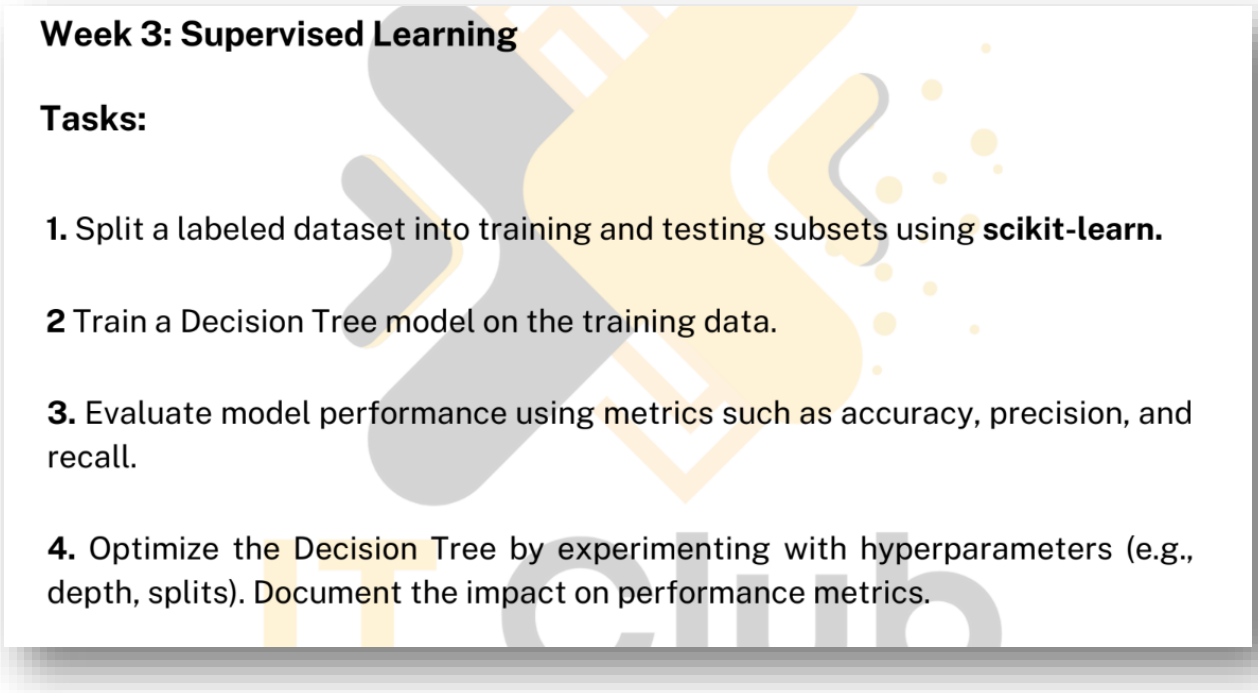
ITCP-25/AI-003 (Noor Fatima)

Week 3: Supervised Learning

Tasks:

I was assigned the task of implementing a **Decision Tree model** using **scikit-learn** for supervised learning. This task involves:

1. Splitting a labeled dataset into **training** and **testing** subsets.
2. Training a **Decision Tree classifier** on the training data.
3. Evaluating the model's performance using metrics such as **accuracy, precision, and recall**.
4. Optimizing the **Decision Tree model** by experimenting with hyperparameters (e.g., depth, splits) and documenting their impact on performance.



Task 1:

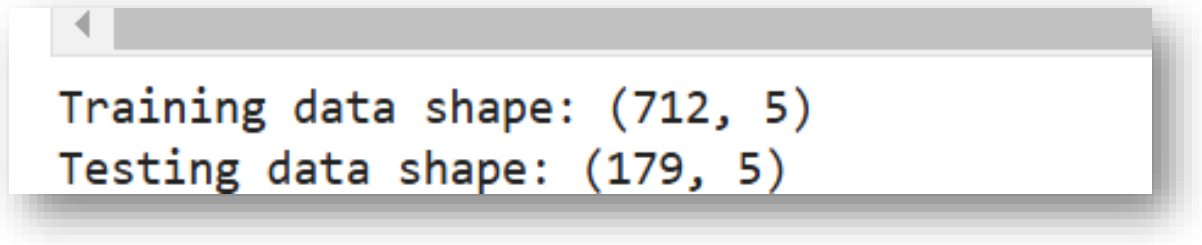
Splitting the Dataset into Training and Testing Subsets

1. **Introduction**: To train the Decision Tree model, the dataset needs to be divided into **training (80%)** and **testing (20%)** subsets.

Code:

```
# Load the Titanic dataset

titanic = sns.load_dataset("titanic")

# Select features and target variable

features = titanic[["pclass", "age", "fare", "sibsp", "parch"]]

target = titanic["survived"]

# Handle missing values by filling with median

features.fillna(features.median(), inplace=True)

# Split data into training (80%) and testing (20%) sets

X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

# Display dataset shapes

print("Training data shape:", X_train.shape)

print("Testing data shape:", X_test.shape)
```

Output:



Task 2:

A **Decision Tree Classifier** was trained using the **training dataset**.

