

```

import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.feature_selection import RFE, SelectKBest, f_classif
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
    fl_score
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier

===== #
Load dataset (\ #
===== #
    "file_path = "StudentsPerformance.csv"
    df = pd.read_csv(file_path)

    (\ * "=")print
    print("Dataset Preview")
    (\ * "=")print
    print(df.head())
    print("\nShape:", df.shape)

===== #
Handle missing values and prepare data (\ #
===== #
    print("\n" + "=" * 70)
    print("Missing Values")
    (\ * "=")print
    print(df.isnull().sum())

:Create a classification target #
    math_pass = 1 if math score >= 50, else 0 #
    Drop 'math score' to avoid target leakage #
df["math_pass"] = np.where(df["math score"] >= 50, 1, 0)
X = df.drop(columns=["math_pass", "math score"])
y = df["math_pass"]

categorical_features =
()X.select_dtypes(include=["object"]).columns.tolist
()numerical_features = X.select_dtypes(exclude=["object"]).columns.tolist

)numeric_transformer = Pipeline
    ]=steps
, (imputer", SimpleImputer(strategy="median"))
, ((scaler", StandardScaler")

[
(

) categorical_transformer = Pipeline
    ]=steps
, (imputer", SimpleImputer(strategy="most_frequent"))

```

```

onehot", OneHotEncoder(handle_unknown="ignore", ")
                                , (sparse_output=False)
                                [
                                    (

                                )preprocessor = ColumnTransformer
                                ]=transformers
                                , (num", numeric_transformer, numerical_features")
                                , (cat", categorical_transformer, categorical_features")
                                , [
                                    , sparse_threshold=0
                                (

                                )X_train, X_test, y_train, y_test = train_test_split
                                X, y, test_size=0.2, random_state=42, stratify=y
                                (

===== #
                                Apply machine learning models with GridSearchCV (r #
===== #
                                } = models
                                } : "Logistic Regression"
                                , model": LogisticRegression(max_iter=1000)"
                                } : "params"
                                [\ , . , \] : "classifier__C"
                                , {
                                    , {
                                        } : "KNN"
                                    , ()model": KNeighborsClassifier"
                                    } : "params"
                                , [\ , *] : "classifier__n_neighbors"
                                , classifier__weights": ["uniform", "distance"]"
                                , {
                                    , {
                                        } : "Decision Tree"
                                    , model": DecisionTreeClassifier(random_state=42)"
                                    } : "params"
                                , classifier__max_depth": [3, 5, None]"
                                , {
                                    , {
                                        {

                                [] = results_before_fs

                                print("\n" + "=" * 70)
                                print("Model Evaluation on Cleaned Dataset")
                                (v. * "=")print

                                : ()for model_name, config in models.items
                                )pipeline = Pipeline
                                ]=steps
                                , (preprocessor", preprocessor")
                                , (classifier", config["model"])"
                                [
                                    (

                                )grid_search = GridSearchCV
                                , estimator=pipeline

```

```

        ,param_grid=config["params"]
        ,cv=3
        ,"scoring="accuracy
        ,n_jobs=1
    (

        grid_search.fit(X_train, y_train)
        y_pred = grid_search.predict(X_test)

        acc = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)

        )results_before_fs.append
    }

    ,Model": model_name"
    ,_Best Params": grid_search.best_params"
    ,Accuracy": round(acc, 4)"
    ,F1 Score": round(f1, 4)"
    {
        (

            print(f"\n{model_name}")
            print("Best Params:", grid_search.best_params_)
            print("Accuracy:", round(acc, 4))
            print("F1 Score:", round(f1, 4))
        print("Classification Report:\n", classification_report(y_test,
                                                                    y_pred))

        results_before_fs_df = pd.DataFrame(results_before_fs)
        print("\nSummary Before Feature Selection:")
        print(results_before_fs_df)

        ===== #
        Two feature selection methods: SelectKBest and RFE (3 #
        ===== #
        X_train_processed = preprocessor.fit_transform(X_train)
        X_test_processed = preprocessor.transform(X_test)
        ()feature_names = preprocessor.get_feature_names_out

        kbest_selector = SelectKBest(score_func=f_classif, k=8)
        X_train_kbest = kbest_selector.fit_transform(X_train_processed, y_train)
        X_test_kbest = kbest_selector.transform(X_test_processed)
        selected_kbest_features = feature_names[kbest_selector.get_support()]

        )rfe_selector = RFE
        ,estimator=LogisticRegression(max_iter=1000)
        ,n_features_to_select=8
    (
        X_train_rfe = rfe_selector.fit_transform(X_train_processed, y_train)
        X_test_rfe = rfe_selector.transform(X_test_processed)
        selected_rfe_features = feature_names[rfe_selector.get_support()]

        print("\n" + "=" * 70)
        print("Selected Features")
        (v * "=")print

        print("\nTop features using SelectKBest:")
        :for feature in selected_kbest_features

```

```

print("-", feature)

print("\nTop features using RFE:")
:for feature in selected_rfe_features
    print("-", feature)

===== #
    Apply models to selected features and evaluate them (o #
===== #
                                } = models_fs
,Logistic Regression": LogisticRegression(max_iter=1000)"
    ,KNN": KNeighborsClassifier(n_neighbors=7)"
    Decision Tree": DecisionTreeClassifier(max_depth=3, "
                                ,random_state=42)
                                {

                                [] = results_after_fs

                                :()for model_name, model in models_fs.items
                                    model.fit(X_train_kbest, y_train)
                                    y_pred = model.predict(X_test_kbest)
                                    )results_after_fs.append
                                    }

                                , "Feature Selection": "SelectKBest"
                                    ,Model": model_name"
,Accuracy": round(accuracy_score(y_test, y_pred), 4)"
    ,F1 Score": round(f1_score(y_test, y_pred), 4)"
                                {
                                    (

                                :()for model_name, model in models_fs.items
                                    model.fit(X_train_rfe, y_train)
                                    y_pred = model.predict(X_test_rfe)
                                    )results_after_fs.append
                                    }

                                , "Feature Selection": "RFE"
                                    ,Model": model_name"
,Accuracy": round(accuracy_score(y_test, y_pred), 4)"
    ,F1 Score": round(f1_score(y_test, y_pred), 4)"
                                {
                                    (

results_after_fs_df = pd.DataFrame(results_after_fs)

                                print("\n" + "=" * 70)
                                print("Results After Feature Selection")
                                    (v * "=")print
                                print(results_after_fs_df)

===== #
                                Final notes #
===== #
                                print("\n" + "=" * 70)
                                print("Final Notes")
                                    (v * "=")print
                                print("1. The dataset was loaded successfully from CSV.")
print("2. Missing values were checked and a preprocessing pipeline was
                                prepared.")

```

```
print("3. Multiple ML models were tuned using GridSearchCV.")
print("4. Two feature selection methods were applied: SelectKBest and
                                             RFE.")
print("5. The models were evaluated again on the selected features.")
print("6. Save this script using your full name and submit it as PDF.")
```