```
#find s
import pandas as pd
def find_s_algorithm(data):
    hypothesis = None
    for _, row in data.iterrows():
        if row[-1] == 'yes':
            if hypothesis is None:
                hypothesis = row[:-1].values
            else:
                for i in range(len(hypothesis)):
                    if hypothesis[i] != row[i]:
                        hypothesis[i] = '?'
        print(hypothesis)
    return hypothesis
data = pd.read_csv('/content/data.csv')
specific_hypothesis = find_s_algorithm(data)
print("Most specific hypothesis:", specific_hypothesis)
```

```
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' '?' '?']
Most specific hypothesis: ['sunny' 'warm' '?' 'strong' '?' '?']
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_c
  and should_run_async(code)
```

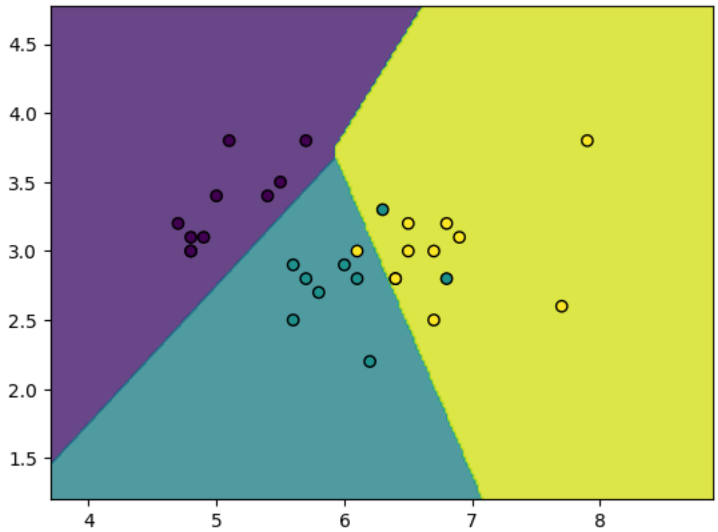
```
#candidate elimination
import pandas as pd
def candidate_elimination(data):
    S, G = ['0'] * (data.shape[1] - 1), [['?'] * (data.shape[1] - 1)]
    print(f"Initial Specific Boundary S: {S}")
    print(f"Initial General Boundary G: {G}\n")
    for i, row in data.iterrows():
        x, y = row[:-1], row[-1]
        print(f"Instance {i+1}: ")

        if y == 'yes':
            S = [xi if si == '0' else si if si == xi else '?' for si, xi in zip(S, x)]
            G = [g for g in G if all (g[i] in ('?', xi) for i, xi in enumerate(x))]
        else:
            G_new = []
            for g in G:
                if all(g[i] in ('?', xi) for i, xi in enumerate(x)):
                    for i in range(len(x)):
                        if g[i] == '?':
                            for val in set(data.iloc[:, i]):
                                if val != x[i]:
                                    new_g = g[:]
                                    new_g[i] = val
                                    if any(all(new_g[j] in ('?', ex[j]) for j in range(len(new_g))) for ex in data[data.iloc[:, -1] == 'y
                                        G_new.append(new_g)
                else:
                    G_new.append(g)
            G = G_new

        print(f"Updated Specific Boundary S: {S}")
        print(f"Updated General Boundary G: {G}\n")

    return S, G

data = pd.read_csv('/content/data.csv')
S, G = candidate_elimination(data)
print("Final Specific Boundary:", S)
print("Final General Boundary:", G)
```

```
Initial Specific Boundary S: ['0', '0', '0', '0', '0', '0']
Initial General Boundary G: [['?', '?', '?', '?', '?', '?']]

Instance 1:
Updated Specific Boundary S: ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
Updated General Boundary G: [['?', '?', '?', '?', '?', '?']]

Instance 2:
Updated Specific Boundary S: ['sunny', 'warm', '?', 'strong', 'warm', 'same']
Updated General Boundary G: [['?', '?', '?', '?', '?', '?']]

Instance 3:
Updated Specific Boundary S: ['sunny', 'warm', '?', 'strong', 'warm', 'same']
Updated General Boundary G: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', 'normal', '?', '?',
```

```
Instance 4:
Updated Specific Boundary S: ['sunny', 'warm', '?', 'strong', '?', '?']
Updated General Boundary G: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', 'cool', '?

Final Specific Boundary: ['sunny', 'warm', '?', 'strong', '?', '?']
Final General Boundary: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', 'cool', '?']]
```

```python
#linear svm
#!pip install scikit-learn matplotlib

from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import numpy as np

# Load and prepare data
iris = datasets.load_iris()
X = iris.data[:, :2]  # Use only the first two features for simplicity
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#train svm
svm = SVC(kernel='linear')
svm.fit(X_train, y_train)

# Evaluate
print("Accuracy:", accuracy_score(y_test, svm.predict(X_test)))

# Plot decision boundaries
def plot_decision_boundaries(X, y, model):
    h = .02
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k')
    plt.show()

plot_decision_boundaries(X_test, y_test, svm)

new_point = svm.predict([[5.5, 2.0]])
print("Prediction for new point [5.5, 2.0]:", new_point)
```

