Upload the dataset

```
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
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score

# Step 1: Load the dataset
df = pd.read_csv("student-dataset (1).csv", delimiter=';')

# Step 2: Define "churn" — students with G3 < 10 are considered 'churned'
df['churn'] = (df['G3'] < 10).astype(int)

# Step 3: Drop original grade columns (G1, G2, G3) to prevent data leakage
df = df.drop(columns=['G1', 'G2', 'G3'])

# Step 4: Encode categorical variables
cat_cols = df.select_dtypes(include='object').columns
df_encoded = pd.get_dummies(df, columns=cat_cols, drop_first=True)

# Step 5: Split data into train/test sets
X = df_encoded.drop('churn', axis=1)
y = df_encoded['churn']
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 models
# Logistic Regression
lr = LogisticRegression()
lr.fit(X_train_scaled, y_train)

# Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Step 8: Evaluation
def evaluate_model(model, X_test, y_test, name="Model"):
    y_pred = model.predict(X_test)
    print(f"\n{name} Evaluation:")
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
    print(f"AUC: {roc_auc_score(y_test, model.predict_proba(X_test)[:,1]):.2f}")

evaluate_model(lr, X_test_scaled, y_test, "Logistic Regression")
evaluate_model(rf, X_test, y_test, "Random Forest")

# Step 9: Feature Importance (Random Forest)
importances = pd.Series(rf.feature_importances_, index=X.columns)
top_features = importances.sort_values(ascending=False).head(10)

plt.figure(figsize=(10,6))
sns.barplot(x=top_features, y=top_features.index)
plt.title("Top 10 Important Features for Predicting Churn")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```

ıon-input-24-72ee7f69330e>", line 21 'age' Module •••

Please explain the error:

```
ModuleNotFoundError: No module named 'streamlit
```

Hypothesis

The error `ModuleNotFoundError: No module named 'streamlit'` indicates that the `streamlit` package is not installed in the current Python environment. Before you can use `import streamlit as st`, you need to install it.

Suggested Changes

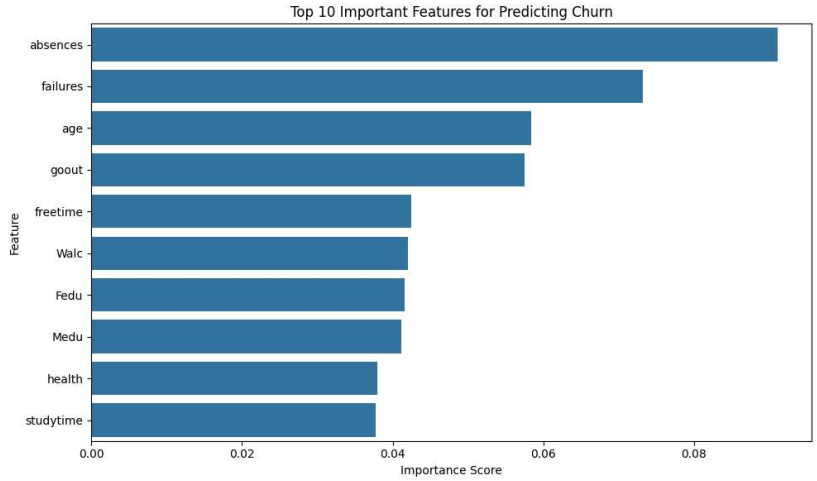
```
!pip install streamlit
```

Use code with caution

▶ Sources

Rate this answer

```
Logistic Regression Evaluation:
[[45  7]
 [16 11]]
              precision    recall  f1-score   support

           0       0.74      0.87      0.80        52
           1       0.61      0.41      0.49        27

    accuracy                           0.71        79
   macro avg       0.67      0.64      0.64        79
weighted avg       0.69      0.71      0.69        79

AUC: 0.74

Random Forest Evaluation:
[[48  4]
 [22  5]]
              precision    recall  f1-score   support

           0       0.69      0.92      0.79        52
           1       0.56      0.19      0.28        27

    accuracy                           0.67        79
   macro avg       0.62      0.55      0.53        79
weighted avg       0.64      0.67      0.61        79

AUC: 0.73
```



Load the dataset

```
import pandas as pd

# Load the dataset using the correct delimiter
file_path = "student-dataset (1).csv"
df = pd.read_csv(file_path, delimiter=';')

