



# prompt: connect google drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import pandas as pd
# Load the Iris dataset from the provided CSV file
iris_data = pd.read_csv('/content/drive/MyDrive/DataSets/iris.csv')
# Get unique species labels from the 'label' column
unique_species = iris_data['label'].unique()
# Dictionary to store mean vectors for each species
mean_vectors = {}
# Calculate mean vectors for each unique species
for species in unique_species:
    # Filter the dataset for the current species
    filtered_data = iris_data[iris_data['label'] == species]
    # Calculate mean values for Sepal length, Sepal width, Petal length, and Petal width
    mean_sepal_length = filtered_data['Sepal length'].mean()
    mean_sepal_width = filtered_data['Sepal width'].mean()
    mean_petal_length = filtered_data['Petal length'].mean()
    mean_petal_width = filtered_data['Petal width'].mean()
    # Create the mean vector
    mean_vector = [mean_sepal_length, mean_sepal_width, mean_petal_length, mean_petal_width]
    # Store the mean vector in the dictionary with species as the key
    mean_vectors[species] = mean_vector
# Print the mean vectors for all unique species
for species, mean_vector in mean_vectors.items():
    print(f"Mean Vector for {species}: {mean_vector}")
```

Mean Vector for Iris-setosa: [5.006, 3.418, 1.464, 0.244]  
Mean Vector for Iris-versicolor: [5.936, 2.7700000000000005, 4.26, 1.3259999999999998]  
Mean Vector for Iris-virginica: [6.587999999999998, 2.974, 5.5520000000000005, 2.0260000000000002]

iris\_data

	Sepal length	Sepal width	Petal length	Petal width	id	label	
0	5.1	3.5	1.4	0.2	id_1	Iris-setosa	
1	4.9	3.0	1.4	0.2	id_2	Iris-setosa	
2	4.7	3.2	1.3	0.2	id_3	Iris-setosa	
3	4.6	3.1	1.5	0.2	id_4	Iris-setosa	
4	5.0	3.6	1.4	0.2	id_5	Iris-setosa	
...	...	...	...	...	...	...	
145	6.7	3.0	5.2	2.3	id_146	Iris-virginica	
146	6.3	2.5	5.0	1.9	id_147	Iris-virginica	
147	6.5	3.0	5.2	2.0	id_148	Iris-virginica	
148	6.2	3.4	5.4	2.3	id_149	Iris-virginica	
149	5.9	3.0	5.1	1.8	id_150	Iris-virginica	

150 rows x 6 columns

Next steps: [View recommended plots](#)

### 1. Eucledian Distance

This code calculates the Euclidean distances between mean vectors of different species in a dataset. It utilizes `scipy.spatial.distance` to compute distances and `itertools.combinations` to generate pairs of species. The results are stored in a table and printed for easy comparison.

```
from scipy.spatial.distance import euclidean
from itertools import combinations
# Compute Euclidean distances between all pairs of species mean vectors
distance_table = pd.DataFrame(index=unique_species, columns=unique_species)
for species1, species2 in combinations(unique_species, 2):
    distance = euclidean(mean_vectors[species1], mean_vectors[species2])
    distance_table.at[species1, species2] = distance
    distance_table.at[species2, species1] = distance
# Print the table of Euclidean distances between species mean vectors
print("\nTable of Euclidean Distances:")
print(distance_table)
```

```
Table of Euclidean Distances:
              Iris-setosa  Iris-versicolor  Iris-virginica
Iris-setosa           NaN             3.205175           4.752592
Iris-versicolor       3.205175             NaN             1.620489
Iris-virginica         4.752592             1.620489             NaN
```

### 2. Z-Score

This code processes the iris dataset by first standardizing the features using z-score standardization. Then, it computes the mean vectors for each species and calculates the Euclidean distance between all pairs of mean vectors. The distances are presented in a table for easy comparison, facilitating analysis of the dissimilarities between species based on their feature means.

