LEVEL 3: ADVANCED TASKS

Task 1: Classification (Iris)¶

Step 1: Load the dataset

In [1]:

```
from sklearn.datasets import load_iris
import pandas as pd

# Load the dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target  # target: 0=setosa, 1=versicolor, 2=virginica
df.head()
```

Out[1]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Step 2: Preprocess the data

In [2]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Features and labels
X = df.drop('target', axis=1)
y = df['target']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 3: Train multiple classification models

In [3]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# Initialize models
models = {
    'Logistic Regression': LogisticRegression(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier()
}

# Train and store predictions
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    print(f"\n📊 {name} Evaluation:")

    from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_s
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Precision:", precision_score(y_test, y_pred, average='weighted'))
    print("Recall:", recall_score(y_test, y_pred, average='weighted'))
    print("F1 Score:", f1_score(y_test, y_pred, average='weighted'))
```

```
📊 Logistic Regression Evaluation:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0

📊 Decision Tree Evaluation:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0

📊 Random Forest Evaluation:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```

Step 4:Evaluate models

In [4]:

```python
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

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

```
y = data.target

# Create pipelines (scaling + model)
models = {
    'Logistic Regression': make_pipeline(StandardScaler(), LogisticRegression(max_i
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(n_estimators=50, max_depth=5, random_st
}

# Evaluate using cross_val_score
for name, model in models.items():
    precision = cross_val_score(model, X, y, cv=5, scoring='precision_macro').mean(
    recall = cross_val_score(model, X, y, cv=5, scoring='recall_macro').mean()
    f1 = cross_val_score(model, X, y, cv=5, scoring='f1_macro').mean()

    print(f"\n📊 {name} Evaluation:")
    print(f"Precision (macro): {precision:.3f}")
    print(f"Recall (macro):    {recall:.3f}")
    print(f"F1 Score (macro):  {f1:.3f}")
```

```
📊 Logistic Regression Evaluation:
Precision (macro): 0.963
Recall (macro):    0.960
F1 Score (macro):  0.960

📊 Decision Tree Evaluation:
Precision (macro): 0.955
Recall (macro):    0.953
F1 Score (macro):  0.953

📊 Random Forest Evaluation:
Precision (macro): 0.971
Recall (macro):    0.967
F1 Score (macro):  0.966
```

Step 4: Hyperparameter tuning with GridSearchCV (Random Forest Example)

In [5]:

```
from sklearn.model_selection import GridSearchCV

# Set parameter grid
param_grid = {
    'n_estimators': [10, 50, 100],
    'max_depth': [None, 3, 5, 10]
}

# Grid Search
grid = GridSearchCV(RandomForestClassifier(), param_grid, cv=5, scoring='accuracy')
grid.fit(X_train_scaled, y_train)

print("\nBest Parameters:", grid.best_params_)
print("Best Accuracy on Training Set:", grid.best_score_)
```

```
Best Parameters: {'max_depth': None, 'n_estimators': 50}
Best Accuracy on Training Set: 0.9428571428571428
```