# Display the first 5 rows
print(df.head())

# Show basic info about the dataset
print("\nDataset Info:")
print(df.info())
```

```
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  ... \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher  ...
1     GP   F   17       U     GT3       T     1     1  at_home     other  ...
2     GP   F   15       U     LE3       T     1     1  at_home     other  ...
3     GP   F   15       U     GT3       T     4     2   health  services  ...
4     GP   F   16       U     GT3       T     3     3    other     other  ...
```

```
  famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
0      4        3      4     1     1      3        6   5   6   6
1      5        3      3     1     1      3        4   5   5   6
2      4        3      2     2     3      3       10   7   8  10
3      3        2      2     1     1      5        2  15  14  15
4      4        3      2     1     2      5        4   6  10  10

[5 rows x 33 columns]

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 395 entries, 0 to 394
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   school      395 non-null    object
 1   sex         395 non-null    object
 2   age         395 non-null    int64
 3   address     395 non-null    object
 4   famsize     395 non-null    object
 5   Pstatus     395 non-null    object
 6   Medu        395 non-null    int64
 7   Fedu        395 non-null    int64
 8   Mjob        395 non-null    object
 9   Fjob        395 non-null    object
 10  reason      395 non-null    object
 11  guardian    395 non-null    object
 12  traveltime  395 non-null    int64
 13  studytime   395 non-null    int64
 14  failures    395 non-null    int64
 15  schoolsup   395 non-null    object
 16  famsup      395 non-null    object
 17  paid        395 non-null    object
 18  activities  395 non-null    object
 19  nursery     395 non-null    object
 20  higher      395 non-null    object
 21  internet    395 non-null    object
 22  romantic    395 non-null    object
 23  famrel      395 non-null    int64
 24  freetime    395 non-null    int64
 25  goout       395 non-null    int64
 26  Dalc        395 non-null    int64
 27  Walc        395 non-null    int64
 28  health      395 non-null    int64
 29  absences    395 non-null    int64
 30  G1          395 non-null    int64
 31  G2          395 non-null    int64
 32  G3          395 non-null    int64
dtypes: int64(16), object(17)
memory usage: 102.0+ KB
```

Data Exploration

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
file_path = "student-dataset (1).csv"
df = pd.read_csv(file_path, delimiter=';')

# 1. Basic overview
print(" 🔹 First 5 Rows:\n", df.head())
print("\n 🔹 Dataset Shape:", df.shape)
print("\n 🔹 Column Names:\n", df.columns.tolist())
print("\n 🔹 Data Types:\n", df.dtypes)
print("\n 🔹 Missing Values:\n", df.isnull().sum())
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
file_path = "student-dataset (1).csv"
df = pd.read_csv(file_path, delimiter=';')

# 1. Basic overview
print(" 🔹 First 5 Rows:\n", df.head())
print("\n 🔹 Dataset Shape:", df.shape)
```

```
print("\n 🔹 Column Names:\n", df.columns.tolist())
print("\n 🔹 Data Types:\n", df.dtypes)
print("\n 🔹 Missing Values:\n", df.isnull().sum())

# 2. Statistical summary of numeric features
print("\n 🔹 Statistical Summary of Numeric Features:\n", df.describe())
```

```
Mjob          0
Fjob          0
reason        0
guardian      0
traveltime    0
studytime     0
failures      0
schoolsup     0
famsup        0
paid          0
activities    0
nursery       0
higher        0
internet      0
romantic      0
famrel        0
freetime      0
goout         0
Dalc          0
Walc          0
health        0
absences      0
G1            0
G2            0
G3            0
dtype: int64

 🔹 Statistical Summary of Numeric Features:
              age        Medu        Fedu  traveltime   studytime    failures
count  395.000000  395.000000  395.000000  395.000000  395.000000  395.000000
mean    16.696203    2.749367    2.521519    1.448101    2.035443    0.334177
std      1.276043    1.094735    1.088201    0.697505    0.839240    0.743651
min     15.000000    0.000000    0.000000    1.000000    1.000000    0.000000
25%     16.000000    2.000000    2.000000    1.000000    1.000000    0.000000
50%     17.000000    3.000000    2.000000    1.000000    2.000000    0.000000
75%     18.000000    4.000000    3.000000    2.000000    2.000000    0.000000
max     22.000000    4.000000    4.000000    4.000000    4.000000    3.000000

           famrel    freetime       goout        Dalc        Walc      health  \
count  395.000000  395.000000  395.000000  395.000000  395.000000  395.000000
mean     3.944304    3.235443    3.108861    1.481013    2.291139    3.554430
std      0.896659    0.998862    1.113278    0.890741    1.287897    1.390303
min      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000
25%      4.000000    3.000000    2.000000    1.000000    1.000000    3.000000
50%      4.000000    3.000000    3.000000    1.000000    2.000000    4.000000
75%      5.000000    4.000000    4.000000    2.000000    3.000000    5.000000
max      5.000000    5.000000    5.000000    5.000000    5.000000    5.000000

         absences          G1          G2          G3
count  395.000000  395.000000  395.000000  395.000000
mean     5.708861   10.908861   10.713924   10.415190
std      8.003096    3.319195    3.761505    4.581443
min      0.000000    3.000000    0.000000    0.000000
25%      0.000000    8.000000    9.000000    8.000000
50%      4.000000   11.000000   11.000000   11.000000
75%      8.000000   13.000000   13.000000   14.000000
max     75.000000   19.000000   19.000000   20.000000
```

