```
import pandas as pd
import numpy as np
import itertools
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, StratifiedKFold,
GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import (accuracy_score, precision_score,
recall_score,
                             f1_score, roc_auc_score, roc_curve,
                             confusion_matrix, ConfusionMatrixDisplay,
classification_report)
# Bypass SSL certificate verification for dataset downloads
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split,
GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

# Define classifier parameter grids including SelectKBest feature
selection 'k'
param_grid_dt = {
    'classifier__max_depth': [3, 5, 10, None],
    'classifier__min_samples_split': [2, 5, 10],
    'feature_selection__k': [5, 10, 15, 'all']
```

```python
}

param_grid_knn = {
    'classifier__n_neighbors': [3, 5, 7, 9],
    'classifier__weights': ['uniform', 'distance'],
    'feature_selection__k': [5, 10, 15, 'all']
}

param_grid_lr = {
    'classifier__C': [0.1, 1, 10],
    'classifier__penalty': ['l2'],
    'classifier__solver': ['lbfgs'],
    'feature_selection__k': [5, 10, 15, 'all']
}

classifiers_to_tune = [
    (DecisionTreeClassifier(random_state=42), param_grid_dt, 'Decision
Tree'),
    (KNeighborsClassifier(), param_grid_knn, 'kNN'),
    (LogisticRegression(max_iter=200), param_grid_lr, 'Logistic
Regression')
]

# Load IBM HR Attrition Dataset
def load_hr_attrition():
    df = pd.read_csv('/WA_Fn-UseC_-HR-Employee-Attrition.csv')
    df['Attrition'] = (df['Attrition'] == 'Yes').astype(int)
    X = df.drop(['EmployeeNumber', 'Attrition'], axis=1, errors='ignore')
    X = pd.get_dummies(X, drop_first=True)
    y = df['Attrition']
    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
test_size=0.3, random_state=42)
    print(f"HR Attrition dataset loaded. Train shape: {X_train.shape},
Test shape: {X_test.shape}")
    return X_train, X_test, y_train, y_test, "HR Attrition"

# Run built-in GridSearchCV for classifiers
def run_builtin_grid_search(X_train, y_train, dataset_name):
    print(f"\n{'='*60}")
    print(f"RUNNING BUILT-IN GRID SEARCH FOR {dataset_name.upper()}")
```

```
    print(f"{'='*60}")

    results_builtin = {}
    n_features = X_train.shape[1]

    for classifier_instance, param_grid, name in classifiers_to_tune:
        print(f"\n--- GridSearchCV for {name} ---")

        # Adjust 'all' in feature_selection__k to number of features
        param_grid_adjusted = dict(param_grid)
        if 'feature_selection__k' in param_grid_adjusted:
            param_grid_adjusted['feature_selection__k'] = [k if k != 'all'
else n_features for k in param_grid_adjusted['feature_selection__k']]


        pipeline = Pipeline(steps=[
            ('scaler', StandardScaler()),
            ('feature_selection', SelectKBest(f_classif)),
            ('classifier', classifier_instance)
        ])

        cv_splitter = StratifiedKFold(n_splits=5, shuffle=True,
random_state=42)

        grid_search = GridSearchCV(pipeline, param_grid_adjusted,
cv=cv_splitter, scoring='roc_auc', n_jobs=-1)
        grid_search.fit(X_train, y_train)

        results_builtin[name] = {
            'best_estimator': grid_search.best_estimator_,
            'best_score (CV)': grid_search.best_score_,
            'best_params': grid_search.best_params_
        }

        print(f"Best params for {name}:
{results_builtin[name]['best_params']}")
        print(f"Best CV score: {results_builtin[name]['best_score
(CV)']:.4f}")

    return results_builtin
```

```python
# Example of running the code
X_train, X_test, y_train, y_test, dataset_name = load_hr_attrition()
results = run_builtin_grid_search(X_train, y_train, dataset_name)

# Display results summary
for model_name, result in results.items():
    print(f"\nModel: {model_name}")
    print(f"Best Params: {result['best_params']}")
    print(f"Best CV ROC AUC: {result['best_score (CV)']:.4f}")
```

```
    print(f"Best CV ROC AUC: {result['best_score (CV)']:.4f}")
HR Attrition dataset loaded. Train shape: (1029, 46), Test shape: (441, 46)

============================================================
RUNNING BUILT-IN GRID SEARCH FOR HR ATTRITION
============================================================

--- GridSearchCV for Decision Tree ---
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_selection.py:111: UserWarning: Features [ 4 16] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_selection.py:112: RuntimeWarning: invalid value encountered in divide
  f = msb / msw
Best params for Decision Tree: {'classifier__max_depth': 3, 'classifier__min_samples_split': 2, 'feature_selection__k': 5}
Best CV score: 0.7152

--- GridSearchCV for kNN ---
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_selection.py:111: UserWarning: Features [ 4 16] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_selection.py:112: RuntimeWarning: invalid value encountered in divide
  f = msb / msw
Best params for kNN: {'classifier__n_neighbors': 9, 'classifier__weights': 'distance', 'feature_selection__k': 10}
Best CV score: 0.7226

--- GridSearchCV for Logistic Regression ---
Best params for Logistic Regression: {'classifier__C': 0.1, 'classifier__penalty': 'l2', 'classifier__solver': 'lbfgs', 'feature_selection__k': 46}
Best CV score: 0.8329

Model: Decision Tree
Best Params: {'classifier__max_depth': 3, 'classifier__min_samples_split': 2, 'feature_selection__k': 5}
Best CV ROC AUC: 0.7152

Model: kNN
Best Params: {'classifier__n_neighbors': 9, 'classifier__weights': 'distance', 'feature_selection__k': 10}
```


```
RUNNING BUILT-IN GRID SEARCH FOR HR ATTRITION
============================================================

--- GridSearchCV for Decision Tree ---
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_selection.py:111: UserWarning: Features [ 4 16] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_selection.py:112: RuntimeWarning: invalid value encountered in divide
  f = msb / msw
Best params for Decision Tree: {'classifier__max_depth': 3, 'classifier__min_samples_split': 2, 'feature_selection__k': 5}
Best CV score: 0.7152

--- GridSearchCV for kNN ---
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_selection.py:111: UserWarning: Features [ 4 16] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_selection.py:112: RuntimeWarning: invalid value encountered in divide
  f = msb / msw
Best params for kNN: {'classifier__n_neighbors': 9, 'classifier__weights': 'distance', 'feature_selection__k': 10}
Best CV score: 0.7226

--- GridSearchCV for Logistic Regression ---
Best params for Logistic Regression: {'classifier__C': 0.1, 'classifier__penalty': 'l2', 'classifier__solver': 'lbfgs', 'feature_selection__k': 46}
Best CV score: 0.8329

Model: Decision Tree
Best Params: {'classifier__max_depth': 3, 'classifier__min_samples_split': 2, 'feature_selection__k': 5}
Best CV ROC AUC: 0.7152

Model: kNN
Best Params: {'classifier__n_neighbors': 9, 'classifier__weights': 'distance', 'feature_selection__k': 10}
Best CV ROC AUC: 0.7226
```


```
Model: Logistic Regression
Best Params: {'classifier__C': 0.1, 'classifier__penalty': 'l2', 'classifier__solver': 'lbfgs', 'feature_selection__k': 46}
```

