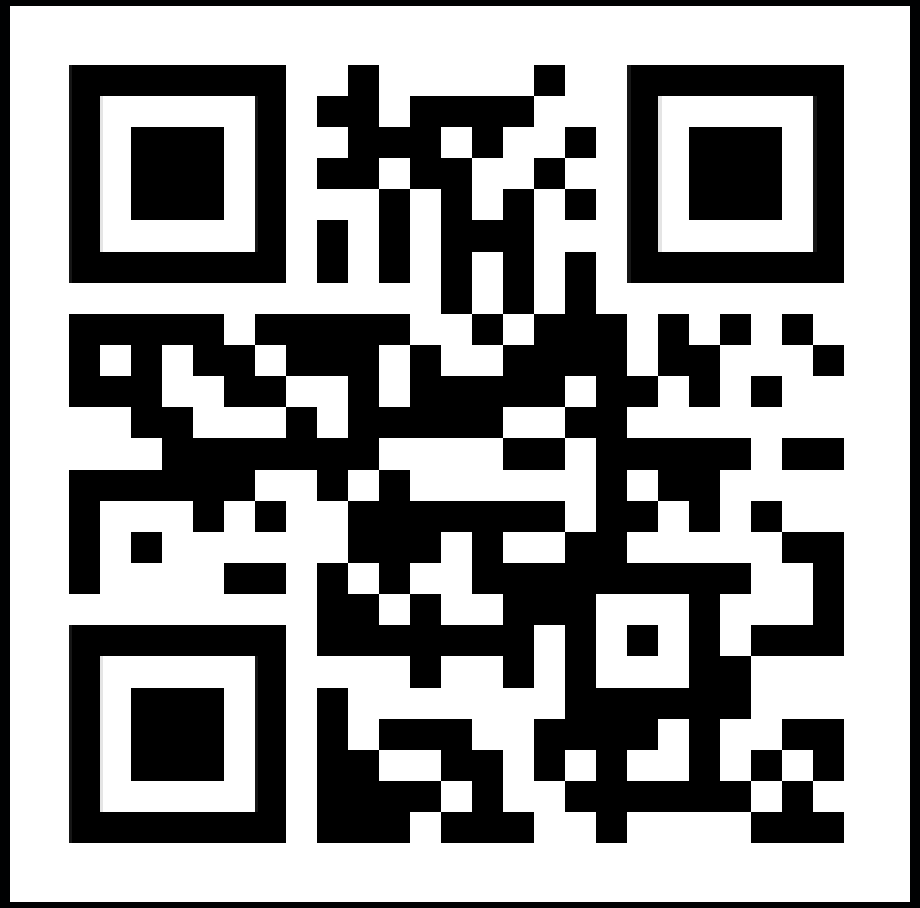
BENHISLOP.COM

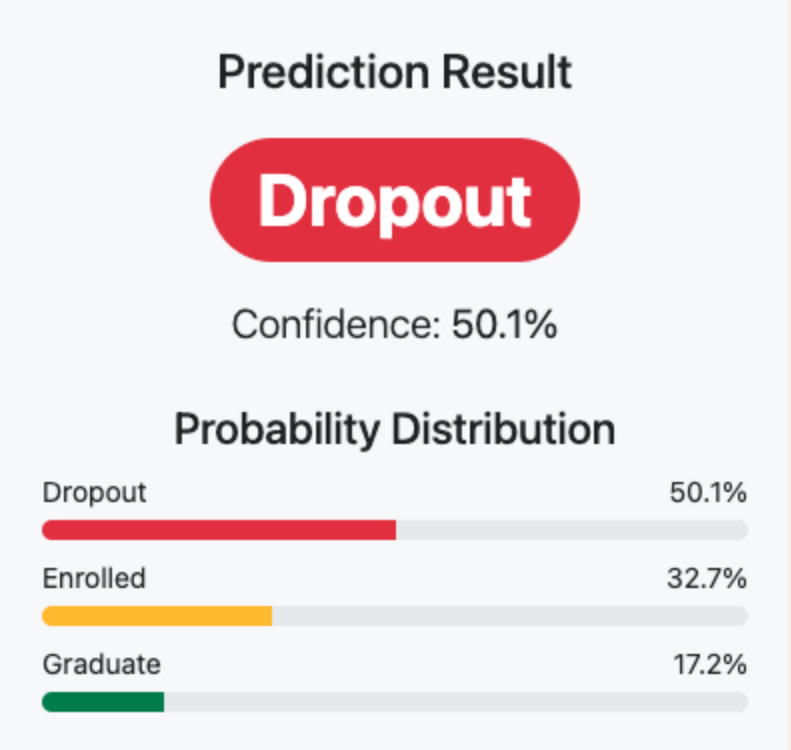


STUDENT DROPOUT RISK PREDICTOR

Analyzing student data to prevent academic dropouts

INTRODUCTION TO THE PROBLEM AND SOLUTION

- Problem of Student Dropout
- Early Risk Identification
- Random Forest Classifier



