```
# Step 1: Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
```

```
# Step 2: Load dataset
from google.colab import files
uploaded = files.upload()  # Upload StudentsPerformance.csv

df = pd.read_csv('StudentsPerformance.csv')
df.head()
```

Choose Files StudentsPerformance.csv

- **StudentsPerformance.csv**(text/csv) - 72036 bytes, last modified: 6/2/2025 - 100% done

```
Saving StudentsPerformance.csv to StudentsPerformance.csv
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Step 3: Encode categorical variables
label_encoders = {}
for column in df.select_dtypes(include='object').columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le
```

```
# Step 4: Create target variable (pass = 1 if avg score ≥ 60, else 0)
df['average_score'] = df[['math score', 'reading score', 'writing score']].mean(axis=1)
df['pass'] = (df['average_score'] >= 60).astype(int)
df = df.drop(columns=['average_score'])
```

```
# Step 5: Split data
X = df.drop('pass', axis=1)
y = df['pass']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Step 6: Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Step 7: Train Logistic Regression
lr_model = LogisticRegression(max_iter=1000, random_state=42)
lr_model.fit(X_train_scaled, y_train)
y_pred_lr = lr_model.predict(X_test_scaled)
```

```
# Step 8: Train k-NN
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train_scaled, y_train)
y_pred_knn = knn_model.predict(X_test_scaled)
```

```
# Step 9: Train Decision Tree
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train_scaled, y_train)
y_pred_dt = dt_model.predict(X_test_scaled)


# Step 10: Evaluation
print("📊 Logistic Regression:\n", classification_report(y_test, y_pred_lr))
print("📊 k-NN:\n", classification_report(y_test, y_pred_knn))
print("📊 Decision Tree:\n", classification_report(y_test, y_pred_dt))
```

```
📊 Logistic Regression:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        62
           1       0.99      1.00      1.00       138

    accuracy                           0.99       200
   macro avg       1.00      0.99      0.99       200
weighted avg       1.00      0.99      0.99       200

📊 k-NN:
              precision    recall  f1-score   support

           0       0.95      0.84      0.89        62
           1       0.93      0.98      0.95       138

    accuracy                           0.94       200
   macro avg       0.94      0.91      0.92       200
weighted avg       0.94      0.94      0.93       200

📊 Decision Tree:
              precision    recall  f1-score   support

           0       0.94      0.95      0.94        62
           1       0.98      0.97      0.97       138

    accuracy                           0.96       200
   macro avg       0.96      0.96      0.96       200
weighted avg       0.97      0.96      0.97       200
```