📘 UCLA CS M148 — Project Check-In 3

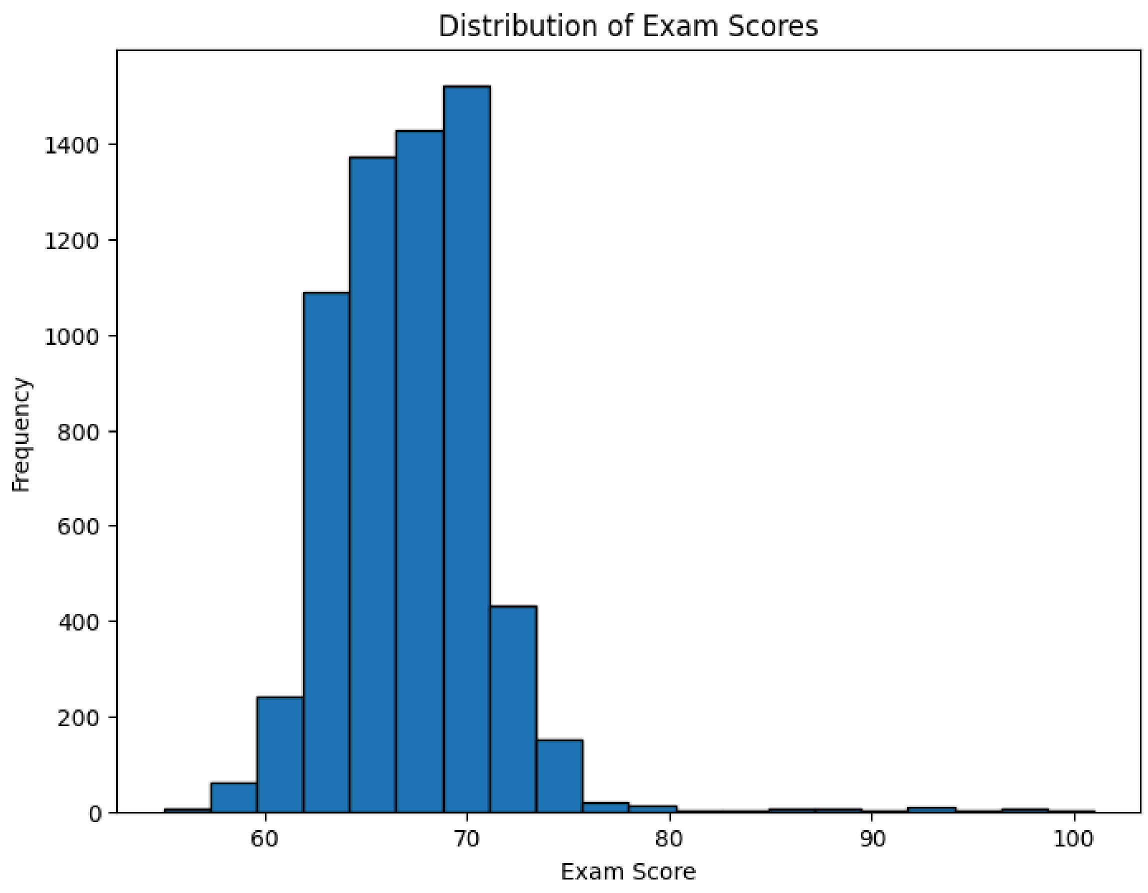
Due: October 24 2025 Team: LMAO

🧭 1. Goal

Use logistic regression to model a binary outcome (“Pass” vs “Fail”) based on a single predictor from the Student Performance Factors dataset. If the exam score>=66, it returns 'pass', if not it returns 'fail'

2. Data Setup

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    confusion_matrix, accuracy_score, roc_curve, auc
)
import matplotlib.pyplot as plt