bles Terminal

```
HR Attrition dataset loaded. Train shape: (1029, 46), Test shape: (441, 46)

============================================================
RUNNING BUILT-IN GRID SEARCH FOR HR ATTRITION
============================================================

--- GridSearchCV for Decision Tree ---
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_selection.py:111: UserWarning: Features [ 4 16] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_selection.py:112: RuntimeWarning: invalid value encountered in divide
  f = msb / msw
Best params for Decision Tree: {'classifier__max_depth': 3, 'classifier__min_samples_split': 2, 'feature_selection__k': 5}
Best CV score: 0.7152

--- GridSearchCV for kNN ---
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_selection.py:111: UserWarning: Features [ 4 16] are constant.
```

```python
# The parameter names must match the pipeline step names, e.g.,
'classifier__max_depth'
# Define base models (Decision Tree, kNN, Logistic Regression)

param_grid_dt = {
    'classifier__max_depth': [3, 5, 10, None],
    'classifier__min_samples_split': [2, 5, 10],
    'feature_selection__k': [5, 10, 15, 'all']
}

param_grid_knn = {
    'classifier__n_neighbors': [3, 5, 7, 9],
    'classifier__weights': ['uniform', 'distance'],
    'feature_selection__k': [5, 10, 15, 'all']
}

param_grid_lr = {
    'classifier__C': [0.1, 1, 10],
    'classifier__penalty': ['l2'],
    'classifier__solver': ['lbfgs'],
    'feature_selection__k': [5, 10, 15, 'all']
}

# Create a list of (classifier, param_grid, name) tuples
classifiers_to_tune = [
    (DecisionTreeClassifier(random_state=42), param_grid_dt, 'Decision
Tree'),
    (KNeighborsClassifier(), param_grid_knn, 'kNN'),
```

```
    (LogisticRegression(max_iter=200), param_grid_lr, 'Logistic
Regression')
]
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, train_test_split,
GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import (accuracy_score, precision_score,
recall_score,
                             f1_score, roc_auc_score, roc_curve,
                             confusion_matrix, ConfusionMatrixDisplay,
classification_report)
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

# Define classifier parameter grids including SelectKBest feature
selection 'k'
param_grid_dt = {
    'classifier__max_depth': [3, 5, 10, None],
    'classifier__min_samples_split': [2, 5, 10],
    'feature_selection__k': [5, 10, 15, 'all']
}

param_grid_knn = {
    'classifier__n_neighbors': [3, 5, 7, 9],
    'classifier__weights': ['uniform', 'distance'],
    'feature_selection__k': [5, 10, 15, 'all']
}

param_grid_lr = {
```

```
    'classifier__C': [0.1, 1, 10],
    'classifier__penalty': ['l2'],
    'classifier__solver': ['lbfgs'],
    'feature_selection__k': [5, 10, 15, 'all']
}

classifiers_to_tune = [
    (DecisionTreeClassifier(random_state=42), param_grid_dt, 'Decision 
Tree'),
    (KNeighborsClassifier(), param_grid_knn, 'kNN'),
    (LogisticRegression(max_iter=200), param_grid_lr, 'Logistic 
Regression')
]

# Load IBM HR Attrition Dataset
def load_hr_attrition():
    df = pd.read_csv('/WA_Fn-UseC_-HR-Employee-Attrition.csv')
    df['Attrition'] = (df['Attrition'] == 'Yes').astype(int)
    X = df.drop(['EmployeeNumber', 'Attrition'], axis=1, errors='ignore')
    X = pd.get_dummies(X, drop_first=True)
    y = df['Attrition']
    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, 
test_size=0.3, random_state=42)
    print(f"HR Attrition dataset loaded. Train shape: {X_train.shape}, 
Test shape: {X_test.shape}")
    return X_train, X_test, y_train, y_test, "HR Attrition"

# Load Wine Quality dataset
def load_wine_quality():
    """Load Wine Quality dataset"""
    url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-
quality/winequality-red.csv'
    try:
        data = pd.read_csv(url, sep=';')
    except Exception as e:
        print(f"Error loading Wine Quality dataset: {e}")
        return None, None, None, None, "Wine Quality (Failed)"

    # Create the binary target variable 'good_quality'
    data['good_quality'] = (data['quality'] > 5).astype(int)
```

```
    X = data.drop(['quality', 'good_quality'], axis=1)
    y = data['good_quality']

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=42, stratify=y
    )

    print("Wine Quality dataset loaded and preprocessed successfully.")
    print(f"Training set shape: {X_train.shape}")
    print(f"Testing set shape: {X_test.shape}")
    return X_train, X_test, y_train, y_test, "Wine Quality"


# Run built-in GridSearchCV for classifiers
def run_builtin_grid_search(X_train, y_train, dataset_name):
    print(f"\n{'='*60}")
    print(f"RUNNING BUILT-IN GRID SEARCH FOR {dataset_name.upper()}")
