CONFUSION MATRIX

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Sample true labels and predicted labels
y_true = np.array([0, 1, 1, 0, 1, 0, 1, 0, 0, 1])
y_pred = np.array([0, 0, 1, 0, 1, 1, 1, 0, 0, 1])

# Generate the confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted 0', 'Predicted 1'],
yticklabels=['True 0', 'True 1'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

DECISION TREE

```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import accuracy_score, confusion_matrix

# Load dataset
data = load_breast_cancer()
x, y = data.data, data.target

# Split data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

# Train the Decision Tree model (with limited depth)
model = DecisionTreeClassifier(max_depth=2).fit(x_train, y_train)

# Predict on test data
y_pred = model.predict(x_test)
```

```
# Print accuracy and confusion matrix
print(f"Accuracy Score: {accuracy_score(y_test, y_pred) * 100:.2f}%")
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Simplified plot of the Decision Tree
plt.figure(figsize=(12,8))  # Adjust the size of the plot
plot_tree(model, feature_names=data.feature_names, filled=True, rounded=True)
plt.title('Decision Tree')
plt.show()
```

RANDOM FOREST

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.tree import plot_tree

# Load the dataset
data = load_breast_cancer()
X, y = data.data, data.target

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the Random Forest model
model = RandomForestClassifier(random_state=42).fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Print accuracy and confusion matrix
print(f"Random Forest Accuracy: {accuracy_score(y_test, y_pred) * 100:.2f}%")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Plotting the first tree in the Random Forest
plt.figure(figsize=(12, 8))
plot_tree(model.estimators_[0], feature_names=data.feature_names, filled=True,
rounded=True)
plt.title("First Decision Tree of Random Forest")
plt.show()
```

```
# Plotting feature importances
feature_importances = model.feature_importances_
plt.figure(figsize=(10, 6))
plt.bar(range(len(feature_importances)), feature_importances)
plt.xlabel('Feature Index')
plt.ylabel('Feature Importance')
plt.title('Feature Importance')
plt.xticks(range(len(feature_importances)), data.feature_names, rotation=90)
plt.show()
```

SVM

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.decomposition import PCA

# Load the dataset and reduce dimensions
X, y = load_breast_cancer(return_X_y=True)
X_reduced = PCA(n_components=2).fit_transform(X)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_reduced, y, test_size=0.3, random_state=42)

# Train the SVM model
model = SVC(kernel='linear').fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Print accuracy and confusion matrix
print(f"SVM Accuracy: {accuracy_score(y_test, y_pred) * 100:.2f}%")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Function to plot the decision boundary
def plot_decision_boundary(X, y, model):
    xx, yy = np.meshgrid(np.linspace(X[:, 0].min()-1, X[:, 0].max()+1, 500),
                         np.linspace(X[:, 1].min()-1, X[:, 1].max()+1, 500))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.3, cmap='RdYlBu')
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap='RdYlBu')
    plt.title("SVM Decision Boundary")
    plt.xlabel("PC 1")
    plt.ylabel("PC 2")
    plt.show()

# Plot the decision boundary
plot_decision_boundary(X_test, y_test, model)
```

GRAPH BASED CLUSTERING

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.cluster import SpectralClustering

# Generate synthetic data
X, _ = make_moons(n_samples=200, noise=0.1)

# Apply Spectral Clustering
model = SpectralClustering(n_clusters=2, affinity='nearest_neighbors').fit(X)
labels = model.labels_

# Plot the results
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.title('Graph-Based Clustering (Spectral)')
plt.show()

# Print clustering information
unique_labels = np.unique(labels)
print(f"Number of Clusters Detected: {len(unique_labels)}")
for label in unique_labels:
    print(f"Size of Cluster {label}: {np.sum(labels == label)}")
```

DBSCAN

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.cluster import DBSCAN

# Generate synthetic data
X, _ = make_moons(n_samples=200, noise=0.1)

# Apply DBSCAN
```

```
model = DBSCAN(eps=0.2, min_samples=5).fit(X)
labels = model.labels_

# Plot the results
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.title('DBSCAN Clustering')
plt.show()

# Print clustering information
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
n_noise = list(labels).count(-1)
print(f"Number of Clusters Detected: {n_clusters}")
print(f"Number of Noise Points: {n_noise}")
```

PCA

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA

# Load dataset
data = load_iris()
X = data.data

# Apply PCA to reduce dimensions to 2
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

# Plot the results
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=data.target, cmap='viridis')
plt.title('PCA Reduction of Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()

# Print PCA information
explained_variance = pca.explained_variance_ratio_
print(f"Explained Variance by Components: {explained_variance}")
print(f"Shape of Reduced Data: {X_reduced.shape}")
```

LINEAR REGRESSION

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.linear_model import LinearRegression

# Generate synthetic data
np.random.seed(0)
X = 2 * np.random.rand(100, 1)  # 100 random points in [0, 2]
y = 4 + 3 * X + np.random.randn(100, 1)  # Linear relation with noise

# Creating the model
model = LinearRegression()
model.fit(X, y)

# Predictions
y_pred = model.predict(X)

# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Actual data')
plt.plot(X, y_pred, color='red', linewidth=2, label='Regression line')
plt.title('Linear Regression (Synthetic Data)')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.show()
```

LOGISTIC REGRESSION

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

# Generating some data points
np.random.seed(0)
X = np.random.uniform(-10, 10, 100)  # Random points between -10 and 10
y = (X > 0).astype(int)  # Label points based on their sign (0 for negative, 1 for positive)

# Reshaping data for Logistic Regression
X = X.reshape(-1, 1)

# Creating and fitting Logistic Regression model
model = LogisticRegression()
model.fit(X, y)

# Predicting probabilities
X_test = np.linspace(-10, 10, 300).reshape(-1, 1)
```

```
y_prob = model.predict_proba(X_test)[:, 1]  # Probability for class 1

# Plotting the curve
plt.plot(X_test, y_prob, color='red', label='Logistic Curve (S-Shape)')

# Plotting data points, with jitter on y-axis to avoid overlap
plt.scatter(X, y + np.random.uniform(-0.02, 0.02, size=y.shape), color='black', label='Data
Points')

# Enhancing plot
plt.xlabel('Feature')
plt.ylabel('Probability of Class 1')
plt.title('Logistic Regression with S-Shaped Curve')
plt.legend()
plt.grid(True)
plt.show()
```