```
import pandas as pd
from scipy.spatial.distance import pdist, squareform
from sklearn.preprocessing import StandardScaler
```

# Step 1: Read the data and filter out unnecessary columns

```
file_path = "/content/drive/MyDrive/DataSets/iris.csv"
```

```
# Read the CSV file
ds = pd.read_csv(file_path)
```

```
ds = ds.drop(columns=["id"]) # Drop the 'id' column

# Standardize the data using z-score standardization
scaler = StandardScaler()
ds_std = pd.DataFrame(scaler.fit_transform(ds.iloc[:, :-1]), columns=ds.columns[:-1])
ds_std['label'] = ds['label']

# Compute the mean vector for each species
mean_vectors = ds_std.groupby('label').mean()

# Compute the Euclidean distance between each pair of mean vectors
distances = pdist(mean_vectors, metric='euclidean')

# Report the distances in a table
distance_ds = pd.DataFrame(squareform(distances), columns=mean_vectors.index, index=mean_vectors.index)
print("Euclidean distance between species mean vectors (after z-score standardization):")
print(distance_ds)

      Euclidean distance between species mean vectors (after z-score standardization):
label      Iris-setosa  Iris-versicolor  Iris-virginica
label
Iris-setosa      0.000000      2.840756      3.952601
Iris-versicolor  2.840756      0.000000      1.494566
Iris-virginica   3.952601      1.494566      0.000000
```

3. Mahalobis Distance

This code computes Mahalanobis distances between pairs of species mean vectors in the iris dataset. It first calculates the covariance matrix for the entire dataset and then computes the inverse of this matrix. Using these, it iterates through pairs of species, calculating Mahalanobis distances between their mean vectors and storing the results in a table for comparison.

```
unique_labels = iris_data['label'].unique()
print(unique_labels)

      ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']

from scipy.spatial.distance import mahalanobis
from scipy.linalg import inv
# Calculate the covariance matrix for the entire dataset
cov_matrix = iris_data[['Sepal length', 'Sepal width', 'Petal length', 'Petal width']].cov()
# Compute the inverse of the covariance matrix
inv_cov_matrix = inv(cov_matrix)
# Compute Mahalanobis distances between all pairs of species mean vectors
mahalanobis_distance_table = pd.DataFrame(index=unique_species, columns=unique_species)
for species1, species2 in combinations(unique_species, 2):
    # Calculate Mahalanobis distance between mean vectors of species1 and species2
    mahalanobis_distance = mahalanobis(mean_vectors[species1], mean_vectors[species2], inv_cov_matrix)
    # Store the Mahalanobis distance in the distance table for both species1 to species2 and species2 to species1
    mahalanobis_distance_table.at[species1, species2] = mahalanobis_distance
    mahalanobis_distance_table.at[species2, species1] = mahalanobis_distance
# Print the table of Mahalanobis distances between species mean vectors
print("\nTable of Mahalanobis Distances:")
print(mahalanobis_distance_table)

      Table of Mahalanobis Distances:
      Iris-setosa  Iris-versicolor  Iris-virginica
Iris-setosa      NaN      1.844099      2.35485
Iris-versicolor  1.844099      NaN      1.29136
Iris-virginica   2.35485      1.29136      NaN
```

4. Cosine similarity

This code calculates cosine similarities between pairs of species mean vectors in the iris dataset. It utilizes the cosine function from scipy.spatial.distance to compute similarities and stores the results in a table for comparison. This enables the analysis of the degree of similarity between species based on their feature means.

```
from scipy.spatial.distance import cosine
# Compute cosine similarities between all pairs of species mean vectors
similarity_table = pd.DataFrame(index=unique_species, columns=unique_species)
for species1, species2 in combinations(unique_species, 2):
    # Calculate cosine similarity between mean vectors of species1 and species2
    similarity = 1 - cosine(mean_vectors[species1], mean_vectors[species2])
    # Store the similarity in the similarity table for both species1 to species2 and species2 to species1
    similarity_table.at[species1, species2] = similarity
    similarity_table.at[species2, species1] = similarity
# Print the table of cosine similarities between species mean vectors
print("\nTable of Cosine Similarities:")
print(similarity_table)

      Table of Cosine Similarities:
      Iris-setosa  Iris-versicolor  Iris-virginica
Iris-setosa      NaN      0.924848      0.888438
Iris-versicolor  0.924848      NaN      0.995713
Iris-virginica   0.888438      0.995713      NaN
```

5. Pearson Coreation

This code computes Pearson correlation coefficients between pairs of species mean vectors in the iris dataset. It utilizes the pearsonr function from scipy.stats to calculate correlations and then stores the coefficients in a table for comparison. This allows for the examination of linear relationships between species based on their feature means.