    print(f"{'='*60}")

    results_builtin = {}
    n_features = X_train.shape[1]

    for classifier_instance, param_grid, name in classifiers_to_tune:
        print(f"\n--- GridSearchCV for {name} ---")

        # Adjust 'all' in feature_selection__k to number of features
        param_grid_adjusted = dict(param_grid)
        if 'feature_selection__k' in param_grid_adjusted:
            param_grid_adjusted['feature_selection__k'] = [k if k != 'all'
else n_features for k in param_grid_adjusted['feature_selection__k']]


        pipeline = Pipeline(steps=[
            ('scaler', StandardScaler()),
            ('feature_selection', SelectKBest(f_classif)),
            ('classifier', classifier_instance)
        ])
```

```
        cv_splitter = StratifiedKFold(n_splits=5, shuffle=True,
random_state=42)

        grid_search = GridSearchCV(pipeline, param_grid_adjusted,
cv=cv_splitter, scoring='roc_auc', n_jobs=-1)
        grid_search.fit(X_train, y_train)

        results_builtin[name] = {
            'best_estimator': grid_search.best_estimator_,
            'best_score (CV)': grid_search.best_score_,
            'best_params': grid_search.best_params_
        }

        print(f"Best params for {name}:
{results_builtin[name]['best_params']}")
        print(f"Best CV score: {results_builtin[name]['best_score
(CV)']:.4f}")

    return results_builtin

# Example of running the code (using Wine Quality dataset)
X_train, X_test, y_train, y_test, dataset_name = load_wine_quality()

if X_train is not None: # Check if dataset loaded successfully
    results = run_builtin_grid_search(X_train, y_train, dataset_name)

    # Display results summary
    for model_name, result in results.items():
        print(f"\nModel: {model_name}")
        print(f"Best Params: {result['best_params']}")
        print(f"Best CV ROC AUC: {result['best_score (CV)']:.4f}")
```

```
        print(f"Best CV ROC AUC: {result['best_score (CV)']:.4f}")
```

```
Wine Quality dataset loaded and preprocessed successfully.
Training set shape: (1119, 11)
Testing set shape: (480, 11)

============================================================
RUNNING BUILT-IN GRID SEARCH FOR WINE QUALITY
============================================================

--- GridSearchCV for Decision Tree ---
Best params for Decision Tree: {'classifier__max_depth': 5, 'classifier__min_samples_split': 5, 'feature_selection__k': 5}
Best CV score: 0.7832

--- GridSearchCV for kNN ---
Best params for kNN: {'classifier__n_neighbors': 9, 'classifier__weights': 'distance', 'feature_selection__k': 5}
Best CV score: 0.8642

--- GridSearchCV for Logistic Regression ---
Best params for Logistic Regression: {'classifier__C': 10, 'classifier__penalty': 'l2', 'classifier__solver': 'lbfgs', 'feature_selection__k': 15}
Best CV score: 0.8051

Model: Decision Tree
Best Params: {'classifier__max_depth': 5, 'classifier__min_samples_split': 5, 'feature_selection__k': 5}
Best CV ROC AUC: 0.7832

Model: kNN
Best Params: {'classifier__n_neighbors': 9, 'classifier__weights': 'distance', 'feature_selection__k': 5}
Best CV ROC AUC: 0.8642

Model: Logistic Regression
Best Params: {'classifier__C': 10, 'classifier__penalty': 'l2', 'classifier__solver': 'lbfgs', 'feature_selection__k': 15}
Best CV ROC AUC: 0.8051
/usr/local/lib/python3.12/dist-packages/sklearn/feature_selection/_univariate_selection.py:783: UserWarning: k=15 is greater than n_features=11. All the features will be returned.
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split

def load_wine_quality():
    """Load Wine Quality dataset"""
    url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv'
    try:
        data = pd.read_csv(url, sep=';')
    except Exception as e:
        print(f"Error loading Wine Quality dataset: {e}")
        return None, None, None, None, "Wine Quality (Failed)"

    # Create the binary target variable 'good_quality'
    data['good_quality'] = (data['quality'] > 5).astype(int)
    X = data.drop(['quality', 'good_quality'], axis=1)
    y = data['good_quality']

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=42, stratify=y
    )

    print("Wine Quality dataset loaded and preprocessed successfully.")
    print(f"Training set shape: {X_train.shape}")
```

```
    print(f"Testing set shape: {X_test.shape}")
    return X_train, X_test, y_train, y_test, "Wine Quality"
X_train, X_test, y_train, y_test, dataset_name = load_wine_quality()

# Display the shapes to confirm data is loaded
if X_train is not None:
    print("\nData loaded successfully:")
    print(f"X_train shape: {X_train.shape}")
    print(f"X_test shape: {X_test.shape}")
    print(f"y_train shape: {y_train.shape}")
    print(f"y_test shape: {y_test.shape}")