Check for Missing Values and Duplicates

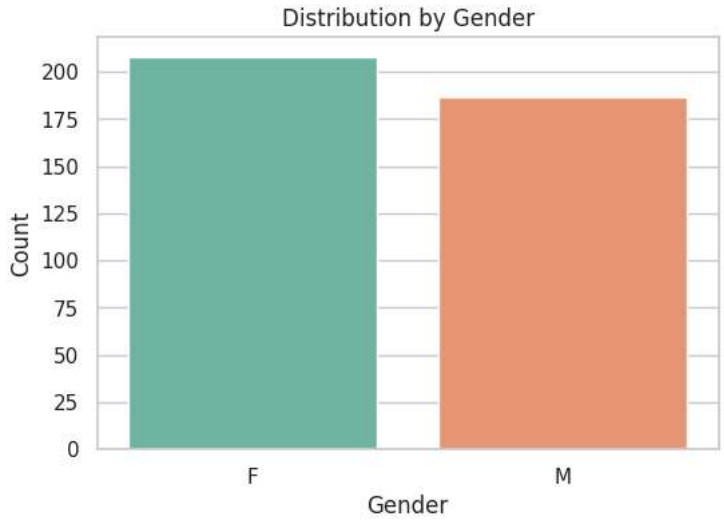
```
import pandas as pd

# Load the dataset with correct delimiter
file_path = "student-dataset (1).csv"
df = pd.read_csv(file_path, delimiter=';')

# 1. Check for missing values
print("🔍 Missing Values in Each Column:")
missing_values = df.isnull().sum()
print(missing_values[missing_values > 0] if missing_values.sum() > 0 else "No missing

# 2. Check for duplicate rows
print("🔍 Duplicate Rows:", df.duplicated().sum()) # Changed 'pr' to 'print' and added
```

```
🔍 Missing Values in Each Column:
No missing values found.
🔍 Duplicate Rows: 0
```

Visualize a Few Features

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = "student-dataset (1).csv"
df = pd.read_csv(file_path, delimiter=';')

# Set seaborn style
sns.set(style="whitegrid")

# 1. Gender Distribution
plt.figure(figsize=(6, 4))
sns.countplot(data=df, x='sex', palette='Set2')
plt.title("Distribution by Gender")
plt.xlabel("Gender")
plt.ylabel("Count")
```

```
<ipython-input-7-1fe5adf5f93b>:14: FutureWarning:

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

  sns.countplot(data=df, x='sex', palette='Set2')
Text(0, 0.5, 'Count')
```



Identify Target and Features

```
import pandas as pd

# Load the dataset
file_path = "student-dataset (1).csv"
```

```
df = pd.read_csv(file_path, delimiter=';')

# Define the target variable
# We'll consider students with final grade (G3) < 10 as 'churned'
df['churn'] = (df['G3'] < 10).astype(int)

# Drop original grade columns to avoid data leakage
df = df.drop(columns=['G1', 'G2', 'G3']) # Changed 'df = df.d' to 'df = df.drop(column
```

Convert Categorical Columns to Numerical