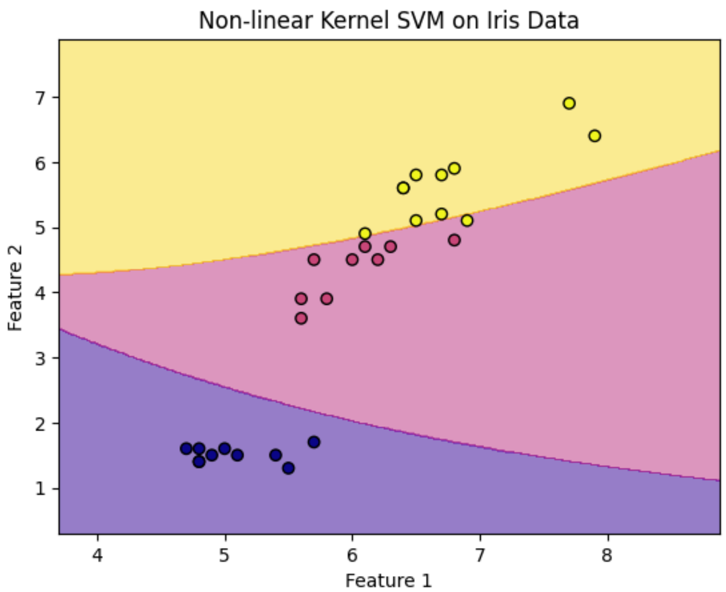
```
Accuracy: 0.9
```




```
Prediction for new point [5.5, 2.0]: [1]
```

```
#non linear svm
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris = load_iris()
X = iris.data[:, [0, 2]]
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

svm = SVC(kernel='poly')
svm.fit(X_train, y_train)
accuracy=accuracy_score(y_test, svm.predict(X_test))
print("Accuracy is: ", accuracy)

def plot_decision_boundaries(X, y, model):
    h = .02
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.5, cmap='plasma')
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap='plasma')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('Non-linear Kernel SVM on Iris Data')
    plt.show()

plot_decision_boundaries(X_test, y_test, svm)

new_point = svm.predict([[5.5, 2.0]])
print("Prediction for new point [5.5, 2.0]:", new_point)
```