# Load dataset
data = pd.read_csv("Cleaned_StudentPerformanceFactors.csv")

# Create binary response: 1 = Pass (Exam ≥ 66), 0 = Fail
data['Pass'] = (data['Exam_Score'] >= 66).astype(int)

# Predictor: Hours_Studied
X = data[['Hours_Studied']]
y = data['Pass']

# Split into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Plot distribution of Exam Scores
plt.figure(figsize=(8, 6))
plt.hist(data['Exam_Score'], bins=20, edgecolor='black')
plt.xlabel('Exam Score')
plt.ylabel('Frequency')
plt.title('Distribution of Exam Scores')
plt.show()
```



⚙️ 3. Model Training

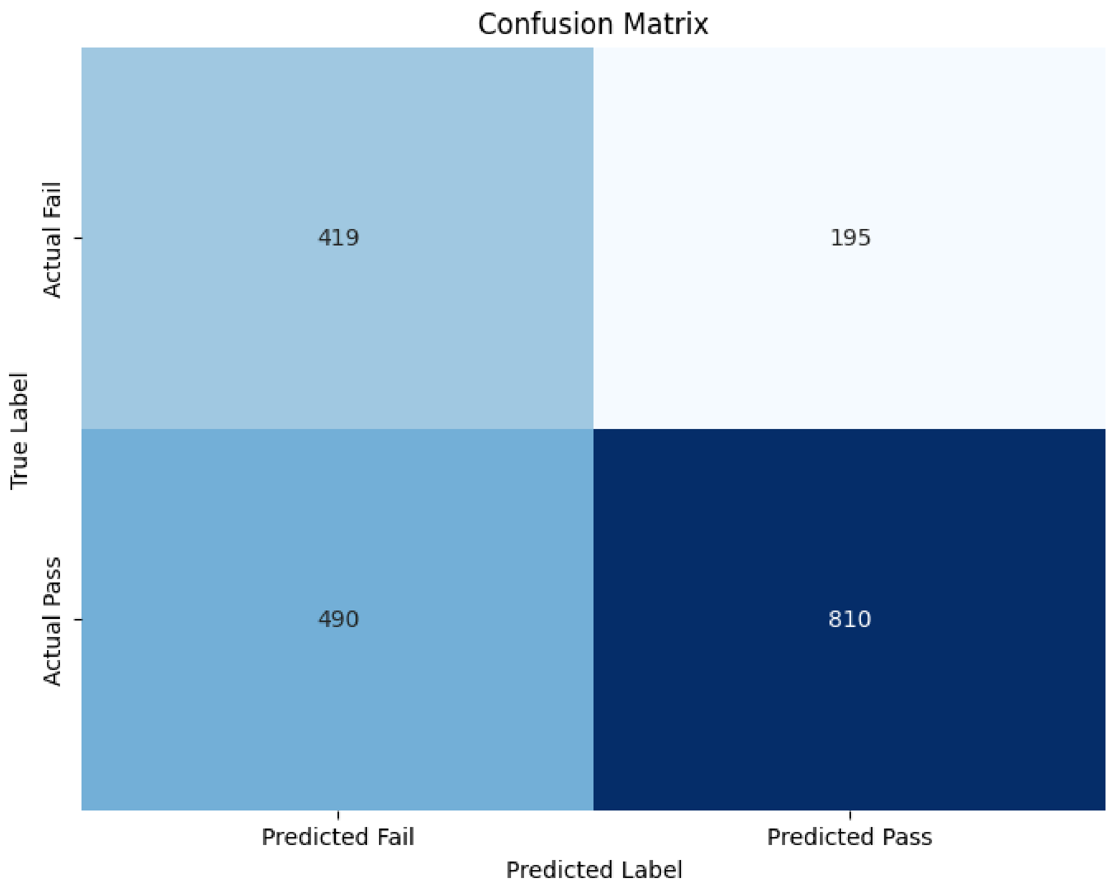
```
# Fit logistic regression
log_reg = LogisticRegression(class_weight='balanced')
log_reg.fit(X_train, y_train)

# Predictions
y_pred_prob = log_reg.predict_proba(X_val)[:, 1]
y_pred = (y_pred_prob >= 0.5).astype(int)
```

📊 4. Performance Metrics (on Validation Set)