Code:

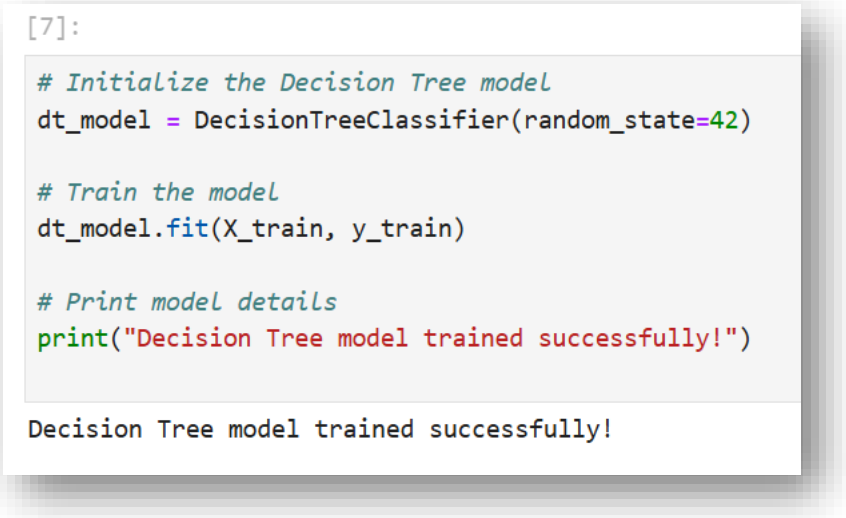
```
# Initialize the Decision Tree model

dt_model = DecisionTreeClassifier(random_state=42)

# Train the model
```

dt_model.fit(X_train, y_train)

Print model details

print("Decision Tree model trained successfully!")

Output:



Task 3:

Evaluating Model Performance

To assess the performance of the model, the following metrics were used:

- **Accuracy**: Measures overall correctness.
- **Precision**: Measures the proportion of correct positive predictions.
- **Recall**: Measures the ability to detect positive instances.

Code:

Make predictions on the test set

y_pred = dt_model.predict(X_test)

Calculate performance metrics

accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred)

recall = recall_score(y_test, y_pred)

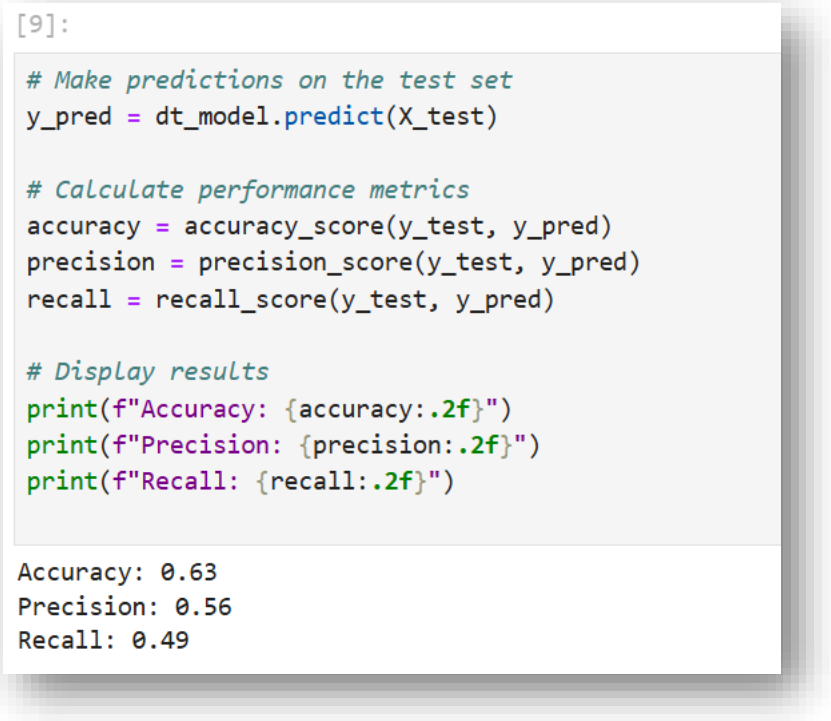
Display results

print(f"Accuracy: {accuracy:.2f}")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

Output:



Task 4:

Optimizing the Decision Tree Model:

To improve performance, hyperparameters such as **max depth** and **min samples split** were adjusted.

Code:

Initialize the optimized Decision Tree model

optimized_dt = DecisionTreeClassifier(max_depth=5, min_samples_split=10, random_state=42)

Train the optimized model

optimized_dt.fit(X_train, y_train)

Make predictions

y_pred_optimized = optimized_dt.predict(X_test)

Calculate new performance metrics

optimized_accuracy = accuracy_score(y_test, y_pred_optimized)

optimized_precision = precision_score(y_test, y_pred_optimized)

optimized_recall = recall_score(y_test, y_pred_optimized)

Display optimized results

print(f"Optimized Accuracy: {optimized_accuracy:.2f}")

print(f"Optimized Precision: {optimized_precision:.2f}")

print(f"Optimized Recall: {optimized_recall:.2f}")

Output:

```
# Initialize the optimized Decision Tree model
optimized_dt = DecisionTreeClassifier(max_depth=5, min_samples_split=10,

# Train the optimized model
optimized_dt.fit(X_train, y_train)

# Make predictions
y_pred_optimized = optimized_dt.predict(X_test)

# Calculate new performance metrics
optimized_accuracy = accuracy_score(y_test, y_pred_optimized)
optimized_precision = precision_score(y_test, y_pred_optimized)
optimized_recall = recall_score(y_test, y_pred_optimized)

# Display optimized results
print(f"Optimized Accuracy: {optimized_accuracy:.2f}")
print(f"Optimized Precision: {optimized_precision:.2f}")
print(f"Optimized Recall: {optimized_recall:.2f}")
```

```
Optimized Accuracy: 0.72
Optimized Precision: 0.85
Optimized Recall: 0.38
```