

```
In [27]: import warnings
warnings.filterwarnings("ignore")
# =====
# Libraries
# =====
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, classification_report

from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

from sklearn.feature_selection import SelectKBest, chi2, RFE
from sklearn.linear_model import LogisticRegression
```

```
In [28]: # =====
# 2. Load Dataset
# https://www.kaggle.com/datasets/guriya79/how-ai-is-changing-student-life/code
# =====
df = pd.read_csv("AI_Student_Life_Pakistan_2026.csv")

print("Shape:", df.shape)
print("\nColumns:\n", df.columns)
```

Shape: (100, 10)

Columns:

```
Index(['Student_ID', 'Age', 'Gender', 'Education_Level', 'City',
      'AI_Tool_Used', 'Daily_Usage_Hours', 'Purpose', 'Impact_on_Grades',
      'Satisfaction_Level'],
      dtype='str')
```

```
In [29]: # =====
# 3. Data Cleaning
# =====

# Drop ID column
if 'Student_ID' in df.columns:
    df.drop(['Student_ID'], axis=1, inplace=True)

# Handle Missing Values
print("\nMissing Values Before:\n", df.isnull().sum())

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

# Fill numeric with mean
for col in df.select_dtypes(include=np.number).columns:
    df[col].fillna(df[col].mean(), inplace=True)

print("\nMissing Values After:\n", df.isnull().sum())
```

Missing Values Before:

Age	0
Gender	0
Education_Level	0
City	0
AI_Tool_Used	0
Daily_Usage_Hours	0
Purpose	0
Impact_on_Grades	0
Satisfaction_Level	0