```
from scipy.stats import pearsonr
# Compute Pearson correlation between all pairs of species mean vectors
correlation_table = pd.DataFrame(index=unique_species, columns=unique_species)
for species1, species2 in combinations(unique_species, 2):
    # Calculate Pearson correlation between mean vectors of species1 and species2
    correlation, _ = pearsonr(mean_vectors[species1], mean_vectors[species2])
    # Store the correlation coefficient in the correlation table for both species1 to species2 and species2 to species1
    correlation_table.at[species1, species2] = correlation
    correlation_table.at[species2, species1] = correlation
# Print the table of Pearson correlations between species mean vectors
print("\nTable of Pearson Correlations:")
print(correlation_table)
```

Table of Pearson Correlations:			
	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	NaN	0.763854	0.618438
Iris-versicolor	0.763854	NaN	0.979489
Iris-virginica	0.618438	0.979489	NaN

6. You now have five tables. Explain the pros and cons of each of the measures for the purpose of quantifying how similar pairs of species are.

▼ Euclidean Distance:

**Advantages:** Euclidean distance is easy to learn and apply. It calculates a meaningful distance metric, with smaller values suggesting greater similarity.

**Cons:** The Euclidean distance is sensitive to the scale of the variables, which can be problematic when the variables have different units or scales. It does not account for relationships between variables, which can be significant in multivariate data. In high-dimensional spaces, Euclidean distances become less useful.

Z-Score

**Advantages:** Z-score standardization normalizes variables, making them comparable across different scales. This is especially handy when working with datasets including variables with different units or scales. Z-score standardization is less susceptible to outliers than other scaling approaches. Outliers can still influence the mean and standard deviation, but their impact is lessened.

**Cons:** After Z-score standardization, variables are no longer in their original units. This may make it more difficult to grasp the changed numbers, particularly in circumstances when the original units have specific significance. Z-score standardization presupposes that the data is normally distributed. If the data is sufficiently non-normal, the transformation might not be appropriate.

Mahalanobis distance

**Pros:** The Mahalanobis distance takes into account the covariance structure of the data, making it appropriate for datasets with correlated variables. It is less sensitive to variations in scale than Euclidean distance.

**Cons:** Calculating Mahalanobis distance requires inverting the covariance matrix, which can be computationally expensive. It assumes that the data follows a multivariate normal distribution, which is not necessarily true.

Cosine Similarity:

**Advantages:** Cosine similarity quantifies the angle between vectors, making it appropriate for high-dimensional areas where Euclidean distances may become meaningless. It is solely affected by the direction of vectors, not their magnitude. Cosine similarity is widely used in text mining and is ideal for high-dimensional data such as word embeddings.

**Cons:** Cosine similarity does not apply when variables have negative values or when negative correlations are significant. It only takes into account the direction of vectors, hence it may not be appropriate when the magnitude (or relative importance) of variables is critical.

Pearson Correlation:

**Advantage:** Pearson correlation measures linear relationships between variables and quantifies the strength and direction of that link. It is insensitive to linear transformations, making it appropriate for variables of varying scale.

**Cons:** Because Pearson correlation presupposes linearity, it may fail to properly represent nonlinear relationships between variables. It may be susceptible to outliers, particularly with small sample sizes. It presupposes that variables follow a normal distribution; if this assumption is broken, the correlation may not adequately reflect the relationship.

7. For each of the species, find if there are outliers in the species. For this purpose compare the 4-dimensional vector of any particular object (row in the csv file) with the 4-dimensional mean vector of the species. Use Mahalanobis Distance as the proximity measure. You will need to choose appropriate thresholds for discerning whether a data point is anomalous or not. Explain how you did this.

This code segment is designed for outlier detection using Mahalanobis distance and the chi-square distribution. It calculates the critical chi-square value based on a given significance level and degrees of freedom. Then, for each species, it computes Mahalanobis distances for each data point relative to the species' mean vector, identifying outliers beyond the threshold. Finally, it prints out the detected outliers for each species, aiding in the identification of potentially anomalous observations within the dataset.

