

0 = Low إذا Final Mark < 50
 1 = Medium إذا 50 <= Final Mark < 75
 2 = High إذا Final Mark >= 75

وهذا كود كامل ومرتب يطبق الفكرة على الملف نفسه، ويعطيك تأكيد عملي عبر

تنظيف البيانات
 تحويل الأعمدة المئوية
 إنشاء Final_Class
 تقسيم البيانات
 تدريب أكثر من موديل
 تقييمها بـ Accuracy / Precision / Recall
 وتجربة Stacking

```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score, precision_score, recall_score

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, StackingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

# =====
# 1) Load dataset
# =====
file_path = "Students_general_report_DB_ready 19 March.csv"
df = pd.read_csv(file_path)

print("Dataset shape:", df.shape)
print("\nColumns:")
print(df.columns.tolist())

# =====
# 2) Clean percentage columns
# =====
# %الأعمدة التي فيها % كنصوص مثل 100
percent_cols = [
    'Homeworks Participation %',
    'Contents Participation %',
    'Virtual Classrooms Participation %',
    'AVG Participations %',
    'Discussion Rooms Participation %',
    'Mailbox Participation %',
    'Student login %'
]

for col in percent_cols:
    if col in df.columns:
        df[col] = (
            df[col]
```

```

        .astype(str)
        .str.replace('%', '', regex=False)
        .str.strip()
    )
    df[col] = pd.to_numeric(df[col], errors='coerce')

```

Dataset shape: (1395, 18)

Columns:

['No. active courses', 'Exams', 'Exams Participation %', 'Homeworks', 'Homeworks Participation %']

```

# =====
# 3) Make sure all remaining columns are numeric if possible
# =====
for col in df.columns:
    if col != 'Final Mark':
        df[col] = pd.to_numeric(df[col], errors='ignore')

```

/tmp/ipykernel_522/1076553039.py:6: FutureWarning: errors='ignore' is deprecated and will raise :
df[col] = pd.to_numeric(df[col], errors='ignore')

```

# =====
# 4) Create classification target from Final Mark
# =====
def classify_final_mark(x):
    if pd.isna(x):
        return np.nan
    elif x < 50:
        return 0 # Low / At Risk
    elif x < 75:
        return 1 # Medium
    else:
        return 2 # High

df['Final_Class'] = df['Final Mark'].apply(classify_final_mark)

# مفقود target حذف أي صف فيه
df = df.dropna(subset=['Final_Class']).copy()
df['Final_Class'] = df['Final_Class'].astype(int)

print("\nClass distribution:")
print(df['Final_Class'].value_counts().sort_index())
print("\nClass proportions:")
print(df['Final_Class'].value_counts(normalize=True).sort_index())

```

Class distribution:
Final_Class
0 241
1 828
2 326
Name: count, dtype: int64

Class proportions:
Final_Class
0 0.172760
1 0.593548
2 0.233692

Name: proportion, dtype: float64

```
# =====
# 5) Define X and y
# =====
# الأصلي target لأنه feature لا نستخدم
X = df.drop(columns=['Final Mark', 'Final_Class'])
y = df['Final_Class']

print("\nFeature columns:")
print(X.columns.tolist())
```

Feature columns:
['No. active courses', 'Exams', 'Exams Participation %', 'Homeworks', 'Homeworks Participation %']

```
# =====
# 5) Define X and y
# =====
# الأصلي target لأنه feature لا نستخدم
X = df.drop(columns=['Final Mark', 'Final_Class'])
y = df['Final_Class']

print("\nFeature columns:")
print(X.columns.tolist())
```

Feature columns:
['No. active courses', 'Exams', 'Exams Participation %', 'Homeworks', 'Homeworks Participation %']

```
# =====
# 6) Train / Test Split
# =====
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.20,
    random_state=42,
    stratify=y
)

print("\nTrain shape:", X_train.shape)
print("Test shape:", X_test.shape)
```

