation was performed similarly to what had been done previously in the case of Y. The euclidean distances of the transformed data points Q from μ are shown below (using barplot).



Bar plot representing the euclidean distances of the transformed data points in Q from μ

Plotting points in Q along with data points in Y



Plot of data points in Q along with the data points in Y

The euclidean distances before the transformation were equal for each point in the dataset P, and they all lay on a circle. However, after the transformation, their trajectory looks like an elliptical figure. The bar plots also show that the euclidean distances of some of the points have decreased, but they follow some pattern, and hence, an elliptical shape is visible in the plot.