EXPLORING AND PREPARING THE DATASET

ATTRIBUTE	DESCRIPTION
Source	UCI Machine Learning Repository
Records	4,424 students
Features	36 (Demographic, Socio-economic, Academic)
Target Classes	Dropout, Enrolled, Graduate
Preprocessing	CSV parsing, and Data Auditing

```
----------------------
Missing Values (top 5):
----------------------

Marital status                              0
Age at enrollment                           0
Curricular units 1st sem (credited)         0
Curricular units 1st sem (enrolled)         0
Curricular units 1st sem (evaluations)      0
dtype: int64


----------------------
Duplicate rows: 0
----------------------


Top 10 Most Important Features:
1. Curricular units 2nd sem (approved) (0.2029)
2. Curricular units 2nd sem (grade) (0.1304)
3. Curricular units 1st sem (approved) (0.1081)
4. Curricular units 1st sem (grade) (0.0772)
5. Tuition fees up to date (0.0590)
6. Curricular units 2nd sem (evaluations) (0.0501)
7. Curricular units 1st sem (evaluations) (0.0464)
8. Age at enrollment (0.0373)
9. Course (0.0282)
10. Curricular units 2nd sem (enrolled) (0.0274)
```

CHOOSING AND TUNING THE RANDOM FOREST MODEL

Random Forest handles complex non-linear relationships and reduces overfitting by aggregating multiple decision trees.

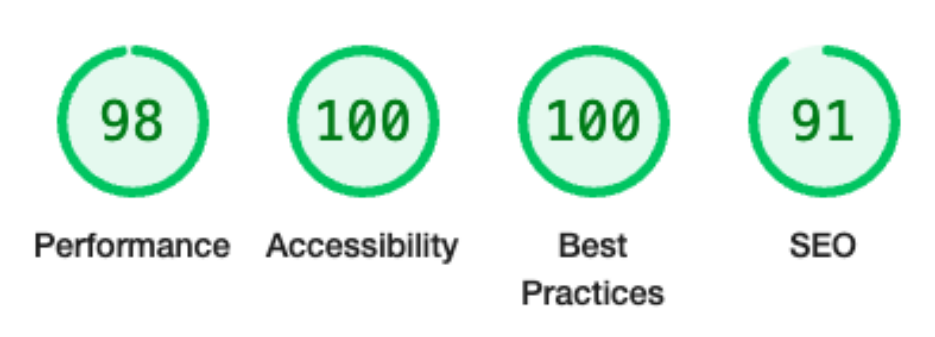
Grid Search w/ Cross-Validation to prevent overfitting

Used balanced_subsample to weight rare classes equally.

Optimized for Macro F1 Score

```
for max_depth, min_samples_leaf, class_weight in product(
    param_grid["max_depth"],
    param_grid["min_samples_leaf"],
    param_grid["class_weight"],
):
    candidate = RandomForestClassifier(
        n_estimators=200,
        random_state=42,
        max_depth=max_depth,
        min_samples_leaf=min_samples_leaf,
        class_weight=class_weight,
    )
    candidate.fit(X_train_small, y_train_small)

    y_val_pred = candidate.predict(X_val)

    # Macro F1: treats each class equally (Dropout / Enrolled / Graduate)
    macro_f1 = f1_score(y_val, y_val_pred, average="macro")

    results.append(
        {
            "max_depth": max_depth,
            "min_samples_leaf": min_samples_leaf,
            "class_weight": class_weight,
            "macro_f1": macro_f1,
        }
    )

# Sort by macro F1 descending and show the top few
results_sorted = sorted(results, key=lambda r: r["macro_f1"], reverse=True)
```

PERFORMANCE METRICS AND KEY INSIGHTS



METRIC	VALUE
Accuracy	74%
Macro F1 Score	0.70
F1 (Graduate)	0.84
F1 (Dropout)	0.75
F1 (Enrolled)	0.51

REFLECTION AND FUTURE DIRECTIONS

What did I learn
- Understanding of Random Forests

App enhancements
- Data retention
- User Authentication