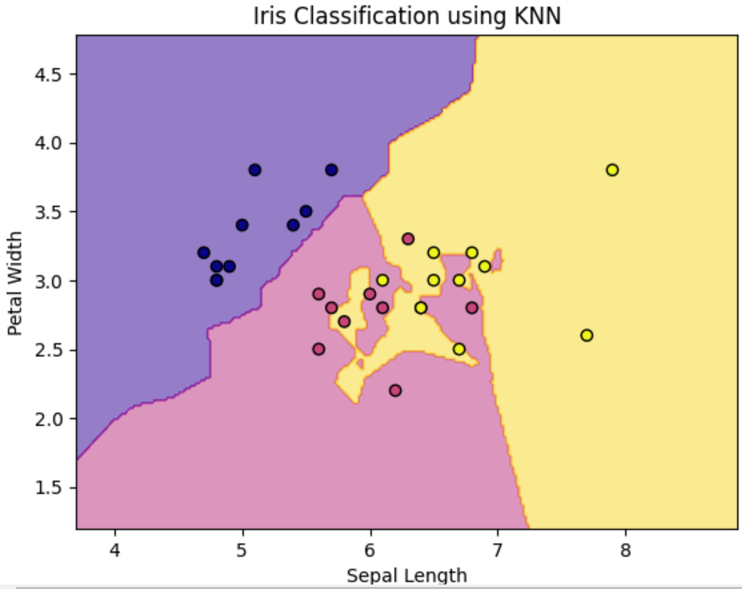
```
Accuracy is:  0.9666666666666667
```



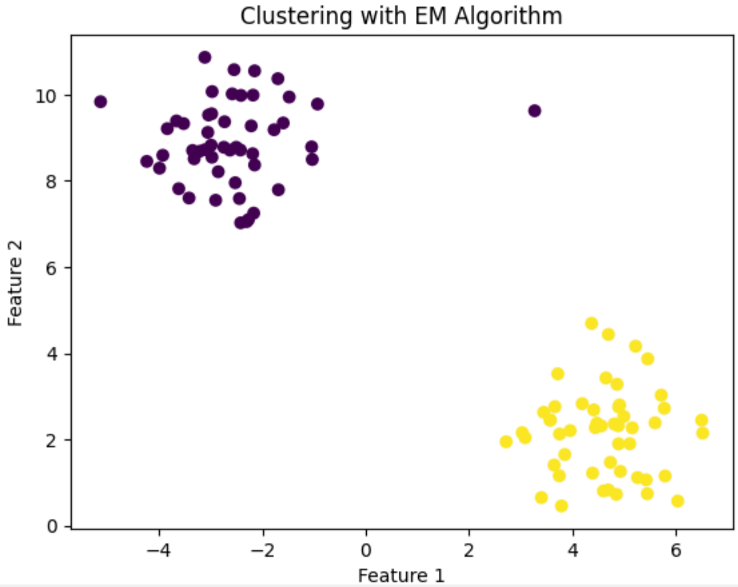

```
Prediction for new point [5.5, 2.0]: [0]
```

```
#knn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

iris = load_iris()
X, y = iris.data[:, :2], iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
accuracy = knn.score(X_test, y_test)
print("Accuracy:", accuracy)
print("Prediction for [1.4, 5.3]:", knn.predict([[1.4, 5.3]]))

def plot_decision_boundaries(X, y, model, title='Decision Boundary'):
    h = .02
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.5, cmap='plasma')
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap='plasma')
    plt.xlabel('Sepal Length')
    plt.ylabel('Petal Width')
    plt.title(title)
    plt.show()

plot_decision_boundaries(X_test, y_test, knn, title='Iris Classification using KNN')
```

