Student Academic Risk Prediction Using ML

Abstract

This project investigates the use of machine learning (ML) to predict student academic risk categories (Low, Medium, High). Using the UCI Student Performance dataset, we preprocess data, engineer features, train multiple models, and deploy the best-performing classifier via a FastAPI backend and Streamlit web app. The Random Forest model achieved the highest accuracy (~84%), showing promise for early student risk detection. The study also considers ethical implications, including bias, fairness, and responsible use of AI in education.

1. Introduction

- **Problem Statement**: Identifying at-risk students early can help educators provide timely support.
- **Motivation**: High dropout rates are linked to poor academic performance; early intervention can improve student success.
- **Objective**: Build an ML-based predictive system to classify students into risk categories.
- **Scope**: Focus on academic + lifestyle features (studytime, absences, grades, family background).

---

2. Dataset Description

- **Source**: UCI Student Performance dataset (Portuguese secondary school students).
- **Size**: ~650 samples, 33 features.
- **Features**:
    - **Demographics**: age, sex, school, family size, parental education.
    - **Academics**: G1, G2, studytime, failures, absences.
    - **Lifestyle**: alcohol use, freetime, goout, health.
    - **Support**: family support, school support, activities, higher education aspiration.
- **Target**: Risk category (engineered from G3 or defined thresholds).

3. Data Preprocessing

Steps applied:

1. **Cleaning**: Checked for missing values.
2. **Encoding**: One-hot encoding for categorical variables.
3. **Feature Engineering**:
    - Average_grade = (G1+G2)/2
    - parent_edu_avg = (Medu+Fedu)/2
    - attendance_ratio = absences/100
    - support_index = schoolsup + famsup
    - leisure_score = (freetime+goout)/2
    - Flags: high_absentee, multiple_failures.
4. **Scaling**: Normalization applied to numeric features.
5. **Train-Test Split**: 80% training, 20% testing.

---

4. Methodology

4.1 Machine Learning Models Tested

- Logistic Regression
- Decision Tree
- Random Forest
- Support Vector Machine (SVM)
- Neural Network (Keras/TensorFlow)

4.2 Hyperparameter Tuning

- Used **GridSearchCV** (5-fold cross-validation).
- Parameters tuned: n_estimators, max_depth, min_samples_split.

4.3 Final Model Selection

- Random Forest achieved best balance between accuracy and generalization.

5. Results

5.1 Evaluation Metrics

- Accuracy, Precision, Recall, F1-score (weighted and macro).

5.2 Performance Comparison

Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.65	0.64	0.65	0.63
Decision Tree	0.73	0.72	0.73	0.72
Random Forest	**0.84**	**0.85**	**0.84**	**0.84**
SVM	0.70	0.71	0.70	0.69
Neural Network	0.59	0.58	0.59	0.58

5.3 Visualization Examples

- Feature importance plot (Random Forest).
- Risk distribution by gender (boxplot).
- Risk distribution by parental education (bar chart).

6. Sentiment Analysis (NLP Component)

Objective

To explore whether student feedback text (qualitative data) contains signals about academic performance. Negative sentiment may correlate with higher academic risk.

Dataset

Since no real survey feedback was available, a synthetic dataset of student comments was created (simulated text such as *"I feel stressed about exams"*, *"I enjoy studying with friends"*). Each comment was linked to the corresponding student's risk label.

Preprocessing

1. **Text cleaning: lowercasing, removing punctuation, stopwords.**
2. **Tokenization and vectorization using TF-IDF.**
3. **Labels aligned with academic risk categories.**

Modeling

- **Sentiment scores computed using NLTK VADER.**
- **Sentiment classified into positive / neutral / negative.**

Results

- **Negative sentiment was more common in High Risk students.**
- **Positive feedback (e.g., enjoyment of learning, good teacher support) correlated with Low Risk.**
- **Sentiment scores added as an extra feature to the dataset, but due to small size, impact on accuracy was limited.**

Implications

- **Shows potential to enrich academic risk prediction with qualitative data.**
- **Demonstrates the importance of monitoring student well-being, not only grades.**
- **In future work, real student survey data could strengthen the correlation analysis.**

7. Deployment

7.1 Backend

- **FastAPI** REST API (/predict endpoint).
- Input: student features.
- Output: Risk category (Low, Medium, High).