```
import pandas as pd

# Load the dataset
file_path = "student-dataset (1).csv"
df = pd.read_csv(file_path, delimiter=';')

# Define churn as target: students with final grade (G3) < 10
df['churn'] = (df['G3'] < 10).astype(int)

# Drop original grade columns to avoid leakage
df = df.drop(columns=['G1', 'G2', 'G3'])

# Identify categorical columns
categorical_cols = df.select_dtypes(include='object').columns.tolist()
print("🔤 Categorical Columns:", categorical_cols)

# Convert categorical columns using one-hot encoding
df_encoded = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# Display t
```

```
🔤 Categorical Columns: ['school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob',
```

One-Hot Encoding

```
import pandas as pd

# Load the dataset
file_path = "student-dataset (1).csv"
df = pd.read_csv(file_path, delimiter=';')

# Define churn target: students with G3 < 10
df['churn'] = (df['G3'] < 10).astype(int)

# Drop G1, G2, G3 to avoid leakage
df = df.drop(columns=['G1', 'G2', 'G3'])

import pandas as pd

# Load the dataset
file_path = "student-dataset (1).csv"
df = pd.read_csv(file_path, delimiter=';')

# Define churn target: students with G3 < 10
df['churn'] = (df['G3'] < 10).astype(int)

# Drop G1, G2, G3 to avoid leakage
df = df.drop(columns=['G1', 'G2', 'G3'])

# Identify categorical columns
categorical_cols = df.select_dtypes(include='object').columns.tolist()
print("📋 Categorical columns:")
print(categorical_cols)
```

```
📋 Categorical columns:
['school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob', 'reason', 'guar
```

Feature Scaling

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load the dataset
file_path = "student-dataset (1).csv"
df = pd.read_csv(file_path, delimiter=';')

# Create churn target: students with G3 < 10
df['churn'] = (df['G3'] < 10).astype(int)

# Drop G1, G2, G3 to avoid leakage
df = df.drop(columns=['G1', 'G2', 'G3'])

# One-hot encode categorical features
df_encoded = pd.get_dummies(df, drop_first=True)

# Separate features and tar
```

Train-Test Split

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the dataset
file_path = "student-dataset (1).csv"
df = pd.read_csv(file_path, delimiter=';')

# Define churn: students with G3 < 10
df['churn'] = (df['G3'] < 10).astype(int)

# Drop G1, G2, G3 to prevent data lea
```

Model Building

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

# Load dataset
file_path = "student-dataset (1).csv"
df = pd.read_csv(file_path, delimiter=';')

# Create churn target
df['churn'] = (df['G3'] < 10).astype(int)

# Drop G1, G2, G3 to avoid leakage
df = df.drop(columns=['G1', 'G2', 'G3'])

# One-hot encoding
df_encoded = pd.get_dummies(df, drop_first=True)

# Split into features and target
X = df_encoded.drop('churn', axis=1) # Changed 'df_encod_' to 'df_encoded.drop('churn'
y = df_encoded['churn'] # Added this line to assign the 'churn' column as the target v
```