```

```
    print(f"y_test shape: {y_test.shape}")

Wine Quality dataset loaded and preprocessed successfully.
Training set shape: (1119, 11)
Testing set shape: (480, 11)

Data loaded successfully:
X_train shape: (1119, 11)
X_test shape: (480, 11)
y_train shape: (1119,)
y_test shape: (480,)

[ ] import pandas as pd
    import numpy as np
    from sklearn.model_selection import train_test_split
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

def load_hr_attrition():
    """Load IBM HR Attrition dataset"""
    try:
        data = pd.read_csv("data/WA_Fn-UseC_-HR-Employee-Attrition.csv")
    except FileNotFoundError:
        print("HR Attrition dataset not found. Please place 'WA_Fn-UseC_-HR-Employee-Attrition.csv' inside a 'data/' folder.")
        return None, None, None, None, "HR Attrition (Failed)"
```

```
    # Target: Attrition = Yes (1), No (0)
    data['Attrition'] = (data['Attrition'] == 'Yes').astype(int)

    # Drop ID-like column
    X = data.drop(['EmployeeNumber', 'Attrition'], axis=1,
errors='ignore')
    y = data['Attrition']

    # One-hot encode categorical variables
    X = pd.get_dummies(X, drop_first=True)

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, stratify=y, test_size=0.3, random_state=42
    )

    print("IBM HR Attrition dataset loaded and preprocessed
successfully.")
    print(f"Training set shape: {X_train.shape}")
    print(f"Testing set shape: {X_test.shape}")
    return X_train, X_test, y_train, y_test, "HR Attrition"
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
import os # Import os module

def load_qsar_biodegradation():
    """Load QSAR Biodegradation dataset"""
    url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/00254/biodeg.csv"
    local_path = "/tmp/biodeg.csv" # Define a local path

    # Check if file already exists to avoid re-downloading
    if not os.path.exists(local_path):
        print(f"Downloading QSAR Biodegradation dataset from {url}...")
        # Use wget to download the file
        try:
            !wget -O {local_path} {url}
```

```
            print("Download complete.")
        except Exception as e:
            print(f"Error downloading QSAR dataset: {e}")
            return None, None, None, None, "QSAR (Failed)"
    else:
        print(f"QSAR Biodegradation dataset already exists at
{local_path}.")


    try:
        # Load data from the local file
        data = pd.read_csv(local_path, sep=';', header=None)
    except Exception as e:
        print(f"Error loading QSAR dataset from local file: {e}")
        return None, None, None, None, "QSAR (Failed)"

    # Last column is target (RB = ready biodegradable, NRB = not)
    X = data.iloc[:, :-1]
    y = (data.iloc[:, -1] == 'RB').astype(int)

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, stratify=y, test_size=0.3, random_state=42
    )

    print("QSAR Biodegradation dataset loaded and preprocessed
successfully.")
    print(f"Training set shape: {X_train.shape}")
    print(f"Testing set shape: {X_test.shape}")
    return X_train, X_test, y_train, y_test, "QSAR Biodegradation"
```

```
X_train, X_test, y_train, y_test, dataset_name =
load_qsar_biodegradation()

# Display the shapes to confirm data is loaded
if X_train is not None:
    print("\nData loaded successfully:")
    print(f"X_train shape: {X_train.shape}")
```

```
    print(f"X_test shape: {X_test.shape}")
    print(f"y_train shape: {y_train.shape}")
    print(f"y_test shape: {y_test.shape}")
```

```
    print(f"X_train shape: {X_train.shape}")
    print(f"X_test shape: {X_test.shape}")
    print(f"y_train shape: {y_train.shape}")
    print(f"y_test shape: {y_test.shape}")
```

```
QSAR Biodegradation dataset loaded successfully.
Training set shape: (738, 41)
Testing set shape: (317, 41)

Data loaded successfully:
X_train shape: (738, 41)
X_test shape: (317, 41)
y_train shape: (738,)
y_test shape: (317,)
```

```
[ ] import pandas as pd
    import numpy as np
    import itertools
    from sklearn.model_selection import StratifiedKFold, train_test_split, GridSearchCV
    from sklearn.preprocessing import StandardScaler
    from sklearn.feature_selection import SelectKBest, f_classif
    from sklearn.pipeline import Pipeline
    from sklearn.tree import DecisionTreeClassifier
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import VotingClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import (accuracy_score, precision_score,
recall_score,
                             f1_score, roc_auc_score, roc_curve,
                             confusion_matrix, ConfusionMatrixDisplay)
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# Your original evaluate_models function (slightly simplified for clarity)
```