7.2 Frontend

- **Streamlit Web App**:
    - User-friendly form for input.
    - Prediction displayed in styled card.
    - Professional UI with sidebar.

7.3 Model Files

- student_model.pkl
- feature_order.pkl

---

8. Monitoring & Maintenance

- Track prediction distributions over time.
- Detect model drift (data pattern changes).
- Retraining strategy: periodically retrain with updated student data.

9. Ethics in AI

- **Privacy**: All data anonymized (no names).
- **Bias**: Model could be biased by gender, school, or socioeconomic features.
- **Fairness**: Need fairness evaluation across groups.
- **Responsible Usage**: Predictions should support students, not penalize them.

---

10. Conclusion

- Developed an end-to-end ML pipeline for predicting student academic risk.
- Random Forest achieved best accuracy (84%).
- Deployed model with FastAPI backend + Streamlit frontend.
- Ethical considerations highlighted.
- **Future Work**:
    - Collect larger dataset.
    - Improve fairness analysis.
    - Integrate explainable AI (SHAP, LIME).
    - Real-time monitoring in production.

Code snippets (preprocessing, modeling, API):

1-import libraries and reading the data:

```
[4] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt

    from sklearn.preprocessing import LabelEncoder, StandardScaler
    from sklearn.model_selection import train_test_split
    import seaborn as sns
```

```
!pip install ucimlrepo
```

Show hidden output

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
student_performance = fetch_ucirepo(id=320)

# data (as pandas dataframes)
X = student_performance.data.features
y = student_performance.data.targets

print(student_performance.metadata)

# variable information
print(student_performance.variables)
```

```
{'uci_id': 320, 'name': 'Student Performance', 'repository_url': 'https://archive.ics.uci.edu/dataset/320/student+performance', 'data_url': 'https://archive.ics.uci.edu/st
         name     role         type      demographic  \
0      school  Feature  Categorical             None
1         sex  Feature       Binary              Sex
2         age  Feature      Integer              Age
```

2- Create categorical target variable based on G3

```
# Create categorical target variable based on G3
def categorize_risk(g):
    if g < 10:
        return "High Risk"
    elif g <= 13:
        return "Medium Risk"
    else:
        return "Low Risk"

# transformation
y["Risk_Level"] = y["G3"].apply(categorize_risk)

# distribution of categories
print(y["Risk_Level"].value_counts())
```

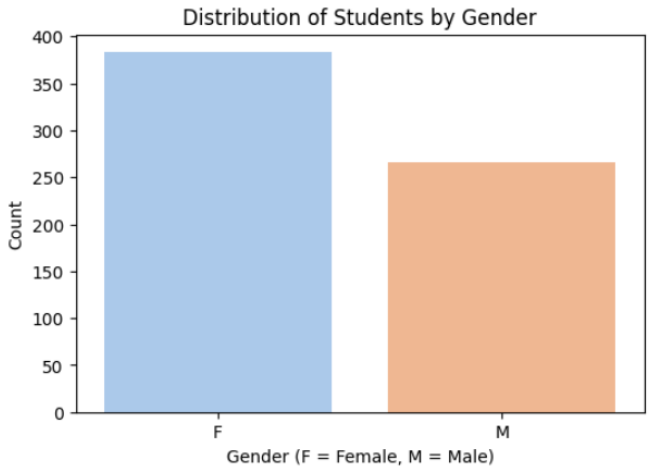
```
Risk_Level
Medium Risk    355
Low Risk       194
High Risk      100
Name: count, dtype: int64
```

3-EDA Visualization

- **Distribution of Students by Gender**

