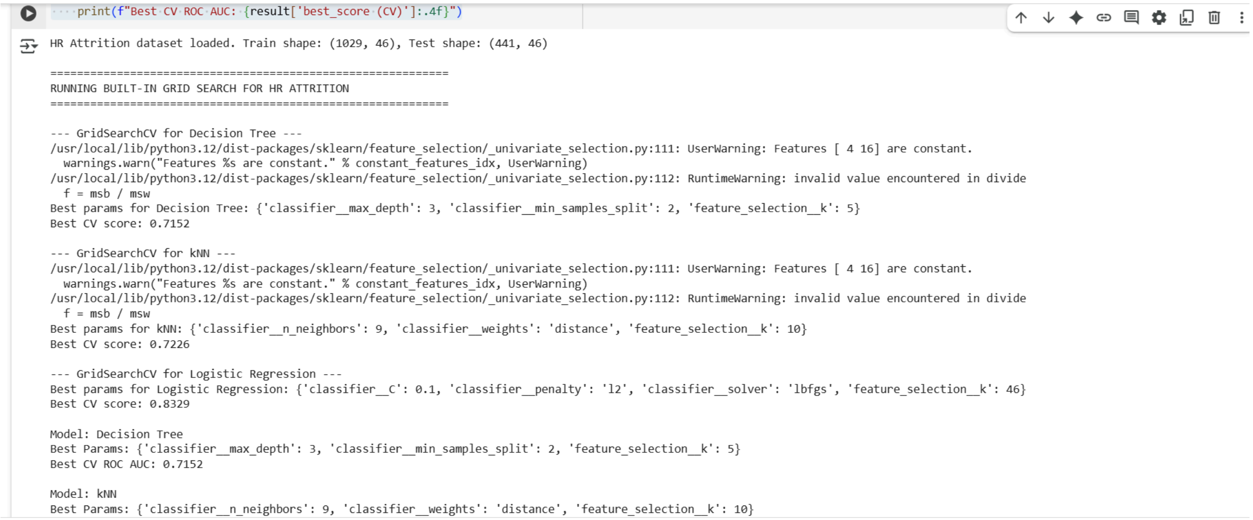
```
import pandas as pd
import numpy as np
import itertools
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, StratifiedKFold,
GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.pipeline 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']
}

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}")
```



```python
    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}
```