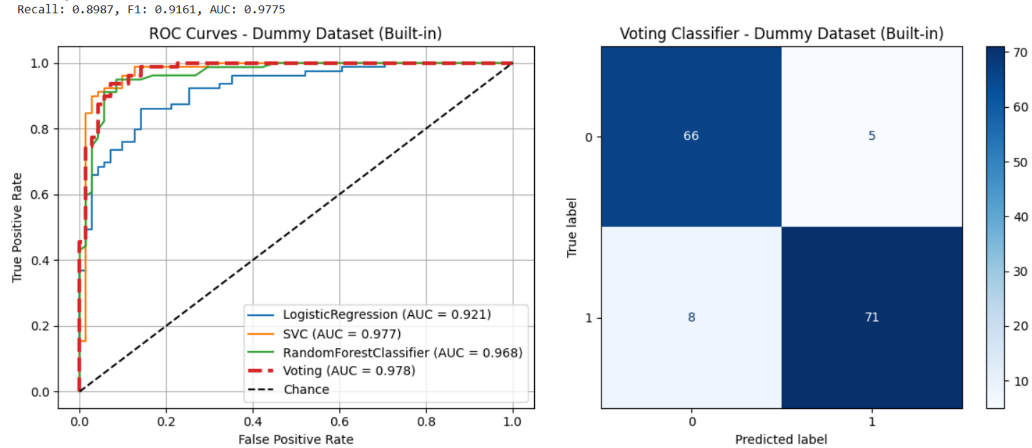
```
def evaluate_models(X_test, y_test, best_estimators, dataset_name,
method_name="Manual"):
    """Evaluate models and create visualizations"""
    print(f"\n{'='*60}")
    print(f"EVALUATING {method_name.upper()} MODELS FOR
{dataset_name.upper()}")
    print(f"{'='*60}")

    # Individual model evaluation
    print(f"\n--- Individual Model Performance ---")
    for name, model in best_estimators.items():
        y_pred = model.predict(X_test)
        y_pred_proba = model.predict_proba(X_test)[:, 1]
        print(f"\n{name}:")
        print(f"  Accuracy: {accuracy_score(y_test, y_pred):.4f}")
        print(f"  Precision: {precision_score(y_test, y_pred,
zero_division=0):.4f}")
        print(f"  Recall: {recall_score(y_test, y_pred,
zero_division=0):.4f}")
        print(f"  F1-Score: {f1_score(y_test, y_pred,
zero_division=0):.4f}")
        print(f"  ROC AUC: {roc_auc_score(y_test, y_pred_proba):.4f}")

    # Voting Classifier
    print(f"\n--- {method_name} Voting Classifier ---")

    # Collect predictions and probabilities from all estimators
    predictions = []
    probabilities = []
    for name, model in best_estimators.items():
        predictions.append(model.predict(X_test))
        probabilities.append(model.predict_proba(X_test)[:, 1])

    predictions_array = np.array(predictions)
    probabilities_array = np.array(probabilities)

    # --- Soft Voting (averaging probabilities) ---
    avg_proba = np.mean(probabilities_array, axis=0)
    y_pred_soft_voting = (avg_proba > 0.5).astype(int)
```

```
    # Compute voting metrics
    accuracy = accuracy_score(y_test, y_pred_soft_voting)
    precision = precision_score(y_test, y_pred_soft_voting,
zero_division=0)
    recall = recall_score(y_test, y_pred_soft_voting, zero_division=0)
    f1 = f1_score(y_test, y_pred_soft_voting, zero_division=0)
    auc = roc_auc_score(y_test, avg_proba)

    print(f"Voting Classifier Performance:")
    print(f"  Accuracy: {accuracy:.4f}, Precision: {precision:.4f}")
    print(f"  Recall: {recall:.4f}, F1: {f1:.4f}, AUC: {auc:.4f}")

    # Visualizations
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)

    # Plot individual model ROC curves
    for name, model in best_estimators.items():
        y_pred_proba = model.predict_proba(X_test)[:, 1]
        fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
        auc_score = roc_auc_score(y_test, y_pred_proba)
        plt.plot(fpr, tpr, label=f'{name} (AUC = {auc_score:.3f})')

    # Add voting classifier to ROC
    fpr_vote, tpr_vote, _ = roc_curve(y_test, avg_proba)
    plt.plot(fpr_vote, tpr_vote, label=f'Voting (AUC = {auc:.3f})',
linewidth=3, linestyle='--')
    plt.plot([0, 1], [0, 1], 'k--', label='Chance')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curves - {dataset_name} ({method_name})')
    plt.legend()
    plt.grid(True)

    # Confusion Matrix for Voting Classifier
    plt.subplot(1, 2, 2)
    cm = confusion_matrix(y_test, y_pred_soft_voting)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=np.unique(y_test))
    disp.plot(ax=plt.gca(), cmap="Blues")
```

```
    plt.title(f'Voting Classifier - {dataset_name} ({method_name})')
    plt.tight_layout()
    plt.show()

    return y_pred_soft_voting, avg_proba

# Create dummy data
X, y = make_classification(n_samples=500, n_features=20, n_informative=10, n_redundant=5, random_state=42)
X = pd.DataFrame(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create and fit dummy models (as if they came from a grid search)
# These are pipelines, as expected by the function
lr_pipe = Pipeline([('scaler', StandardScaler()), ('classifier', LogisticRegression())])
svm_pipe = Pipeline([('scaler', StandardScaler()), ('classifier', SVC(probability=True))])
rf_pipe = Pipeline([('scaler', StandardScaler()), ('classifier', RandomForestClassifier())])

lr_pipe.fit(X_train, y_train)
svm_pipe.fit(X_train, y_train)
rf_pipe.fit(X_train, y_train)

# Dictionary of fitted estimators
best_estimators = {
    'LogisticRegression': lr_pipe,
    'SVC': svm_pipe,
    'RandomForestClassifier': rf_pipe
}

# Run the function with dummy data
evaluate_models(X_test, y_test, best_estimators, "Dummy Dataset", "Built-in")
```