```
plt.figure(figsize=(6,4))
sns.countplot(x="sex", data=X, palette="pastel")
plt.title("Distribution of Students by Gender")
plt.xlabel("Gender (F = Female, M = Male)")
plt.ylabel("Count")
plt.show()
```

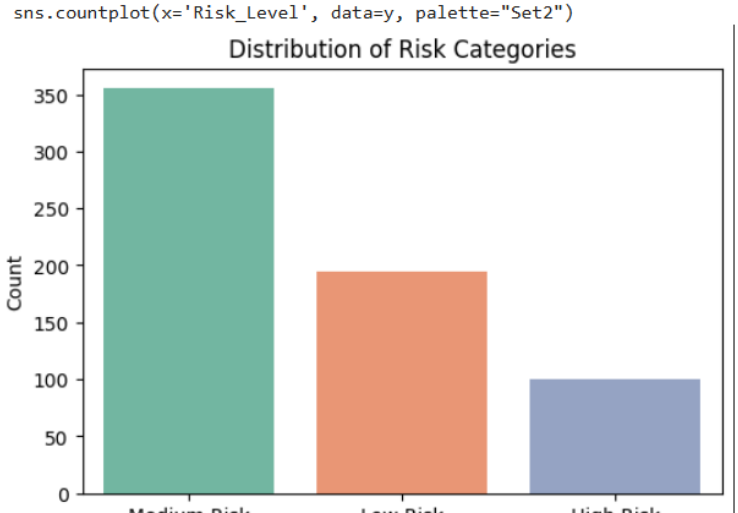
```
/tmp/ipython-input-2833699946.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assi

  sns.countplot(x="sex", data=X, palette="pastel")
```



- **Distribution of Risk Categories**

```
plt.figure(figsize=(6,4))
sns.countplot(x='Risk_Level', data=y, palette="Set2")
plt.title("Distribution of Risk Categories")
plt.xlabel("Risk Category")
plt.ylabel("Count")
plt.show()
```

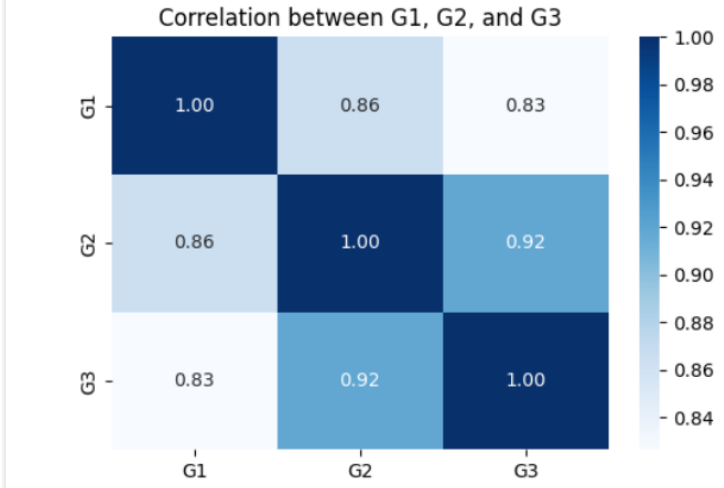
```
/tmp/ipython-input-4029544323.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.