dtype: int64

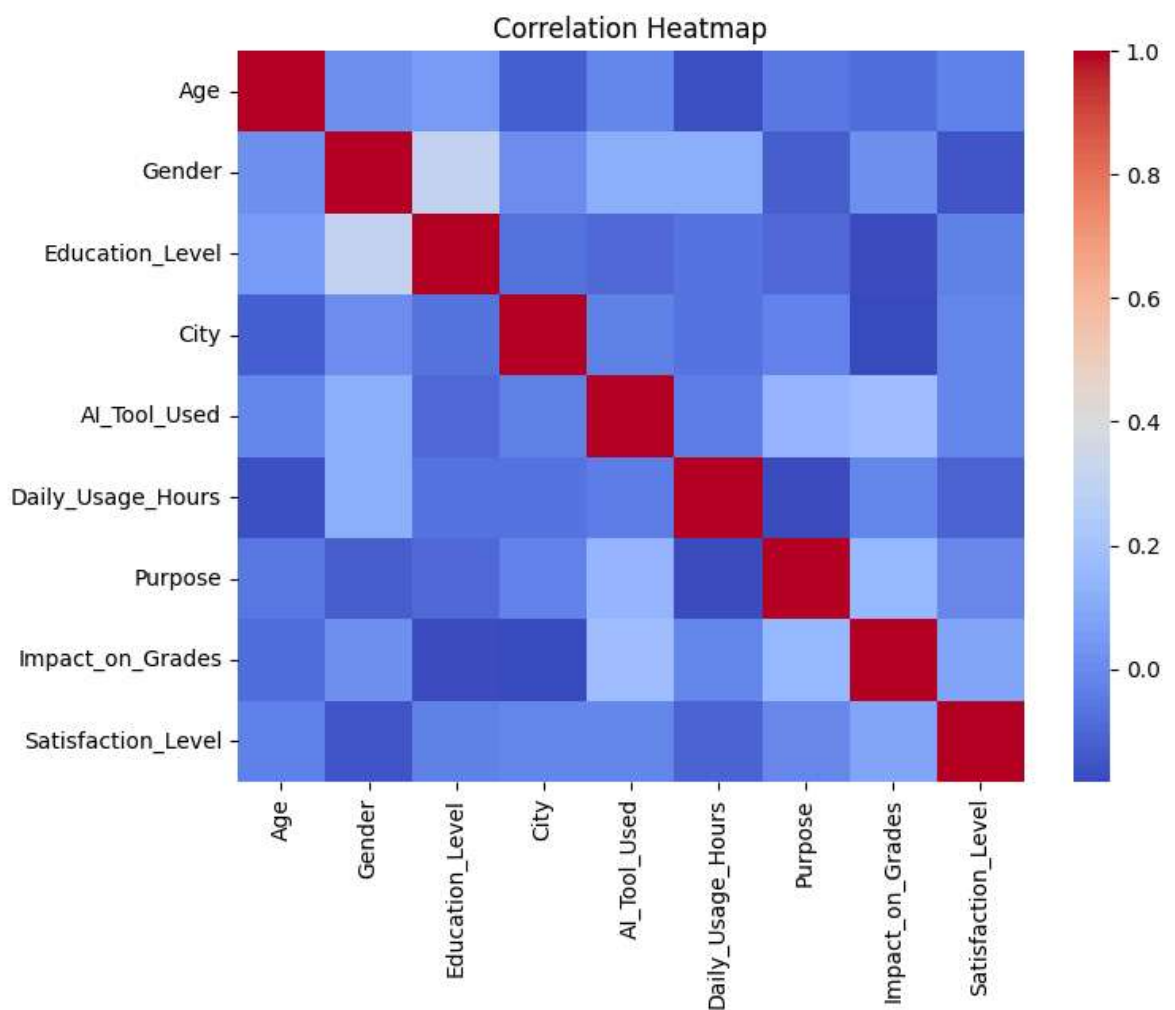
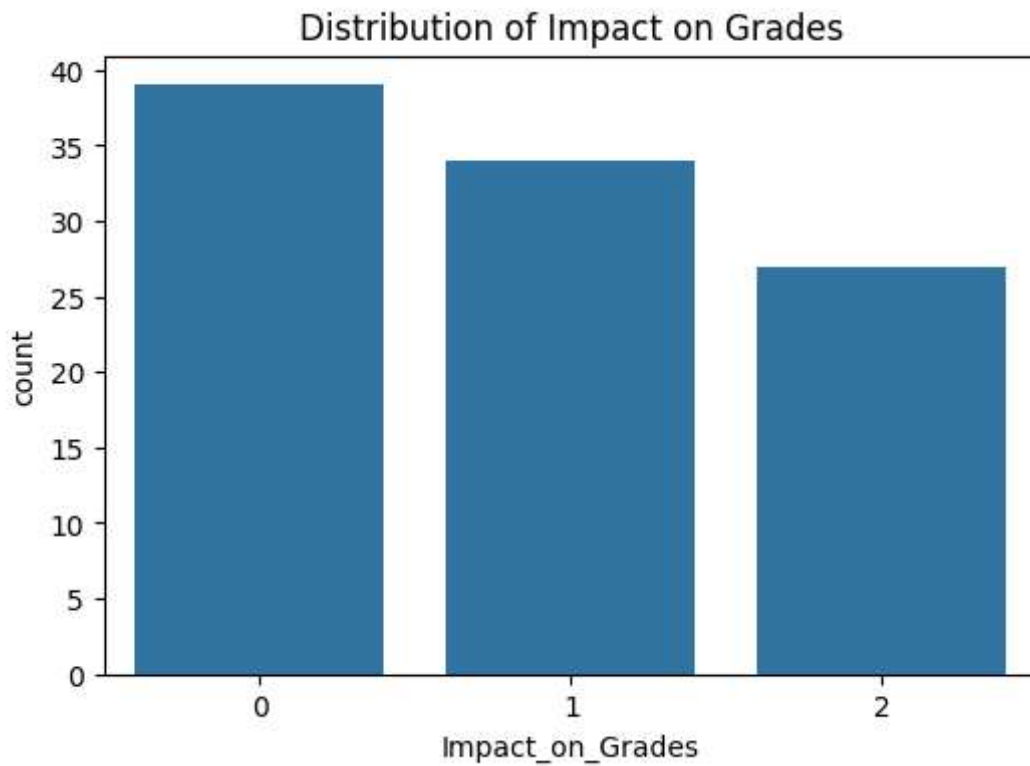
Missing Values After:

Age	0
Gender	0
Education_Level	0
City	0
AI_Tool_Used	0
Daily_Usage_Hours	0
Purpose	0
Impact_on_Grades	0
Satisfaction_Level	0