Train shape: (1116, 17)
Test shape: (279, 17)

```
# 7) Preprocessing
# =====
numeric_features = X.columns.tolist()

numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
```

```

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features)
    ]
)

```

```

# =====
# 8) Base models
# =====
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42, max_depth=5),
    "Random Forest": RandomForestClassifier(
        n_estimators=200,
        random_state=42,
        class_weight='balanced'
    ),
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "Logistic Regression": LogisticRegression(
        max_iter=2000,
        class_weight='balanced'
    )
}

# =====
# 9) Train and evaluate each model
# =====
for name, model in models.items():
    clf = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('classifier', model)
    ])

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

    print("\n" + "="*60)
    print(f"Model: {name}")
    print("="*60)
    print("Accuracy :", round(accuracy_score(y_test, y_pred), 4))
    print("Precision:", round(precision_score(y_test, y_pred, average='weighted', zero_division=0), 4))
    print("Recall   :", round(recall_score(y_test, y_pred, average='weighted', zero_division=0), 4))
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred, zero_division=0))
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

```

```
Model: KNN
```

```
Accuracy : 0.767
```

```
Precision: 0.7679
```

```
Recall : 0.767
```

```
Classification Report:
```

	precision	recall	f1-score	support
0	0.73	0.62	0.67	48
1	0.77	0.93	0.84	166
2	0.79	0.46	0.58	65
accuracy			0.77	279
macro avg	0.76	0.67	0.70	279
weighted avg	0.77	0.77	0.75	279

```
Confusion Matrix:
```

```
[[ 30 16  2]
 [  6 154  6]
 [  5  30 30]]
```

```
Model: Logistic Regression
```

```
Accuracy : 0.6452
```

```
Precision: 0.6838
```

```
Recall : 0.6452
```

```
Classification Report:
```

	precision	recall	f1-score	support
0	0.47	0.71	0.56	48
1	0.81	0.66	0.73	166
2	0.51	0.57	0.54	65
accuracy			0.65	279
macro avg	0.60	0.64	0.61	279
weighted avg	0.68	0.65	0.65	279

```
Confusion Matrix:
```

```
[[ 34 10  4]
 [ 26 109 31]
 [ 13 15 37]]
```

```
# =====
# 10) Stacking Ensemble
# =====
estimators = [
    ('dt', DecisionTreeClassifier(random_state=42, max_depth=5)),
    ('rf', RandomForestClassifier(
        n_estimators=200,
        random_state=42,
        class_weight='balanced'
    )),
    ('knn', KNeighborsClassifier(n_neighbors=5))
]

stacking_model = StackingClassifier(
    estimators=estimators,
    final_estimator=LogisticRegression(max_iter=2000, class_weight='balanced'),
    passthrough=False
```

```

)

stack_clf = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', stacking_model)
])

stack_clf.fit(X_train, y_train)
y_pred_stack = stack_clf.predict(X_test)

print("\n" + "="*60)
print("Model: Stacking Ensemble")
print("="*60)
print("Accuracy :", round(accuracy_score(y_test, y_pred_stack), 4))
print("Precision:", round(precision_score(y_test, y_pred_stack, average='weighted', zero_division=0), 4))
print("Recall   :", round(recall_score(y_test, y_pred_stack, average='weighted', zero_division=0), 4))
print("\nClassification Report:")
print(classification_report(y_test, y_pred_stack, zero_division=0))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_stack))

```

```

=====
Model: Stacking Ensemble
=====
Accuracy : 0.7921
Precision: 0.7899
Recall   : 0.7921

Classification Report:

```

	precision	recall	f1-score	support
0	0.74	0.58	0.65	48
1	0.85	0.89	0.87	166
2	0.68	0.71	0.69	65
accuracy			0.79	279
macro avg	0.75	0.73	0.74	279
weighted avg	0.79	0.79	0.79	279

```

Confusion Matrix:
[[ 28  13   7]
 [  4 147  15]
 [  6  13  46]]

```

Start coding or [generate](#) with AI.