  sns.countplot(x='Risk_Level', data=y, palette="Set2")
```



- **Heat map Correlation between G1, G2, and G3**

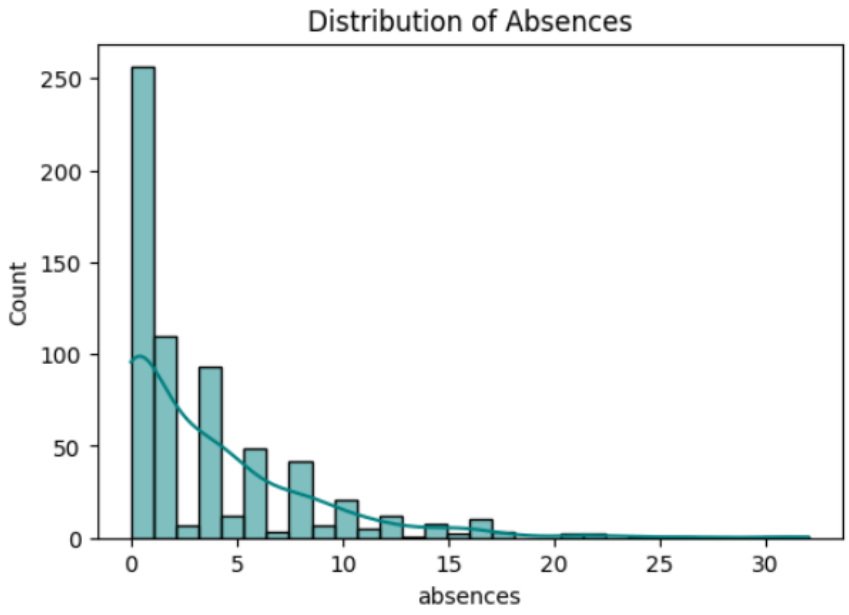
```
# Compute correlation matrix
corr_matrix = grades.corr()

# Display correlation values
print(corr_matrix)

# Heatmap visualization
plt.figure(figsize=(6,4))
sns.heatmap(corr_matrix, annot=True, cmap="Blues", fmt=".2f")
plt.title("Correlation between G1, G2, and G3")
plt.show()
```

```
          G1        G2        G3
G1  1.000000  0.864982  0.826387
G2  0.864982  1.000000  0.918548
G3  0.826387  0.918548  1.000000
```



- **Distribution of Absences**

```
plt.figure(figsize=(6,4))
sns.histplot(X["absences"], bins=30, kde=True, color="teal")
plt.title("Distribution of Absences")
plt.show()
```



4-Feature Engineering

```
# safest approach
X = X.copy()

# now add the feature
X.loc[:, "parent_edu_avg"] = (X["Medu"] + X["Fedu"]) / 2
```

```
X["attendance_ratio"] = 1 - (X["absences"] / 200)
```

```
X["support_index"] = X[["schoolsup", "famsup"]].apply(lambda x: sum(x == "yes"), axis=1)
```

```
X["leisure_score"] = X["freetime"] + X["goout"] - (X["Dalc"] + X["Walc"])
```

```
X["study_commitment"] = X["studytime"] - X["failures"]
```

```
X["high_absentee"] = (X["absences"] > 15).astype(int)
X["multiple_failures"] = (X["failures"] > 1).astype(int)
```

5-Splitting

```
from sklearn.model_selection import train_test_split

# Use Risk_Level as target
y_target = y["Risk_Level"]

# Train/test split (80/20) with stratification to keep class balance
X_train, X_test, y_train, y_test = train_test_split(
    X, y_target,
    test_size=0.2,
    random_state=42,
    stratify=y_target
)



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


print("Train set size:", X_train.shape[0])
print("Test set size:", X_test.shape[0])
print("Class distribution in train:")
print(y_train.value_counts(normalize=True))
print("\nClass distribution in test:")
print(y_test.value_counts(normalize=True))
```

```
Train set size: 519
Test set size: 130
Class distribution in train:
Risk_Level
Medium Risk    0.547206
Low Risk       0.298651
High Risk      0.154143
Name: proportion, dtype: float64

Class distribution in test:
Risk_Level
Medium Risk    0.546154
Low Risk       0.300000
High Risk      0.153846
```

- **6-Normalize numeric features (MinMaxScaler).**