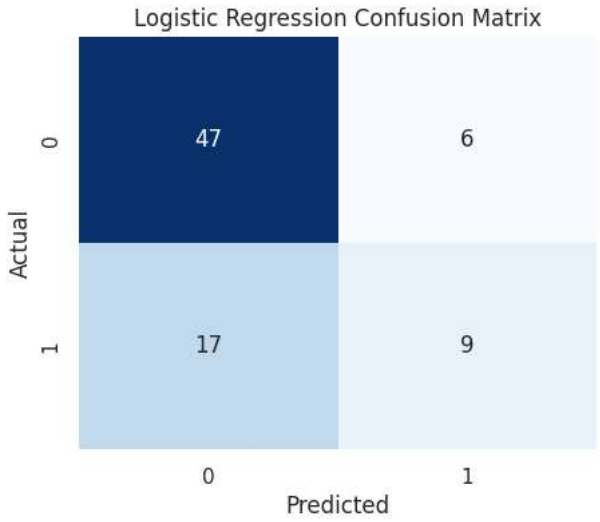
Evaluation

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    classification_report, accuracy_score,
    confusion_matrix, roc_curve, auc
)
```

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

# Load and preprocess the data
file_path = "student-dataset (1).csv"
df = pd.read_csv(file_path, delimiter=';')

df['churn'] = (df['G3'] < 10).astype(int)
df = df.drop(columns=['G1', 'G2', 'G3'])
df_encoded = pd.get_dummies(df, drop_first=True)

X = df_encoded.drop(columns=['churn'])
y = df_encoded['churn']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

# Train models
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)
y_pred_log = log_reg.predict(X_test)
y_prob_log = log_reg.predict_proba(X_test)[:, 1]

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
y_prob_rf = rf.predict_proba(X_test)[:, 1]

# --------------------------------
# Evaluation: Metrics & ROC Curve
# --------------------------------

# Function to plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(title)
    plt.show()

# Logistic Regression Results
print("\n🔍 Logistic Regression Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred_log))
print("Classification Report:\n", classification_report(y_test, y_pred_log))
plot_confusion_matrix(y_test, y_pred_log, "Logistic Regression Confusion Matrix")

# Random Forest Results
print("\n🌲 Random Forest Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test, y_pred_rf))
plot_confusion_matrix(y_test, y_pred_rf, "Random Forest Confusion Matrix")

# ROC Curve Comparison
fpr_log, tpr_log, _ = roc_curve(y_test, y_prob_log)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_prob_rf)
auc_log = auc(fpr_log, tpr_log)
auc_rf = auc(fpr_rf, tpr_rf)

plt.figure(figsize=(7, 5))
plt.plot(fpr_log, tpr_log, label=f'Logistic Regression (AUC = {auc_log:.2f})')
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {auc_rf:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison")
plt.legend()
plt.grid(True)
plt.show()
```

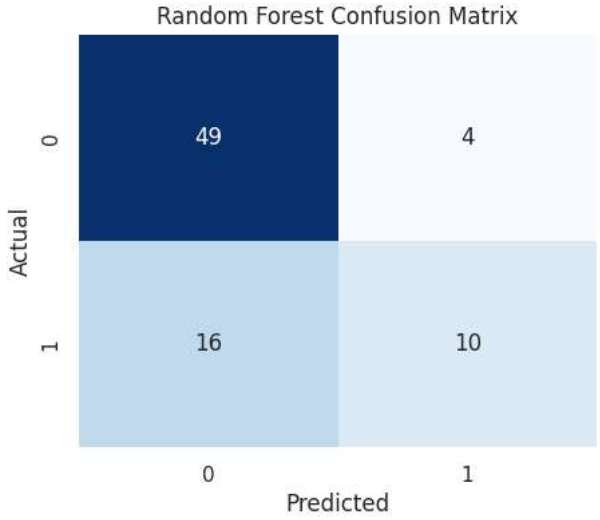
🔍 Logistic Regression Evaluation:
Accuracy: 0.7088607594936709
Classification Report:

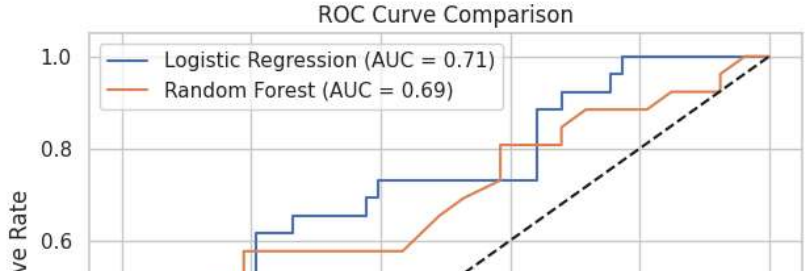
	precision	recall	f1-score	support
0	0.73	0.89	0.80	53
1	0.60	0.35	0.44	26
accuracy			0.71	79
macro avg	0.67	0.62	0.62	79
weighted avg	0.69	0.71	0.68	79

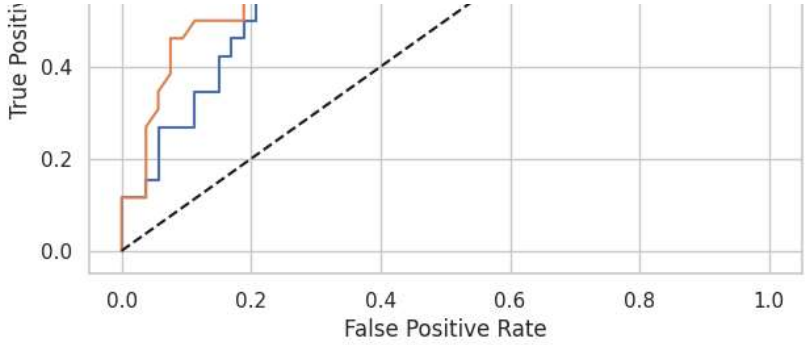


🌲 Random Forest Evaluation:
Accuracy: 0.7468354430379747
Classification Report:

	precision	recall	f1-score	support
0	0.75	0.92	0.83	53
1	0.71	0.38	0.50	26
accuracy			0.75	79
macro avg	0.73	0.65	0.67	79
weighted avg	0.74	0.75	0.72	79