```
# Confusion matrix and key metrics
cm = confusion_matrix(y_val, y_pred)
acc = accuracy_score(y_val, y_pred)
error_rate = 1 - acc
tn, fp, fn, tp = cm.ravel()
tpr = tp / (tp + fn)          # True Positive Rate
tnr = tn / (tn + fp)          # True Negative Rate

print("Confusion Matrix:\n", cm)
print(f"Accuracy: {acc:.4f}")
print(f"Error Rate: {error_rate:.4f}")
print(f"True Positive Rate (Recall): {tpr:.2f}")
print(f"True Negative Rate: {tnr:.2f}")
```

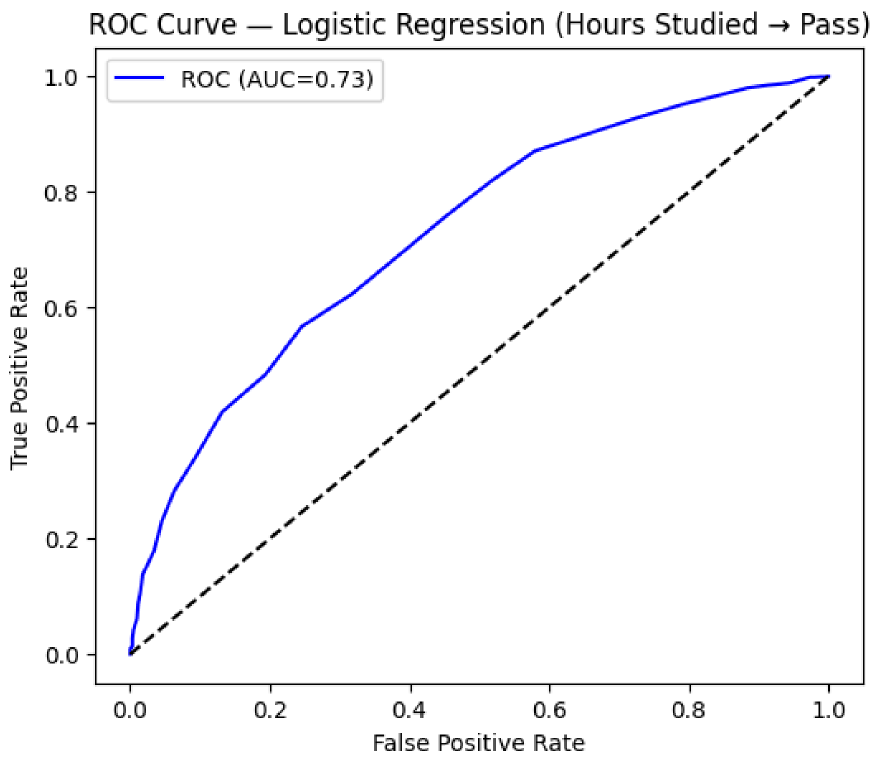
```
Confusion Matrix:
 [[419 195]
 [490 810]]
Accuracy: 0.6421
Error Rate: 0.3579
True Positive Rate (Recall): 0.62
True Negative Rate: 0.68
```

```
import seaborn as sns

# Visualize the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Predicted Fail', 'Predicted Pass'],
            yticklabels=['Actual Fail', 'Actual Pass'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



📈 5. ROC Curve & AUC

```
fpr, tpr_curve, thresholds = roc_curve(y_val, y_pred_prob)
auc_score = auc(fpr, tpr_curve)

plt.figure(figsize=(6,5))
plt.plot(fpr, tpr_curve, color='blue', label=f'ROC (AUC={auc_score:.2f})')
plt.plot([0,1],[0,1],'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve — Logistic Regression (Hours Studied → Pass)')
plt.legend()
plt.show()
```