```
from sklearn.preprocessing import MinMaxScaler

# Select numeric features
numeric_features = X_train.select_dtypes(include=["int64", "float64"]).columns

# Initialize scaler
scaler = MinMaxScaler()

# Fit on training data and transform both train & test
X_train[numeric_features] = scaler.fit_transform(X_train[numeric_features])
X_test[numeric_features] = scaler.transform(X_test[numeric_features])

print("Scaled numeric features:")
print(X_train[numeric_features].head())
```

```
Scaled numeric features:
          age  Medu  Fedu  traveltime  studytime  failures  famrel  freetime  \
318  0.428571  0.75  0.50    0.333333   0.333333       0.0    0.75      0.00
20   0.000000  1.00  0.75    0.000000   0.333333       0.0    0.75      0.75
190  0.142857  0.50  0.50    0.333333   0.333333       0.0    0.50      0.75
218  0.142857  0.50  0.50    0.333333   1.000000       0.0    1.00      0.50
286  0.285714  0.50  0.25    0.000000   0.000000       0.0    0.75      0.75

     goout  Dalc  Walc  health  absences        G1        G2  Average_grade  \
318   0.00  0.00  0.00     1.0       0.5  0.631579  0.473684        0.50000
20    0.00  0.00  0.00     0.0       0.0  0.631579  0.684211        0.62500
190   0.75  0.00  0.75     1.0       0.0  0.684211  0.631579        0.62500
218   1.00  0.00  0.00     1.0       0.2  0.684211  0.684211        0.65625
286   0.25  0.25  0.75     1.0       0.0  0.631579  0.631579        0.59375

     parent_edu_avg  attendance_ratio
318           0.625               0.5
20            0.875               1.0
190           0.500               1.0
218           0.500               0.8
286           0.375               1.0
```

- **Encode categorical data**

```
from sklearn.preprocessing import OneHotEncoder

# Select categorical features
categorical_features = X_train.select_dtypes(include=["object"]).columns

# Initialize OneHotEncoder
encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False)


# Loading... ining categorical data and transform both train & test
X_train_encoded = encoder.fit_transform(X_train[categorical_features])
X_test_encoded = encoder.transform(X_test[categorical_features])

# Convert encoded arrays back to DataFrames with proper column names
encoded_cols = encoder.get_feature_names_out(categorical_features)

X_train_encoded = pd.DataFrame(X_train_encoded, columns=encoded_cols, index=X_train.index)
X_test_encoded = pd.DataFrame(X_test_encoded, columns=encoded_cols, index=X_test.index)


X_train = pd.concat([X_train.drop(columns=categorical_features), X_train_encoded], axis=1)
X_test = pd.concat([X_test.drop(columns=categorical_features), X_test_encoded], axis=1)

print("Final Train shape:", X_train.shape)
print("Final Test shape:", X_test.shape)
```

```
Final Train shape: (519, 61)
Final Test shape: (130, 61)
```

7-Modeling

```python
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report

# ---------------------------
# 1. Initialize Models
# ---------------------------
models = {
    "Logistic Regression": LogisticRegression(multi_class="multinomial", solver="lbfgs", max_iter=1000, random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "SVM": SVC(kernel="rbf", probability=True, random_state=42)
}

# ---------------------------
# 2. Train & Evaluate Models
# ---------------------------
results = {}

for name, model in models.items():
    model.fit(X_train, y_train)           # Train
    y_pred = model.predict(X_test)        # Predict

    # Store evaluation metrics
    results[name] = {
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred, average="weighted"),
        "Recall": recall_score(y_test, y_pred, average="weighted"),
        "F1-score": f1_score(y_test, y_pred, average="weighted")
    }

    print(f"\n{name} Classification Report:")
    print(classification_report(y_test, y_pred))

# ---------------------------
# 3. Display Results
# ---------------------------
import pandas as pd

results_df = pd.DataFrame(results).T
display(results_df)
```

	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.761538	0.756706	0.761538	0.756901
Decision Tree	0.738462	0.742480	0.738462	0.738062
Random Forest	0.846154	0.848751	0.846154	0.841407
SVM	0.653846	0.665062	0.653846	0.616643

8-Grid search and saving the best model