```
from scipy.stats import chi2
# Degrees of freedom (number of dimensions)
degrees_of_freedom = 4
# Significance level (1 - alpha) for outlier detection
alpha = 0.45
# Calculate the Chi-Square critical value for the given alpha and degrees of freedom
chi2_critical_value = chi2.ppf(1 - alpha, degrees_of_freedom)
# Dictionary to store outliers for each species
outliers = {}
# Detect outliers for each species using Mahalanobis distance
for species, mean_vector in mean_vectors.items():
    # Calculate the inverse of the covariance matrix for the current species
    species_cov_matrix = iris_data[iris_data['label'] == species][['Sepal length', 'Sepal width', 'Pet
    inv_species_cov_matrix = inv(species_cov_matrix)
    # Calculate Mahalanobis distance for each data point of the current species
    distances = []
    for _, row in iris_data[iris_data['label'] == species][['Sepal length', 'Sepal width', 'Petal leng
    .iterrows():
```

```

        data_point = row.values
        mahalanobis_distance = mahalanobis(data_point, mean_vector, inv_species_cov_matrix)
        distances.append(mahalanobis_distance)
    # Determine outliers based on the threshold
    outliers[species] = iris_data[iris_data['label'] == species][distances > chi2_critical_value]
# Print outliers for each species
for species, outlier_data in outliers.items():
    if not outlier_data.empty: # Check if DataFrame is not empty
        print(f"Outliers for {species}:")
        print(outlier_data)

    Outliers for Iris-virginica:
      Sepal length  Sepal width  Petal length  Petal width      id \
118           7.7           2.6           6.9           2.3  id_119

      label
118  Iris-virginica
```

8. Same as 7 except use Euclidean distance as the proximity measure. Do not standardize the column vectors. You will need to choose appropriate thresholds for discerning whether a data point is anomalous or not. Explain how you did this.

This snippet implements outlier detection based on the Euclidean distance from the mean vector, utilizing quartiles and the interquartile range (IQR) to establish a threshold. It iterates over each species, calculates the distances for each data point, identifies outliers exceeding the threshold, and finally prints out the detected outliers for each species, aiding in the identification of potentially anomalous observations within the dataset.

```

outliers = {}
# Constant for determining the threshold (k times IQR)
k = 1.5
for species, mean_vector in mean_vectors.items():
    # Calculate Euclidean distance for each data point of the current species
    distances = []
    for _, row in iris_data[iris_data['label'] == species][['Sepal length', 'Sepal width', 'Petal length', 'Petal width']]\
        .iterrows():
        data_point = row.values
        euclidean_distance = euclidean(data_point, mean_vector)
        distances.append(euclidean_distance)
    # Reset the index of the DataFrame for proper alignment
    species_data = iris_data[iris_data['label'] == species].reset_index(drop=True)
    # Calculate quartiles and IQR
    q1 = pd.Series(distances).quantile(0.25)
    q3 = pd.Series(distances).quantile(0.75)
    iqr = q3 - q1
    # Determine outliers based on the threshold (Q3 + k * IQR)
    outliers[species] = species_data[pd.Series(distances) > (q3 + k * iqr)]
# Print outliers for each species
for species, outlier_data in outliers.items():
    print(f"Outliers for {species}:")
    print(outlier_data)