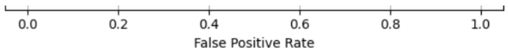
```
============================================================
EVALUATING BUILT-IN MODELS FOR DUMMY DATASET
============================================================

--- Individual Model Performance ---

LogisticRegression:
  Accuracy: 0.8533
  Precision: 0.8701
  Recall: 0.8481
  F1-Score: 0.8590
  ROC AUC: 0.9212

SVC:
  Accuracy: 0.9133
  Precision: 0.9125
  Recall: 0.9241
  F1-Score: 0.9182
  ROC AUC: 0.9768

RandomForestClassifier:
  Accuracy: 0.9200
  Precision: 0.9467
  Recall: 0.8987
  F1-Score: 0.9221
  ROC AUC: 0.9676

--- Built-in Voting Classifier ---
Voting Classifier Performance:
  Accuracy: 0.9133, Precision: 0.9342
  Recall: 0.8987, F1: 0.9161, AUC: 0.9775
```

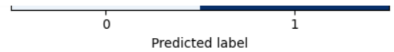
```
Recall: 0.8987, F1: 0.9161, AUC: 0.9775
```




```
(array([1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0,
```

es Terminal




```
(array([1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0,
        0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1,
        1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
        0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1,
        0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1,
        0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0]),
 array([0.91413111, 0.4303724 , 0.19773283, 0.98966219, 0.27349463,
        0.03623587, 0.90930545, 0.09716991, 0.89292068, 0.14449785,
        0.13986384, 0.92471648, 0.03937819, 0.55430608, 0.75388127,
        0.50795928, 0.56708797, 0.07411628, 0.71085153, 0.92628003,
        0.91592984, 0.11172413, 0.11626705, 0.97074615, 0.0641497 ,
        0.34015276, 0.37214307, 0.03334749, 0.49654109, 0.89476801,
        0.05689676, 0.06918987, 0.09840457, 0.4679079 , 0.74228926,
        0.86540719, 0.94924122, 0.83448531, 0.03530719, 0.89612195,
        0.77401921, 0.10452077, 0.94113878, 0.04888414, 0.67605702,
        0.95118747, 0.89860261, 0.88872596, 0.95703869, 0.09143349,
        0.92496344, 0.68218142, 0.84517275, 0.42469126, 0.05875327,
        0.79554807, 0.93892418, 0.94201457, 0.88802953, 0.27230458,
        0.03313419, 0.06896638, 0.20139073, 0.11456599, 0.86171412,
        0.5238338 , 0.96009826, 0.45677059, 0.07713625, 0.20262731,
        0.959624  , 0.97248948, 0.06335743, 0.41334499, 0.93932384,
        0.90114551, 0.06903243, 0.38671038, 0.23619312, 0.7365317 ,
        0.09293074, 0.93223785, 0.35202564, 0.04399325, 0.10698455,
        0.64357045, 0.0790545 , 0.19612217, 0.10225029, 0.98037279,
        0.15575129, 0.08858233, 0.90252106, 0.91280379, 0.97400105,
        0.65821507, 0.76832865, 0.99572026, 0.71196169, 0.69512551,
        0.04621708, 0.88749502, 0.07772958, 0.61941575, 0.0823108 ,
        0.0786285 , 0.26225902, 0.24882174, 0.92014631, 0.95120355,
        0.12771622, 0.37674071, 0.91960244, 0.01772844, 0.7589988 ,
```

```python
def run_complete_pipeline(dataset_loader, dataset_name):
    """Run complete pipeline for a dataset"""
    print(f"\n{'#'*80}")
    print(f"PROCESSING DATASET: {dataset_name.upper()}")
    print(f"{'#'*80}")

    # Load dataset
    X_train, X_test, y_train, y_test, actual_name = dataset_loader()
    if X_train is None:
        print(f"Skipping {dataset_name} due to loading error.")
        return

    print("-" * 30)

    # Part 1: Manual Implementation
    manual_estimators = run_manual_grid_search(X_train, y_train,
actual_name)
    manual_votes, manual_proba = evaluate_models(X_test, y_test,
manual_estimators, actual_name, "Manual")

    print("-" * 30)

    # Part 2: Built-in Implementation
```

```
    builtin_results = run_builtin_grid_search(X_train, y_train,
actual_name)
    builtin_estimators = {name: results['best_estimator']
                          for name, results in builtin_results.items()}
    builtin_votes, builtin_proba = evaluate_models(X_test, y_test,
builtin_estimators, actual_name, "Built-in")

    print(f"\nCompleted processing for {actual_name}")
