

Amro Ismail Mohamed - studentsProject

April 21, 2026

Dr. Hager Saleh

Name: Amro Ismail Mohamed

https://www.kaggle.com/datasets/mubashirsidiki/student-academic-performance-500-students/data?select=student_performance.csv

1 Import the Libraries

```
[129]: import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.feature_selection import SelectKBest, chi2, RFE
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, \
    precision_score, recall_score, confusion_matrix
```

2 Load the dataset

```
[130]: df = pd.read_csv('student_performance.csv')
df.head()
```

```
[130]:
```

	student_id	gender	age	study_hours_per_week	attendance_rate	\
0	STU0001	Male	15	25	63.8	
1	STU0002	Female	15	2	54.7	
2	STU0003	Female	19	10	90.5	
3	STU0004	Male	16	26	66.8	
4	STU0005	Female	15	25	73.0	

	parent_education	internet_access	extracurricular	previous_score	\
0	Bachelor	Yes	Yes	41	
1	Bachelor	Yes	Yes	83	
2	High School	Yes	No	73	
3	High School	No	Yes	75	
4	High School	No	Yes	67	

	final_score	passed
0	67	Yes
1	28	No
2	49	No
3	70	Yes
4	77	Yes

3 Handle missing values appropriately

```
[131]: # Checking for missing values
df.isnull().sum()
```

```
[131]: student_id      0
gender              0
age                0
study_hours_per_week  0
attendance_rate     0
parent_education    117
internet_access     0
extracurricular     0
previous_score      0
final_score         0
passed             0
dtype: int64
```

```
[132]: df.dtypes
```

```
[132]: student_id      object
gender              object
age                int64
study_hours_per_week  int64
attendance_rate     float64
parent_education    object
internet_access     object
extracurricular     object
previous_score      int64
final_score         int64
passed             object
```

dtype: object

```
[133]: # Fill categorical missing values with mode
for col in df.select_dtypes(include=['object']).columns:
    df[col].fillna(df[col].mode()[0], inplace=True)
```

```
[134]: # Fill numerical missing values with mean
for col in df.select_dtypes(include=['float64', 'int64']).columns:
    df[col].fillna(df[col].mean(), inplace=True)
```

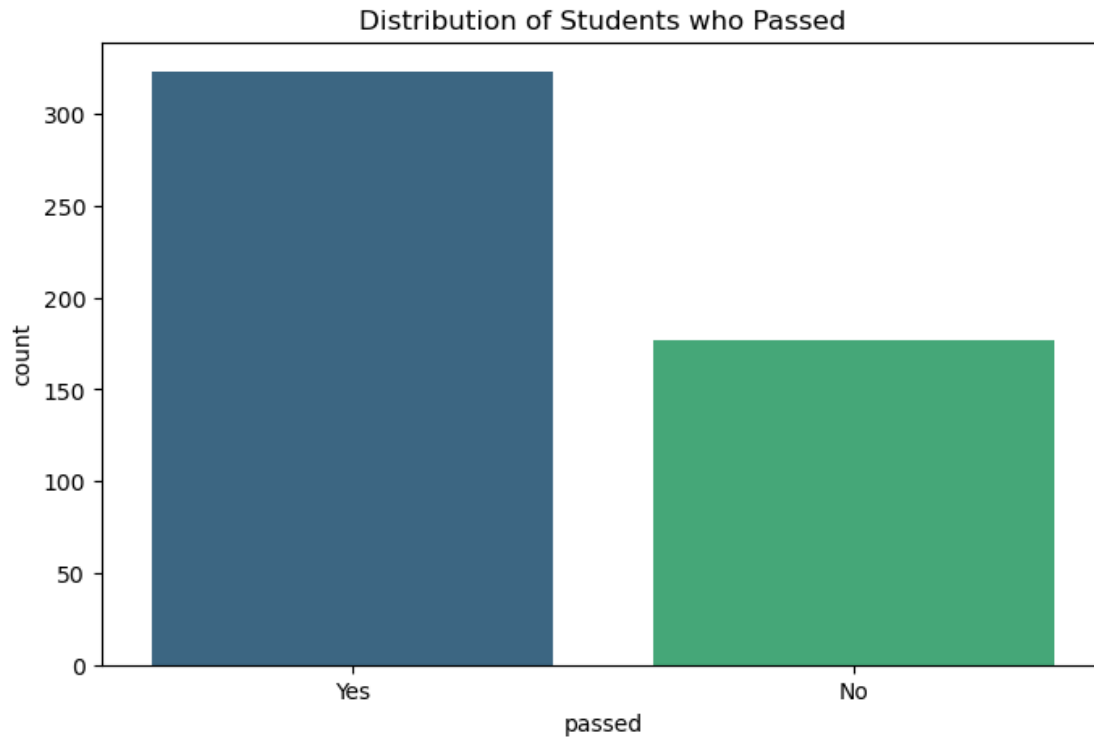
```
[135]: # Drop student_id if exists
if 'student_id' in df.columns:
    df.drop(['student_id'], axis=1, inplace=True)

df.dtypes
```

```
[135]: gender                object
age                    int64
study_hours_per_week  int64
attendance_rate        float64
parent_education       object
internet_access         object
extracurricular         object
previous_score          int64
final_score             int64
passed                  object
dtype: object
```