    Outliers for Iris-setosa:
      Sepal length  Sepal width  Petal length  Petal width      id      label
15           5.7           4.4           1.5           0.4  id_16  Iris-setosa
41           4.5           2.3           1.3           0.3  id_42  Iris-setosa
    Outliers for Iris-versicolor:
      Sepal length  Sepal width  Petal length  Petal width      id \
7           4.9           2.4           3.3           1.0  id_58
10          5.0           2.0           3.5           1.0  id_61
43          5.0           2.3           3.3           1.0  id_94
48          5.1           2.5           3.0           1.1  id_99

      label
7  Iris-versicolor
10 Iris-versicolor
43 Iris-versicolor
48 Iris-versicolor
    Outliers for Iris-virginica:
      Sepal length  Sepal width  Petal length  Petal width      id \
6           4.9           2.5           4.5           1.7  id_107
18          7.7           2.6           6.9           2.3  id_119

      label
6  Iris-virginica
18 Iris-virginica
```

9. Analyze your anomaly detection results. Does Mahalanobis distance work better than Euclidean distance? Or the other way around? Or the results are inconclusive?

The Mahalanobis distance might excel in tasks such as clustering (e.g., utilizing k-means) or classification (e.g., employing k-nearest neighbors) within the Iris dataset, given its capability to consider both scale and correlation among features. Nevertheless, the Euclidean distance could suffice and offer simplicity, particularly if the dataset has been normalized or if differences in scale and correlation are minimal. In essence, empirical testing would be the most reliable approach to ascertain which distance measure performs better. This entails evaluating the performance of each distance measure in specific tasks like clustering or classification using the Iris dataset.

10. Evaluate the efficacy of the following (nearest-neighbors- like) classifier. First calculate the mean vector of each species as described at the beginning of this assignment. Next, for each point in the data set (i.e. row in the csv file restricted to the first 4 columns) predict the species of that point to be the species of the mean vector that is the closest to that point. Use Mahalanobis Distance as the proximity measure. Report your results as a 3-by-3 contingency table T, where T[i][j] is the number of instances in which the true species is i and the predicted species is j. Attach a brief interpretation of what this table reveals about how well your classifier has done. Note that we are well aware that this evaluation is being done on the training set; its not meant for drawing conclusions about predictive accuracy, rather only to get an exploratory sense for what the contingency table reveals and what we can conclude from it.

This code generates predictions for the species of each data point using Mahalanobis Distance. It iterates through the dataset, calculates Mahalanobis distances from each species' mean vector, and predicts the species based on the closest mean vector. Finally, it constructs a 3x3 contingency table to evaluate the classification performance, presenting the counts of correct and incorrect predictions for each species pair.

```
predictions = {'Iris-setosa': [], 'Iris-versicolor': [], 'Iris-virginica': []}
# Predict species for each data point using Mahalanobis Distance
for _, row in iris_data[['Sepal length', 'Sepal width', 'Petal length', 'Petal width', 'label']].iterrows():
    data_point = row[['Sepal length', 'Sepal width', 'Petal length', 'Petal width']].values
    true_species = row['label']
    # Calculate Mahalanobis distances from each species' mean vector
    mahalanobis_distances = {}
    for species, mean_vector in mean_vectors.items():
        species_cov_matrix = iris_data[iris_data['label'] == species][['Sepal length', 'Sepal width', 'Petal length',
                                                                        'Petal width']].cov()

        inv_species_cov_matrix = inv(species_cov_matrix)
        mahalanobis_distance = mahalanobis(data_point, mean_vector, inv_species_cov_matrix)
        mahalanobis_distances[species] = mahalanobis_distance
    # Predict the species based on the closest mean vector
    predicted_species = min(mahalanobis_distances, key=mahalanobis_distances.get)
    # Store the prediction
    predictions[true_species].append(predicted_species)
# Create the 3x3 contingency table
contingency_table = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
for i, true_species in enumerate(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']):
    for j, predicted_species in enumerate(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']):
        contingency_table[i][j] = predictions[true_species].count(predicted_species)
# Print the contingency table
print("Contingency Table:")
for row in contingency_table:
    print(row)

Contingency Table:
[50, 0, 0]
[0, 47, 3]
[0, 0, 50]
```