    print("="*80)
```

```
# --- Run Pipeline for All Datasets ---
datasets = [
    (load_wine_quality, "Wine Quality"),
    (load_hr_attrition, "HR Attrition"),
    (load_banknote, "Banknote Authentication"),
    (load_qsar_biodegradation, "QSAR Biodegradation")
]

# Run for each dataset
for dataset_loader, dataset_name in datasets:
    try:
        run_complete_pipeline(dataset_loader, dataset_name)
    except Exception as e:
        print(f"Error processing {dataset_name}: {e}")
        continue

print("\n" + "="*80)
print("ALL DATASETS PROCESSED!")
print("="*80)
```

```
print("\n" + "="*80)
print("ALL DATASETS PROCESSED!")
print("="*80)
```

```
Skipping Wine Quality due to loading error.

################################################################################
PROCESSING DATASET: HR ATTRITION
################################################################################
Loading HR Attrition data...
Skipping HR Attrition due to loading error.

################################################################################
PROCESSING DATASET: BANKNOTE AUTHENTICATION
################################################################################
Loading Banknote Authentication data...
Skipping Banknote Authentication due to loading error.

################################################################################
PROCESSING DATASET: QSAR BIODEGRADATION
################################################################################
QSAR Biodegradation dataset already exists at /tmp/biodeg.csv.
QSAR Biodegradation dataset loaded and preprocessed successfully.
Training set shape: (738, 41)
Testing set shape: (317, 41)
------------------------------

============================================================
RUNNING MANUAL GRID SEARCH FOR QSAR BIODEGRADATION
============================================================
--- Manual Grid Search for Decision Tree ---
--------------------------------------------------------------------------------
Best parameters for Decision Tree: {'classifier__max_depth': 3, 'classifier__min_samples_split': 2, 'feature_selection__k': 4
Best cross-validation AUC: 0.8369
```


```
print("ALL DATASETS PROCESSED!")
print("="*80)
```


```
Skipping Banknote Authentication due to loading error.
```


```
################################################################################
PROCESSING DATASET: QSAR BIODEGRADATION
################################################################################
QSAR Biodegradation dataset already exists at /tmp/biodeg.csv.
QSAR Biodegradation dataset loaded and preprocessed successfully.
Training set shape: (738, 41)
Testing set shape: (317, 41)
------------------------------

============================================================
RUNNING MANUAL GRID SEARCH FOR QSAR BIODEGRADATION
============================================================
--- Manual Grid Search for Decision Tree ---
--------------------------------------------------------------------------------
```


```
Best parameters for Decision Tree: {'classifier__max_depth': 3, 'classifier__min_samples_split': 2, 'feature_selection__k': 41}
Best cross-validation AUC: 0.8369
--- Manual Grid Search for kNN ---
--------------------------------------------------------------------------------
Best parameters for kNN: {'classifier__n_neighbors': 5, 'classifier__weights': 'distance', 'feature_selection__k': 41}
Best cross-validation AUC: 0.9003
--- Manual Grid Search for Logistic Regression ---
--------------------------------------------------------------------------------
Best parameters for Logistic Regression: {'classifier__C': 0.1, 'classifier__penalty': 'l2', 'classifier__solver': 'lbfgs', 'feature_selection__k': 41}
Best cross-validation AUC: 0.9315

============================================================
EVALUATING MANUAL MODELS FOR QSAR BIODEGRADATION
============================================================
```