4 Data Visualization

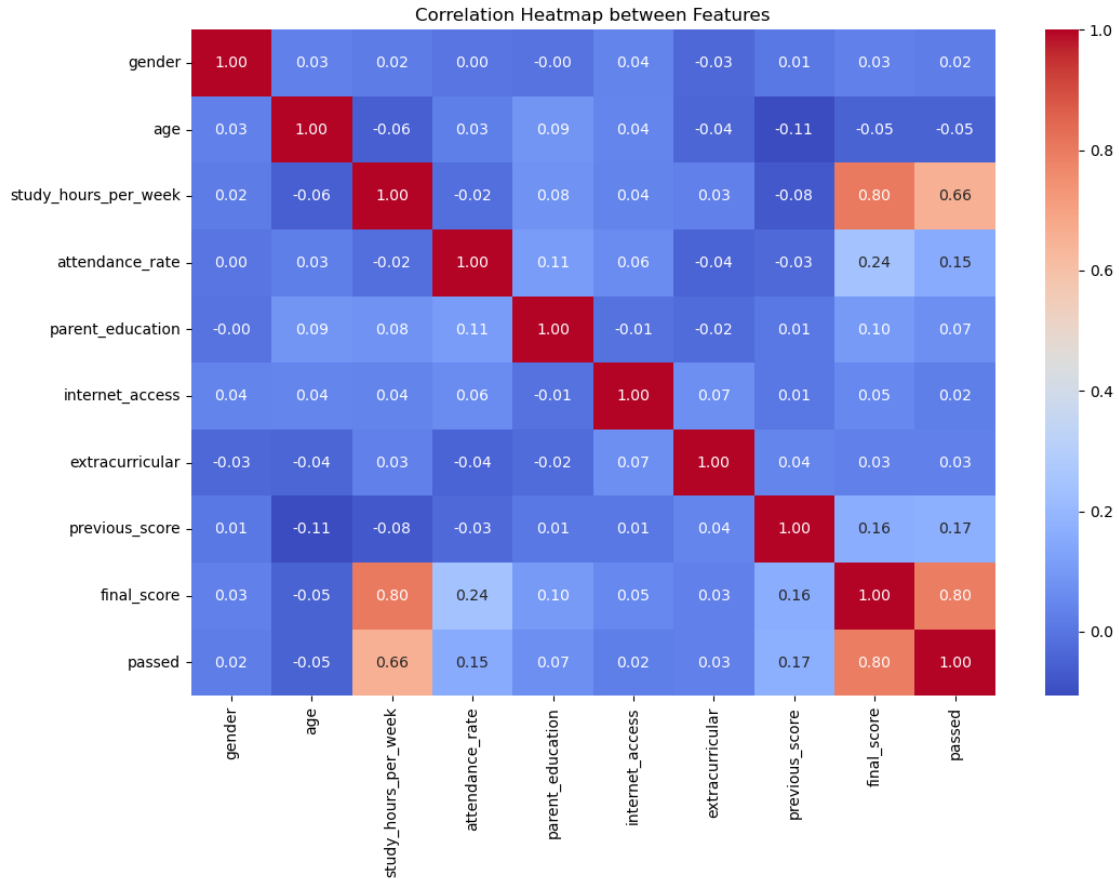
```
[136]: # Chart A: Categorical Objects Distribution (Passed status)
plt.figure(figsize=(8,5))
sns.countplot(x='passed', data=df, palette='viridis')
plt.title('Distribution of Students who Passed')
plt.show()
```



```
[137]: # Chart B: Correlation Heatmap
# We must encode first to show correlation for categorical objects
temp_df = df.copy()

le_temp = LabelEncoder()
for col in temp_df.select_dtypes(include=['object']).columns:
    temp_df[col] = le_temp.fit_transform(temp_df[col])

plt.figure(figsize=(12,8))
sns.heatmap(temp_df.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap between Features')
plt.show()
```



5 Preprocessing: Encoding and Scaling

```
[138]: # Remove final_score to prevent data leakage
df.drop(['final_score'], axis=1, inplace=True)

# Define Features and Target
X = df.drop(['passed'], axis=1)
y = df['passed']

# Split FIRST (important)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=42, stratify=y
)
```

```
[139]: label_encoders = {}

for column in X_train.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
```

```

X_train[column] = le.fit_transform(X_train[column])
X_test[column] = le.transform(X_test[column])
label_encoders[column] = le

# Encode target
target_encoder = LabelEncoder()
y_train = target_encoder.fit_transform(y_train)
y_test = target_encoder.transform(y_test)

X_train

