```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, KFold, StratifiedKFold, GridSea
rchCV, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder, O
neHotEncoder, RobustScaler
from sklearn.impute import SimpleImputer
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
 roc_auc_score, confusion_matrix, classification_report
from sklearn.pipeline import Pipeline
import time
import warnings
warnings.filterwarnings('ignore')  # For cleaner output


# 1. Dataset Loading and Initial Exploration

# Dataset 1: Iris (Classification - built-in)
from sklearn.datasets import load_iris
iris = load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target

# Dataset 2: IMDB Movie Reviews (Text Classification - requires download)
# (Replace with your actual path)
try:
    imdb_df = pd.read_csv("IMDB Dataset.csv")  #  Ensure you have downloaded this
dataset.
except FileNotFoundError:
    print("Error: IMDB dataset not found. Please download and place it in the correct d
irectory.")
    exit() # Exit if dataset isn't found

# 2. Exploratory Data Analysis (EDA) and Visualization

# Iris EDA
print("Iris EDA:")
print(iris_df.describe())
```

```
sns.pairplot(iris_df, hue='target')
plt.show()

# IMDB EDA
print("\nIMDB EDA:")
print(imdb_df.head())
print(imdb_df['sentiment'].value_counts()) # Check class balance
sns.countplot(x='sentiment', data=imdb_df)
plt.show()
imdb_df['review_length'] = imdb_df['review'].apply(len)
sns.histplot(imdb_df['review_length'], bins=50) # Review length distribution
plt.show()

# 3. Data Pre-processing

# Iris Preprocessing (Scaling)
scaler_iris = StandardScaler()
iris_df[iris.feature_names] = scaler_iris.fit_transform(iris_df[iris.feature_names])


# IMDB Preprocessing (Text Processing, Encoding)
imdb_df['sentiment'] = imdb_df['sentiment'].map({'positive': 1, 'negative': 0}) # Encode
 sentiment

from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(max_features=5000) # Example: Use TF-IDF
X_imdb = tfidf.fit_transform(imdb_df['review']).toarray() # Convert to array for compati
bility
y_imdb = imdb_df['sentiment']

# 4. Train-Test Split and Cross-Validation

# Iris
X_iris = iris_df[iris.feature_names]
y_iris = iris_df['target']
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris, test_si
ze=0.2, random_state=42, stratify=y_iris) # Stratified

# IMDB
X_train_imdb, X_test_imdb, y_train_imdb, y_test_imdb = train_test_split(X_imdb, y_i
mdb, test_size=0.2, random_state=42, stratify=y_imdb) # Stratified

# Example K-Fold (for Iris)
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
for train_index, val_index in kf.split(X_iris):
   # ... (Train and evaluate model within each fold)
   # 5. Model Training and Comparison
   # Iris Models

   # The models_iris and models_imdb dictionaries should be indented to be inside t
he for loop
   models_iris = {
      "Decision Tree": DecisionTreeClassifier(),
      "SVM": SVC(),
      "Neural Network": MLPClassifier(max_iter=500, random_state=42) # Adjust ma
x_iter as needed
   }

   # IMDB Models
   models_imdb = {
      "Decision Tree": DecisionTreeClassifier(),
      "Neural Network": MLPClassifier(max_iter=500, random_state=42)
   }

   # Training and Evaluation (Iris)
   for name, model in models_iris.items():
      start_time = time.time()
      model.fit(X_train_iris, y_train_iris)
      y_pred_iris = model.predict(X_test_iris)
      end_time = time.time()

      print(f"Iris - {name}:")
      print(classification_report(y_test_iris, y_pred_iris))
      print(f"Time taken: {end_time - start_time:.2f} seconds")

   # Training and Evaluation (IMDB)
   for name, model in models_imdb.items():
      start_time = time.time()
      model.fit(X_train_imdb, y_train_imdb)
      y_pred_imdb = model.predict(X_test_imdb)
      end_time = time.time()

      print(f"IMDB - {name}:")
      print(classification_report(y_test_imdb, y_pred_imdb))
      print(f"Time taken: {end_time - start_time:.2f} seconds")
```

```
# ... (Rest of the code) ...


# 6. Hyperparameter Tuning (Example with GridSearchCV - Iris Decision Tree)
param_grid = {'max_depth': [None, 5, 10], 'min_samples_split': [2, 5, 10]}
grid_search = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=5, scoring='a
ccuracy')
grid_search.fit(X_train_iris, y_train_iris)
print("\nBest Hyperparameters (Iris Decision Tree):", grid_search.best_params_)
best_dt = grid_search.best_estimator_
y_pred_best_dt = best_dt.predict(X_test_iris)
print(classification_report(y_test_iris, y_pred_best_dt))

# 7. Evaluation Metrics (Already included in classification_report)

# 8. Time and Memory Analysis (Time already measured above)
# Memory profiling might require additional libraries like memory_profiler

# 9. Overfitting/Underfitting Mitigation (Example - Iris Decision Tree)
# (Adjust max_depth, min_samples_split, etc. or use regularization in other models)

# 10. Sensitivity to Noise (Example - Add noise to IMDB test data)
# (Implement noise addition and evaluate the model's performance)

# ... (Add noise and test the model's robustness)
```

RESULT

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
Count	150.000000	150.000000	150.000000	150.000000	150.000000
Mean	5.843333	3.057333	3.758000	1.199333	1.000000
Standard	0.828066	0.435866	1.765298	0.762238	0.819232
Minimum	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000

75%	6.400000	3.300000	5.100000	1.800000	2.000000
Maximum	7.900000	4.400000	6.900000	2.500000	2.000000