Variables Terminal

```python
print("\n" + "="*80)
print("ALL DATASETS PROCESSED!")
print("="*80)
```

```
------------------------------------------------------------------------------------------
Best parameters for Logistic Regression: {'classifier__C': 0.1, 'classifier__penalty': 'l2', 'classifier__solver': 'lbfgs', 'feature_selection__k': 41}
Best cross-validation AUC: 0.9315

============================================================
EVALUATING MANUAL MODELS FOR QSAR BIODEGRADATION
============================================================

--- Individual Model Performance ---

Decision Tree:
  Accuracy: 0.7634
  Precision: 0.6600
  Recall: 0.6168
  F1-Score: 0.6377
  ROC AUC: 0.8007

kNN:
  Accuracy: 0.8644
  Precision: 0.8077
  Recall: 0.7850
  F1-Score: 0.7962
  ROC AUC: 0.8931

Logistic Regression:
  Accuracy: 0.8423
  Precision: 0.8065
  Recall: 0.7009
  F1-Score: 0.7500
  ROC AUC: 0.9069
```

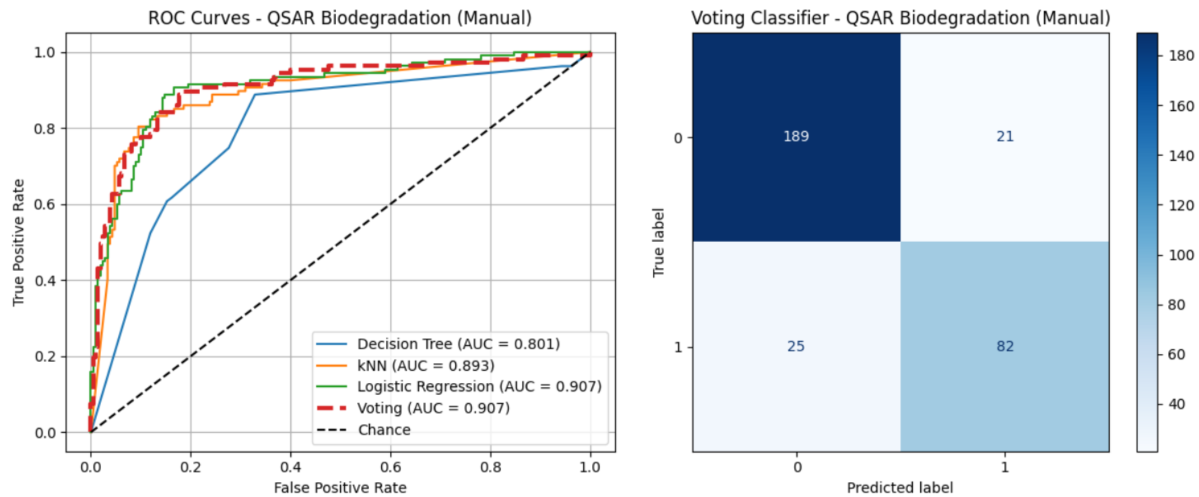
Variables Terminal


```python
print("\n" + "="*80)
print("ALL DATASETS PROCESSED!")
print("="*80)
```

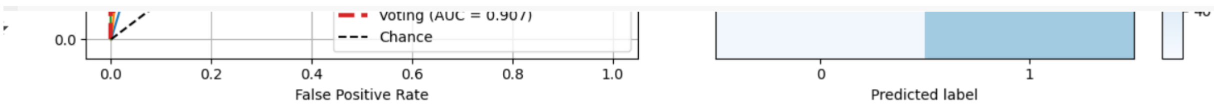
```
  Accuracy: 0.7634
  Precision: 0.6600
  Recall: 0.6168
  F1-Score: 0.6377
  ROC AUC: 0.8007

kNN:
  Accuracy: 0.8644
  Precision: 0.8077
  Recall: 0.7850
  F1-Score: 0.7962
  ROC AUC: 0.8931

Logistic Regression:
  Accuracy: 0.8423
  Precision: 0.8065
  Recall: 0.7009
  F1-Score: 0.7500
  ROC AUC: 0.9069
```


```
--- Manual Voting Classifier ---
Voting Classifier Performance:
  Accuracy: 0.8549, Precision: 0.7961
  Recall: 0.7664, F1: 0.7810, AUC: 0.9074
```

```python
print("\n" + "="*80)
print("ALL DATASETS PROCESSED!")
print("="*80)
```

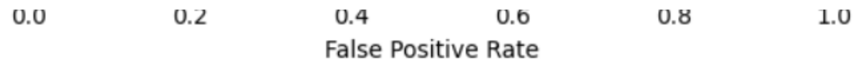



```
-----------------------------
```




```
-----------------------------

--- Running Built-in Grid Search for QSAR Biodegradation ---
Built-in search for LogisticRegression...
Fitting 5 folds for each of 3 candidates, totalling 15 fits
Best params for LogisticRegression: {'classifier__C': 10, 'classifier__solver': 'liblinear'}
Best ROC AUC: 0.9369
Built-in search for SVC...
Fitting 5 folds for each of 3 candidates, totalling 15 fits
Best params for SVC: {'classifier__C': 10, 'classifier__kernel': 'rbf'}
Best ROC AUC: 0.9338
Built-in search for RandomForestClassifier...
Fitting 5 folds for each of 9 candidates, totalling 45 fits
Best params for RandomForestClassifier: {'classifier__max_depth': None, 'classifier__n_estimators': 50}
Best ROC AUC: 0.9282

============================================================
EVALUATING BUILT-IN MODELS FOR QSAR BIODEGRADATION
============================================================

--- Individual Model Performance ---

LogisticRegression:
  Accuracy: 0.8675
```

bles    Terminal




```
Completed processing for QSAR Biodegradation
================================================================================

================================================================================
ALL DATASETS PROCESSED!
================================================================================
```

## Concepts

- **Hyperparameter Tuning: The process of searching for the best combination of parameters (hyperparameters) that optimize model performance.**
- **Grid Search: A systematic way to explore hyperparameter combinations by evaluating all specified settings.**
- **K-Fold Cross-Validation: Splitting data into k subsets (folds) to repeatedly train and validate the model, yielding robust performance estimates.**

1. **StandardScaler: Standardizes features to have zero mean and unit variance.**
2. **SelectKBest: Selects the top `k` features based on statistical tests (ANOVA F-value `f_classif`), where `k` is a hyperparameter to be tuned.**
3. **Classifier: The final step, which can be a Decision Tree, k-Nearest Neighbors (kNN), or Logistic Regression model.**
4. **Logistic Regression model.**

## Manual Grid Search Implementation

- **Define hyperparameter grids for each classifier (e.g., max_depth for Decision Tree, number of neighbors for kNN, regularization strength for Logistic Regression).**
- **Implement nested loops to generate all hyperparameter combinations.**
- **For each combination, perform 5-fold stratified cross-validation:**
    - **For each fold, train the pipeline on the training split and evaluate on the validation split.**
    - **Collect the ROC AUC scores and average across folds.**
- **Select the hyperparameter combination with the highest mean ROC AUC.**

- **Fit the final pipeline with the best parameters on the entire training dataset.**