dtype: int64

```
In [30]: # =====  
# 4. Encoding  
# =====  
le = LabelEncoder()  
  
for col in df.select_dtypes(include='object').columns:  
    df[col] = le.fit_transform(df[col])
```

```
In [31]: # =====  
# 5. Visualization  
# =====  
plt.figure(figsize=(6,4))  
sns.countplot(x='Impact_on_Grades', data=df)  
plt.title("Distribution of Impact on Grades")  
plt.show()  
  
plt.figure(figsize=(8,6))  
sns.heatmap(df.corr(), cmap='coolwarm')  
plt.title("Correlation Heatmap")  
plt.show()
```



```
In [32]: # =====
# 6. Define Features & Target
# =====
X = df.drop('Impact_on_Grades', axis=1)
y = df['Impact_on_Grades']
```

```
In [33]: # =====
# 7. Train/Test Split
# =====
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```
In [34]: # =====
# 8. Models + Grid Search
# =====

models = {
    "Random Forest": {
        "model": RandomForestClassifier(),
        "params": {
            "n_estimators": [100, 200],
            "max_depth": [None, 10]
        }
    },
    "Decision Tree": {
        "model": DecisionTreeClassifier(),
        "params": {
            "max_depth": [None, 10, 20]
        }
    },
    "SVM": {
        "model": SVC(),
        "params": {
            "C": [0.1, 1],
            "kernel": ['rbf', 'linear']
        }
    }
}

results = []

print("\n===== MODEL TRAINING WITH GRID SEARCH =====")

for name, config in models.items():
    print(f"\n--- {name} ---")

    grid = GridSearchCV(config["model"], config["params"], cv=5)
    grid.fit(X_train, y_train)

    best_model = grid.best_estimator_
    y_pred = best_model.predict(X_test)

    print("Best Params:", grid.best_params_)
    print(classification_report(y_test, y_pred))

    results.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred)
    })

results_df = pd.DataFrame(results)
print("\nModel Comparison:\n", results_df)
```

===== MODEL TRAINING WITH GRID SEARCH =====

--- Random Forest ---

Best Params: {'max_depth': None, 'n_estimators': 100}

	precision	recall	f1-score	support
0	0.57	0.50	0.53	8
1	0.50	0.50	0.50	6
2	0.57	0.67	0.62	6
accuracy			0.55	20
macro avg	0.55	0.56	0.55	20
weighted avg	0.55	0.55	0.55	20

--- Decision Tree ---

Best Params: {'max_depth': 20}

	precision	recall	f1-score	support
0	0.50	0.62	0.56	8
1	0.20	0.17	0.18	6
2	0.40	0.33	0.36	6
accuracy			0.40	20
macro avg	0.37	0.38	0.37	20
weighted avg	0.38	0.40	0.39	20

--- SVM ---

Best Params: {'C': 0.1, 'kernel': 'linear'}

	precision	recall	f1-score	support
0	0.60	0.75	0.67	8
1	0.44	0.67	0.53	6
2	1.00	0.17	0.29	6
accuracy			0.55	20
macro avg	0.68	0.53	0.50	20
weighted avg	0.67	0.55	0.51	20

Model Comparison:

	Model	Accuracy
0	Random Forest	0.55
1	Decision Tree	0.40
2	SVM	0.55

```
In [35]: # =====
# 9. Feature Selection (Chi-Square)
# =====
selector = SelectKBest(score_func=chi2, k=5)
X_new = selector.fit_transform(X, y)

selected_features_chi = X.columns[selector.get_support()]
print("\nSelected Features (Chi2):", list(selected_features_chi))
```

Selected Features (Chi2): ['Education_Level', 'City', 'AI_Tool_Used', 'Daily_Usage_Hours', 'Purpose']

```
In [36]: # =====
# 10. Feature Selection (RFE)
# =====
model = LogisticRegression(max_iter=500)
rfe = RFE(model, n_features_to_select=5)
rfe.fit(X, y)

selected_features_rfe = X.columns[rfe.support_]
print("\nSelected Features (RFE):", list(selected_features_rfe))
```

Selected Features (RFE): ['Gender', 'Education_Level', 'City', 'Daily_Usage_Hours', 'Satisfaction_Level']

```
In [37]: # =====
# 11. Train Again using Selected Features
# =====
print("\n==== TRAINING AFTER FEATURE SELECTION =====")

X_selected = X[selected_features_chi]

X_train, X_test, y_train, y_test = train_test_split(
    X_selected, y, test_size=0.2, random_state=42
)

model = RandomForestClassifier()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("\nAfter Feature Selection:")
print(classification_report(y_test, y_pred))
```

==== TRAINING AFTER FEATURE SELECTION =====

After Feature Selection:

	precision	recall	f1-score	support
0	0.57	0.50	0.53	8
1	0.33	0.17	0.22	6
2	0.40	0.67	0.50	6
accuracy			0.45	20
macro avg	0.43	0.44	0.42	20
weighted avg	0.45	0.45	0.43	20

```
In [38]: # =====
# 12. Confusion Matrix Plot
# =====
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