```
[ ] from sklearn.model_selection import GridSearchCV

    param_grid = {
        "n_estimators": [100, 200, 500],
        "max_depth": [None, 10, 20],
        "min_samples_split": [2, 5, 10]
    }

    grid = GridSearchCV(RandomForestClassifier(class_weight="balanced", random_state=42),
                        param_grid, cv=5, scoring="f1_macro", n_jobs=-1)
    grid.fit(X_train, y_train)

    print("Best params:", grid.best_params_)
    print("Best score:", grid.best_score_)
```

```
Best params: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 100}
Best score: 0.8207321642141954
```

```
    from sklearn.metrics import accuracy_score

    y_pred = grid.best_estimator_.predict(X_test)
    test_acc = accuracy_score(y_test, y_pred)
    print("Test accuracy:", test_acc)
```

```
Test accuracy: 0.8461538461538461
```

```
[ ] from joblib import dump

    # Extract best model
    best_model = grid.best_estimator_
    print(best_model)

    # Save it
    dump(best_model, "best_rf_model.pkl")
```

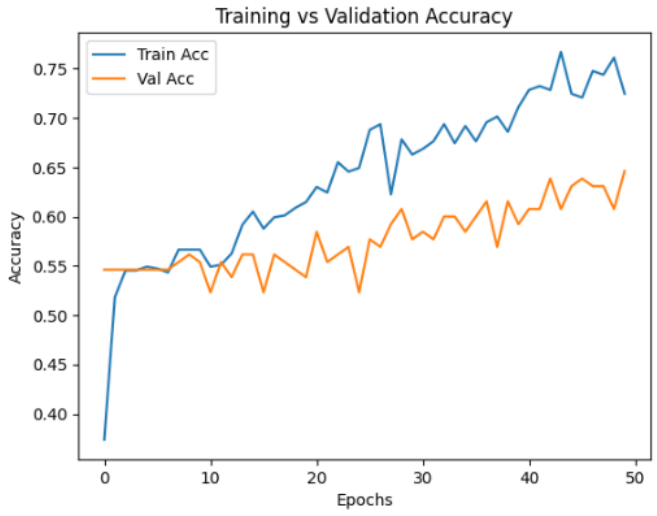
```
RandomForestClassifier(class_weight='balanced', min_samples_split=5,
                       random_state=42)
['best_rf_model.pkl']
```

9-Train a CNN model

```
# One-hot encode target for Keras
from tensorflow.keras.utils import to_categorical
y_train_nn = to_categorical(y_train_enc)
y_test_nn = to_categorical(y_test_enc)

# Build a simpler + regularized neural network
nn_model = keras.Sequential([
    layers.Input(shape=(X_train.shape[1],)),
    layers.Dense(128, activation="relu", kernel_regularizer=regularizers.l2(0.01)),
    layers.Dropout(0.4),

    layers.Dense(64, activation="relu", kernel_regularizer=regularizers.l2(0.01)),
    layers.Dropout(0.25),

    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.01)),
    layers.Dropout(0.2),
    layers.Dense(y_train_nn.shape[1], activation="softmax")  # output layer
])

# Compile model
nn_model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

# Callbacks
early_stop = keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=5, restore_best_weights=True
)

lr_scheduler = keras.callbacks.ReduceLROnPlateau(
    monitor="val_loss", factor=0.5, patience=3, verbose=1
)

# Train model
history = nn_model.fit(
    X_train, y_train_nn,
    validation_data=(X_test, y_test_nn),
    epochs=50,
    batch_size=32,
    callbacks=[early_stop, lr_scheduler],
    verbose=1
)

# Evaluate
loss, acc = nn_model.evaluate(X_test, y_test_nn, verbose=0)
print(f"Improved Neural Network Accuracy: {acc:.4f}")
```

- **Plot Training vs Validation Accuracy**

```
plt.plot(history.history["accuracy"], label="Train Acc")
plt.plot(history.history["val_accuracy"], label="Val Acc")
plt.title("Training vs Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



