```python
# Decision Tree on Iris Dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target, name='species')
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
dt_model = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=42)
dt_model.fit(X_train, y_train)
y_pred = dt_model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
plt.figure(figsize=(12, 8))
plot_tree(dt_model, feature_names=iris.feature_names, class_names=iris.target_names,
filled=True)
plt.title("Decision Tree Visualization")
plt.show()
```

Where to Use:
Decision Trees are used for classification/regression. Great for interpretable models. Can handle both numerical and categorical data.
Use Cases: Student pass/fail prediction, medical diagnosis, customer churn.

Random Forest on Iris Dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

# Plot feature importance
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]
plt.figure(figsize=(10, 6))
plt.title("Feature Importances")
plt.bar(range(X.shape[1]), importances[indices], color='lightblue', align="center")
plt.xticks(range(X.shape[1]), [X.columns[i] for i in indices], rotation=45)
plt.tight_layout()
plt.show()
```

Where to Use:
Random Forests are ideal for handling **high-dimensional** datasets with noise. They avoid overfitting and generalize well.
Use Cases: Spam detection, loan default prediction, fraud detection.

Naive Bayes on Iris Dataset

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred = nb.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

Where to Use:
Naive Bayes is best for **text classification** and problems where features are assumed to be independent.
Use Cases: Email spam detection, document classification, sentiment analysis.

Linear Regression on Boston Housing Dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

data = fetch_california_housing()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target, name="HouseValue")

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)

print("MSE:", mean_squared_error(y_test, y_pred))
print("R^2 Score:", r2_score(y_test, y_pred))

plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual vs Predicted House Values")
plt.grid(True)
plt.show()
```

Where to Use:
Linear Regression is used when the target is **continuous** and there's a linear relationship between features and output.
Use Cases: House price prediction, salary estimation, stock price forecasting.

K-Means on Iris Dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

iris = load_iris()
X = iris.data

kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
labels = kmeans.labels_

# Visualize using PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap='viridis')
plt.title("K-Means Clustering (PCA-reduced)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

Where to Use:
K-means is for **unsupervised clustering** tasks with well-separated groups.
Use Cases: Customer segmentation, image compression, grouping users by behavior.

GMM Clustering on Iris Dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.mixture import GaussianMixture
from sklearn.decomposition import PCA

iris = load_iris()
X = iris.data

gmm = GaussianMixture(n_components=3, random_state=42)
gmm.fit(X)
labels = gmm.predict(X)

# Visualize using PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap='viridis')
plt.title("GMM Clustering (PCA-reduced)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

Where to Use:
GMM is good for **soft clustering** or when clusters **overlap** and data is probabilistic.
Use Cases: Speaker recognition, anomaly detection, market segmentation.

SVM Classifier on Iris Dataset (Binary Classification)

```
import pandas as pd
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 classification_report, accuracy_score

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

# Convert to binary classification (class 0 vs class 1)
X = X[y != 2]
y = y[y != 2]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Where to Use:
SVM is best for **binary classification** problems with **clear margins** between classes.
Use Cases: Cancer detection, spam classification, facial expression recognition.

SVM Multiclass using One-vs-Rest (OvR)

```
from sklearn.svm import SVC
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import pandas as pd

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

svm_multi = SVC(kernel='rbf', decision_function_shape='ovr')
svm_multi.fit(X_train, y_train)
y_pred = svm_multi.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Where to Use:
Multiclass SVM handles problems with more than 2 labels using **OvR** or **OvO** strategy.
Use Cases: Handwritten digit recognition, multi-class object detection.

SVM on Text Classification using TF-IDF

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

data = fetch_20newsgroups(subset='train', categories=['rec.autos', 'sci.space'],
remove=('headers', 'footers', 'quotes'))
X = data.data
y = data.target

vectorizer = TfidfVectorizer(stop_words='english')
X_tfidf = vectorizer.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42)

svm_text = SVC()
svm_text.fit(X_train, y_train)
y_pred = svm_text.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Where to Use:
SVM with TF-IDF is excellent for **text classification** problems.
Use Cases: News categorization, spam filtering, sentiment analysis.

```
# SVR on synthetic non-linear regression
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

# Generate synthetic data
X = np.sort(5 * np.random.rand(100, 1), axis=0)
y = np.sin(X).ravel() + np.random.normal(0, 0.1, X.shape[0])

svr_rbf = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1)
svr_rbf.fit(X, y)
y_pred = svr_rbf.predict(X)

plt.scatter(X, y, color='darkorange', label='Data')
plt.plot(X, y_pred, color='navy', lw=2, label='SVR model')
plt.xlabel("X")
plt.ylabel("y")
plt.title("SVR on Non-linear Data")
plt.legend()
plt.show()
```

Where to Use:
SVR is used when you want a **non-linear regression model** with **margin tolerance**.
Use Cases: Stock price prediction, energy load forecasting, salary prediction.

 # SLP on Linearly Separable Dataset

```
from sklearn.linear_model import Perceptron
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X, y = make_classification(n_samples=100, n_features=2, n_classes=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=0.2, random_state=42)

slp = Perceptron(max_iter=1000)
slp.fit(X_train, y_train)
y_pred = slp.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
```

Where to Use:

SLPs are only effective when data is **linearly separable**.

Use Cases: Basic binary classification tasks, early neural net experiments.

MLP on XOR problem (Non-linear)

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

# XOR dataset
X = [[0,0],[0,1],[1,0],[1,1]]
y = [0,1,1,0]

mlp = MLPClassifier(hidden_layer_sizes=(4,), activation='relu', max_iter=1000,
random_state=1)
mlp.fit(X, y)
pred = mlp.predict(X)

print("Predictions:", pred)
print("Accuracy:", accuracy_score(y, pred))
```

Where to Use:
MLPs can solve **non-linear** problems using **hidden layers** and backpropagation.
Use Cases: OCR, time series prediction, image classification.