```

```

[139]:
gender  age  study_hours_per_week  attendance_rate  parent_education  \
107      0   15                    29             92.3             1
454      1   15                    16             72.2             2
124      1   18                     2             81.0             2
357      0   15                     3             96.1             2
471      0   15                    14             69.4             2
..      ...  ...
236      1   17                    11             72.7             2
429      0   16                    26             88.7             2
310      0   19                    14             79.4             2
343      1   19                     8             98.7             0
443      1   17                    22             61.6             2

internet_access  extracurricular  previous_score
107              0                1              48
454              0                1              93
124              0                1              38
357              0                0              88
471              1                1              62
..              ...
236              0                1              94
429              1                0              41
310              0                0              52
343              0                0              38
443              0                0              36

[350 rows x 8 columns]

```

6 Apply Machine Learning with Grid Search Optimizer

```

[140]: # Optimizing Random Forest as an example
rf = RandomForestClassifier(
    random_state=42,
    class_weight='balanced'
)

```

```

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    cv=5,
    scoring='f1_weighted',
    n_jobs=-1
)

grid_search.fit(X_train, y_train)

best_rf = grid_search.best_estimator_
y_pred_rf = best_rf.predict(X_test)

print("Best Parameters found by Grid Search:", grid_search.best_params_)

```

Best Parameters found by Grid Search: {'max_depth': None, 'min_samples_split': 10, 'n_estimators': 200}

```

[141]: # Model Evaluation
print("\nAccuracy:", accuracy_score(y_test, y_pred_rf))

print("\nClassification Report:\n")
print(classification_report(y_test, y_pred_rf))

print("\nConfusion Matrix:\n")
print(confusion_matrix(y_test, y_pred_rf))

```

Accuracy: 0.8133333333333334

Classification Report:

	precision	recall	f1-score	support
0	0.70	0.81	0.75	53
1	0.89	0.81	0.85	97
accuracy			0.81	150
macro avg	0.80	0.81	0.80	150
weighted avg	0.82	0.81	0.82	150

Confusion Matrix:

```
[[43 10]
 [18 79]]
```

7 Feature Selection Methods

```
[142]: # Method 1: Chi-Square (Filter Method)
# Note: Chi-Square requires non-negative features. Ensure data is encoded/
#       ↪ scaled properly.
chi2_selector = SelectKBest(score_func=chi2, k=5)

X_train_chi2 = chi2_selector.fit_transform(X_train, y_train)
X_test_chi2 = chi2_selector.transform(X_test)

chi2_features = X_train.columns[chi2_selector.get_support()]

print("Top 5 Features selected by Chi-Square:", list(chi2_features))
```

Top 5 Features selected by Chi-Square: ['gender', 'study_hours_per_week', 'attendance_rate', 'extracurricular', 'previous_score']

```
[143]: # Method 2: Recursive Feature Elimination (RFE) (Wrapper Method)
lr_model = LogisticRegression(max_iter=1000)

rfe_selector = RFE(estimator=lr_model, n_features_to_select=5)

X_train_rfe = rfe_selector.fit_transform(X_train, y_train)
X_test_rfe = rfe_selector.transform(X_test)

rfe_features = X_train.columns[rfe_selector.get_support()]

print("Top 5 Features selected by RFE:", list(rfe_features))
```

Top 5 Features selected by RFE: ['age', 'study_hours_per_week', 'attendance_rate', 'internet_access', 'extracurricular']

8 Apply Models to Selected Features and Evaluate

```
[144]: def evaluate_on_features(X_tr, X_te, y_tr, y_te, method_name):

    models = {
        "Random Forest": RandomForestClassifier(
            random_state=42,
            class_weight='balanced'
        ),
```

```

"Decision Tree": DecisionTreeClassifier(
    random_state=42,
    class_weight='balanced'
),
"SVM": SVC(kernel='rbf', random_state=42)
}

results = []

print(f"\n--- Performance using {method_name} ---")

for name, model in models.items():
    model.fit(X_tr, y_tr)
    preds = model.predict(X_te)

    results.append({
        "Method": method_name,
        "Model": name,
        "Accuracy": accuracy_score(y_te, preds),
        "Precision": precision_score(y_te, preds, average='weighted'),
        "Recall": recall_score(y_te, preds, average='weighted')
    })

return results

```

```

[145]: # Evaluating results for both feature selection methods
results_chi2 = evaluate_on_features(
    X_train_chi2, X_test_chi2, y_train, y_test, "Chi-Square"
)

results_rfe = evaluate_on_features(
    X_train_rfe, X_test_rfe, y_train, y_test, "RFE"
)

```

--- Performance using Chi-Square ---

--- Performance using RFE ---

```

[146]: # Display final comparison table [26-28]
final_results = pd.DataFrame(results_chi2 + results_rfe)

print("\nFinal Comparison of Model Performance on Selected Features:")

final_results

```

Final Comparison of Model Performance on Selected Features:

[146]:

	Method	Model	Accuracy	Precision	Recall
0	Chi-Square	Random Forest	0.813333	0.815145	0.813333
1	Chi-Square	Decision Tree	0.780000	0.777600	0.780000
2	Chi-Square	SVM	0.800000	0.800000	0.800000
3	RFE	Random Forest	0.833333	0.838331	0.833333
4	RFE	Decision Tree	0.766667	0.772726	0.766667
5	RFE	SVM	0.826667	0.830530	0.826667

The best performing model : Random Forest with RFE feature selection achieving 83.33% accuracy.