10-NLP

- **Simulate small dataset**

```
data = {
    "feedback": [
        "The course was amazing and I learned a lot!",
        "I struggled with the lectures, they were confusing.",
        "Assignments were helpful but too many.",
        "Great teacher, explained concepts clearly.",
        "I didn’t like the exams, very stressful.",
        "Loved the project work, very practical.",
        "The material was boring and hard to follow.",
        "Excellent explanations, I feel confident now."
    ],
    "performance": [1, 0, 0, 1, 0, 1, 0, 1]
}

df = pd.DataFrame(data)
print(df)
```

```
                                            feedback  performance
0        The course was amazing and I learned a lot!            1
1  I struggled with the lectures, they were confu...            0
2             Assignments were helpful but too many.            0
3         Great teacher, explained concepts clearly.            1
4           I didn’t like the exams, very stressful.            0
5            Loved the project work, very practical.            1
6        The material was boring and hard to follow.            0
7      Excellent explanations, I feel confident now.            1
```

- **Text processing**

```
import re
import nltk
from nltk.corpus import stopwords

nltk.download("stopwords")
stop_words = set(stopwords.words("english"))

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r"[^a-z\s]", "", text)
    words = [word for word in text.split() if word not in stop_words]
    return " ".join(words)

df["clean_feedback"] = df["feedback"].apply(preprocess_text)
print(df[["feedback", "clean_feedback"]])
```

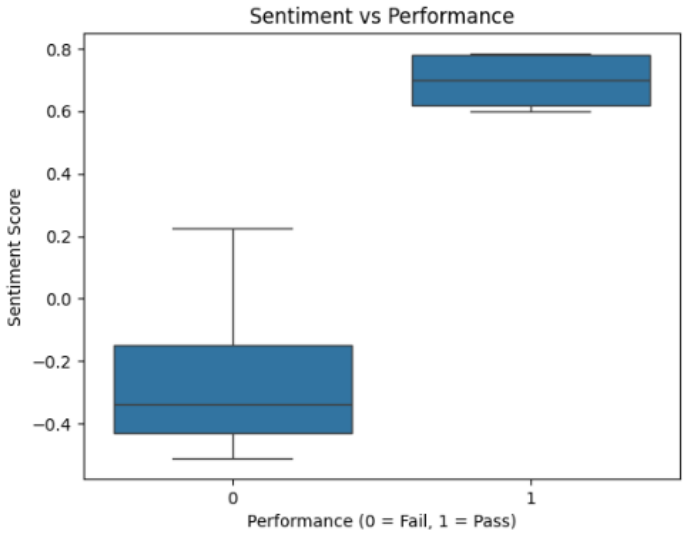
```
                                            feedback  \
0        The course was amazing and I learned a lot!
1  I struggled with the lectures, they were confu...
2             Assignments were helpful but too many.
3         Great teacher, explained concepts clearly.
4           I didn’t like the exams, very stressful.
5            Loved the project work, very practical.
6        The material was boring and hard to follow.
7      Excellent explanations, I feel confident now.

                          clean_feedback
0                course amazing learned lot
1              struggled lectures confusing
2                 assignments helpful many
3  great teacher explained concepts clearly
4                didnt like exams stressful
5              loved project work practical
6               material boring hard follow
7     excellent explanations feel confident
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

- **Sentiment Analysis**

```
from nltk.sentiment import SentimentIntensityAnalyzer
nltk.download("vader_lexicon")

sia = SentimentIntensityAnalyzer()

df["sentiment"] = df["feedback"].apply(lambda x: sia.polarity_scores(x)["compound"])
df["sentiment_label"] = df["sentiment"].apply(lambda x: "positive" if x > 0 else "negative")

print(df[["feedback", "sentiment", "sentiment_label", "performance"]])
```

Show hidden output

```
[ ] import seaborn as sns
    import matplotlib.pyplot as plt

    sns.boxplot(x=df["performance"], y=df["sentiment"])
    plt.xlabel("Performance (0 = Fail, 1 = Pass)")
    plt.ylabel("Sentiment Score")
    plt.title("Sentiment vs Performance")
    plt.show()
```



11-Deploying ML Model