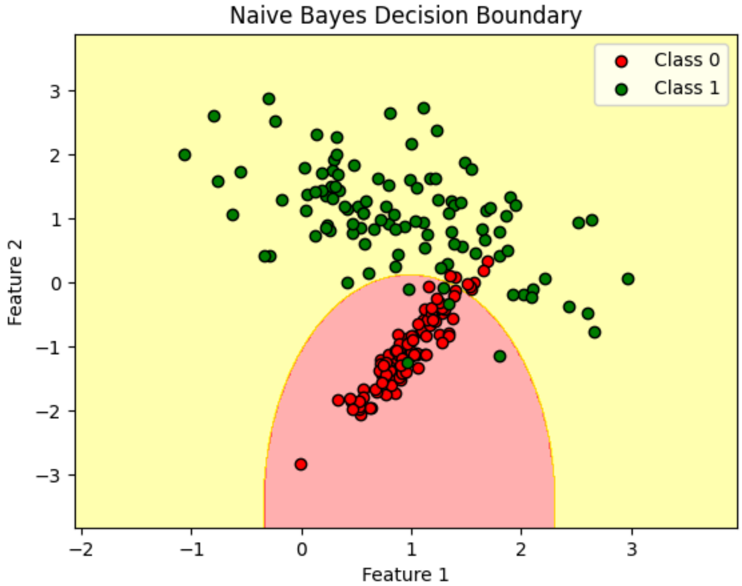
```
Accuracy: 0.8333333333333334
Prediction for [1.4, 5.3]: [0]
```



```
#em
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.datasets import make_blobs

# Generate synthetic data
X, _ = make_blobs(n_samples=100, centers=2, cluster_std=1.0, random_state=42)

# Randomly introduce missing values
X[np.random.choice(X.shape[0], size=1, replace=False), 0] = np.nan

# Handle missing values by mean imputation
X_imputed = np.nan_to_num(X, nan=np.nanmean(X))

# Apply EM algorithm using Gaussian Mixture Model
gmm = GaussianMixture(n_components=2, random_state=42)
gmm.fit(X_imputed)
labels = gmm.predict(X_imputed)

plt.scatter(X_imputed[:, 0], X_imputed[:, 1], c=labels, cmap='viridis', marker='o')
plt.title('Clustering with EM Algorithm')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



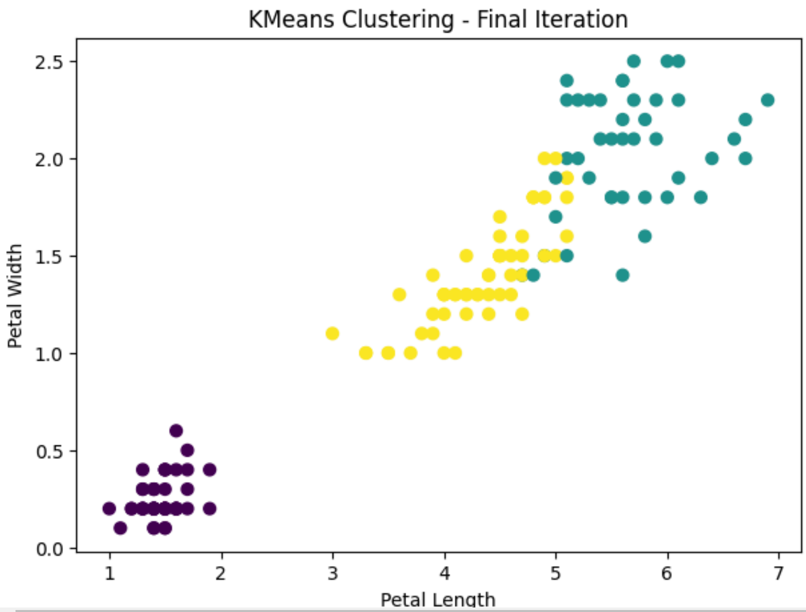
```
#naive bayes
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.datasets import make_classification
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

X, y = make_classification(n_samples=300, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred = nb.predict(X_test)

def plot_decision_boundaries(X, y, model, title='Decision Boundary'):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.3, cmap=ListedColormap(('red', 'yellow')))
    for i, color in enumerate(['red', 'green']):
        plt.scatter(X[y == i, 0], X[y == i, 1],
                    c=color, label=f'Class {i}', edgecolors='k')
    plt.title('Naive Bayes Decision Boundary')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.show()

plot_decision_boundaries(X_train, y_train, nb)

cm = confusion_matrix(y_test, y_pred)
print(cm)
```




```
[[45  1]
 [ 1 43]]
```

```
#k means
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

dataset = load_iris()
X = pd.DataFrame(dataset.data, columns=['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])

initial_centroids = np.array([
    [5.0, 3.5, 1.4, 0.2],
    [6.5, 3.0, 5.5, 2.0],
    [5.5, 2.5, 4.0, 1.3]
])