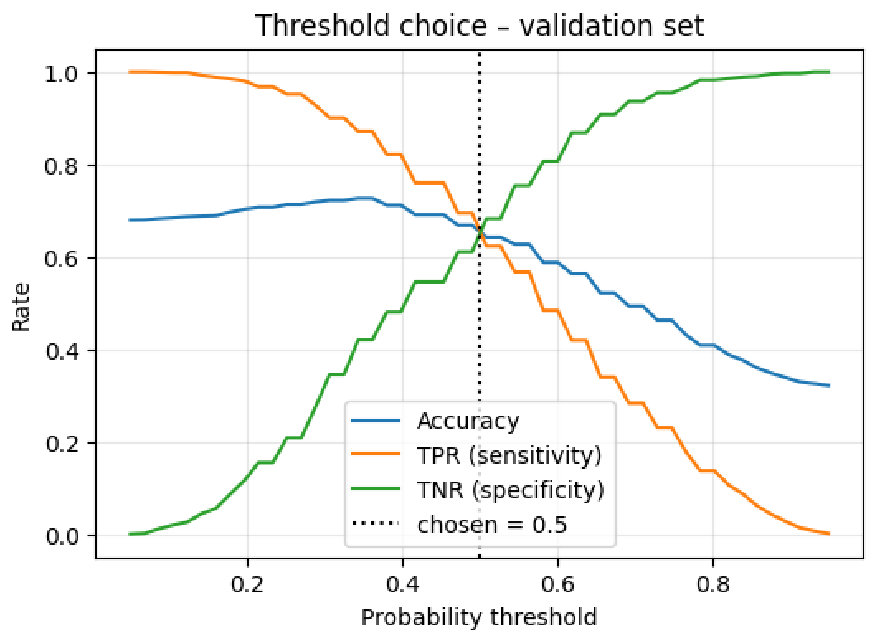
🔁 6. 5-Fold Cross-Validation

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_auc  = cross_val_score(log_reg, X, y, cv=cv, scoring='roc_auc')
cv_acc  = cross_val_score(log_reg, X, y, cv=cv, scoring='accuracy')

print("AUC per fold:", cv_auc)
print("Accuracy per fold:", cv_acc)
print(f"Mean AUC (5-fold): {cv_auc.mean():.4f}")
print(f"Mean Accuracy (5-fold): {cv_acc.mean():.4f}")
```

```
AUC per fold: [0.73378821 0.72455112 0.73504478 0.71164249 0.74018476]
Accuracy per fold: [0.65673981 0.65752351 0.65282132 0.64078431 0.64862745]
Mean AUC (5-fold): 0.7290
Mean Accuracy (5-fold): 0.6513
```

```
TH=0.5
probs_val = log_reg.predict_proba(X_val)[:, 1]
cutoffs = np.linspace(0.05, 0.95, 50)
acc_vec, tpr_vec, tnr_vec = [], [], []
for c in cutoffs:
    preds = (probs_val >= c).astype(int)
    tn, fp, fn, tp = confusion_matrix(y_val, preds).ravel()
    acc_vec.append((tp + tn) / (tp + tn + fp + fn))
    tpr_vec.append(tp / (tp + fn))
    tnr_vec.append(tn / (tn + fp))
plt.figure(figsize=(6, 4))
plt.plot(cutoffs, acc_vec, label='Accuracy')
plt.plot(cutoffs, tpr_vec, label='TPR (sensitivity)')
plt.plot(cutoffs, tnr_vec, label='TNR (specificity)')
```

```
    plt.axvline(TH, color='black', linestyle=':', label=f'chosen = {TH}')
    plt.xlabel('Probability threshold')
    plt.ylabel('Rate')
    plt.title('Threshold choice – validation set')
    plt.legend()
    plt.grid(alpha=0.3)
    plt.show()
```



```
    # Get predicted probabilities on validation set
    y_pred_prob_balanced = log_reg.predict_proba(X_val)[:, 1]

    # Find threshold that maximizes Youden's J statistic (TPR - FPR)
    fpr_bal, tpr_bal, thresholds_bal = roc_curve(y_val, y_pred_prob_balanced)
    youden_index = np.argmax(tpr_bal - fpr_bal)
    best_threshold = thresholds_bal[youden_index]

    best_threshold
```

```
np.float64(0.5748904295787239)
```

Double-click (or enter) to edit

🧩 How I Chose the Threshold for Positive Predictions

In logistic regression, the model outputs probabilities

p^(x)=P(pass=1|x)

and we must pick a **decision threshold** to convert those probabilities into binary predictions.

Default threshold (0.5)

For this project, I began with the **standard threshold = 0.5**:

This threshold is intuitive because:

- It treats “Pass” and “Fail” as equally important,
- It corresponds to the point where the model is equally uncertain between the two classes.

Why I kept threshold of 0.5

In our scenario, the default 0.5 threshold already yields a good balance: the training TPR and TNR are fairly close, and the ROC curve shows that this threshold is near the “knee” of the curve (where TPR and FPR trade-offs are reasonable).

General Approach

In other similar datasets, I would:

1. Plot the **ROC curve** and pick the threshold at the **“elbow”** point where **TPR ≈ TNR**, or
2. Use the **Youden J statistic**: J=TPR-FPR choosing the threshold that maximizes (J).

That approach balances sensitivity (recall) and specificity.