n_clusters = 3
iterations = 5

for i in range(iterations):
    kmeans = KMeans(n_clusters=n_clusters, init=initial_centroids, n_init=1, max_iter=1, random_state=42)
    kmeans.fit(X)
    print(f"Iteration {i+1}:")
    print(f"Cluster Centers:\n{kmeans.cluster_centers_}\n")
    initial_centroids = kmeans.cluster_centers_

plt.figure(figsize=(7, 5))
plt.scatter(X['Petal_Length'], X['Petal_Width'], c=kmeans.labels_, cmap='viridis', s=40)
plt.title('KMeans Clustering - Final Iteration')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
```

```
Iteration 1:
Cluster Centers:
[[5.006      3.428      1.462      0.246     ]
 [6.61129032 2.9983871  5.39032258 1.92258065]
 [5.69210526 2.66578947 4.11578947 1.27368421]]

Iteration 2:
Cluster Centers:
[[5.006      3.428      1.462      0.246     ]
 [6.63220339 2.99830508 5.43050847 1.93728814]
 [5.72926829 2.6902439  4.15121951 1.3       ]]

Iteration 3:
Cluster Centers:
[[5.006      3.428      1.462      0.246     ]
 [6.66481481 3.00740741 5.5        1.96851852]
 [5.78913043 2.71304348 4.20869565 1.3326087 ]]

Iteration 4:
Cluster Centers:
[[5.006 3.428 1.462 0.246]
 [6.702 3.016 5.556 1.992]
 [5.822 2.728 4.256 1.36 ]]

Iteration 5:
Cluster Centers:
[[5.006      3.428      1.462      0.246     ]
 [6.76956522 3.03695652 5.6        2.00869565]
 [5.82962963 2.73148148 4.31481481 1.39259259]]
```




```
#Apriori
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Create the dataset
dataset = [
    ['Milk', 'Bread', 'Butter'],
    ['Bread', 'Butter'],
    ['Beer', 'Cookies', 'Diaper'],
    ['Milk', 'Diaper', 'Butter', 'Bread'],
    ['Beer', 'Diaper']
]

# Convert to DataFrame and one-hot encode
df = pd.DataFrame(dataset)
df = df.stack().str.get_dummies().groupby(level=0).sum()

# Apply Apriori algorithm
frequent_itemsets = apriori(df, min_support=0.5, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

# Display results
print("Frequent Itemsets:")
print(frequent_itemsets)
print("\nAssociation Rules:")
print(rules)
```

```
Frequent Itemsets:
   support         itemsets
0      0.6          (Bread)
1      0.6         (Butter)
2      0.6         (Diaper)
3      0.6  (Bread, Butter)

Association Rules:
  antecedents consequents  antecedent support  consequent support  support  \
0     (Bread)    (Butter)                 0.6                 0.6      0.6
1    (Butter)     (Bread)                 0.6                 0.6      0.6

   confidence      lift  leverage  conviction  zhangs_metric
0         1.0  1.666667      0.24         inf            1.0
1         1.0  1.666667      0.24         inf            1.0
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_c
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:109: DeprecationWarning: DataFrames with non-bool type
  warnings.warn(
```

```
#pca
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

dataset = load_iris()
X = pd.DataFrame(dataset.data, columns=dataset.feature_names)
y = dataset.target

X_meaned = X - np.mean(X, axis=0)
cov_matrix = np.cov(X_meaned, rowvar=False)
eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)
sorted_indices = np.argsort(eigenvalues)[::-1]
sorted_eigenvectors = eigenvectors[:, sorted_indices]

n_components = 2
principal_eigenvectors = sorted_eigenvectors[:, :n_components]
X_reduced = np.dot(principal_eigenvectors.transpose(), X_meaned.transpose()).transpose()

plt.figure(figsize=(8, 6))
for target in np.unique(y):
    plt.scatter(X_reduced[y == target, 0], X_reduced[y == target, 1], label=dataset.target_names[target])

plt.title('PCA of Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_c
```