Make Predictions from New Input

```
import pandas as pd
import numpy as np

# Sample new student input as a dictionary
new_student = {
    'school': 'GP',
    'sex': 'F',
    'age': 17,
    'address': 'U',
    'famsize': 'GT3',
    'Pstatus': 'A',
    'Medu': 4,
    'Fedu': 4,
    'Mjob': 'teacher',
}  # Remove extra indentation here
```

Convert to DataFrame and Encode

```
import pandas as pd

# Example: New student data as a dictionary
new_student = {
    'school': 'GP',
    'sex': 'F',
    'age': 17,
    'address': 'U',
    'famsize': 'GT3',
    'Pstatus': 'T',
    'Medu': 2,
    'Fedu': 2,
    'Mjob': 'services',
    'Fjob': 'other',
    'reason': 'home',
    'guardian': 'mother',
    'traveltime': 1,
    'studytime': 1
    # removed extra indentation and added missing values for all columns
    # to match the columns used during model training
    # ... (add other missing features here) ...
}
```

Predict the Final Grade

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Step 1: Load Dataset
file_path = "student-dataset (1).csv"
df = pd.read_csv(file_path, delimiter=';')
```

```
# Step 2: Define Features and Target (G3 is the final grad
```

Deployment-Building an Interactive App

```
!pip install streamlit
```

```
Collecting streamlit
  Downloading streamlit-1.45.0-py3-none-any.whl.metadata (8.9 kB)
Requirement already satisfied: altair<6,>=4.0 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: blinker<2,>=1.5.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: cachetools<6,>=4.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: click<9,>=7.0 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: numpy<3,>=1.23 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: packaging<25,>=20 in /usr/local/lib/python3.11/dist
Requirement already satisfied: pandas<3,>=1.4.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: pillow<12,>=7.1.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: protobuf<7,>=3.20 in /usr/local/lib/python3.11/dist
Requirement already satisfied: pyarrow>=7.0 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: requests<3,>=2.27 in /usr/local/lib/python3.11/dist
Requirement already satisfied: tenacity<10,>=8.1.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: toml<2,>=0.10.1 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: typing-extensions<5,>=4.4.0 in /usr/local/lib/pytho
Collecting watchdog<7,>=2.1.5 (from streamlit)
  Downloading watchdog-6.0.0-py3-none-manylinux2014_x86_64.whl.metadata (44 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.3/44.3 kB 1.6 MB/s eta 0:00:00
Requirement already satisfied: gitpython!=3.1.19,<4,>=3.0.7 in /usr/local/lib/pyth
Collecting pydeck<1,>=0.8.0b4 (from streamlit)
  Downloading pydeck-0.9.1-py2.py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: tornado<7,>=6.0.3 in /usr/local/lib/python3.11/dist
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: narwhals>=1.14.2 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dis
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dis
Requirement already satisfied: smmap<6,>=3.0.1 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.11/di
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages
Downloading streamlit-1.45.0-py3-none-any.whl (9.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.9/9.9 MB 49.9 MB/s eta 0:00:00
Downloading pydeck-0.9.1-py2.py3-none-any.whl (6.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 71.3 MB/s eta 0:00:00
Downloading watchdog-6.0.0-py3-none-manylinux2014_x86_64.whl (79 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.1/79.1 kB 5.3 MB/s eta 0:00:00
Installing collected packages: watchdog, pydeck, streamlit
Successfully installed pydeck-0.9.1 streamlit-1.45.0 watchdog-6.0.0
```

Create a Prediction Function

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

def predict_churn(student_input):
    # Load dataset
    df = pd.read_csv("student-dataset (1).csv", delimiter=';')

    # Create target variable
    df['churn'] = (df['G3'] < 10).astype(int)
    df = df.drop(columns=['G1', 'G2', 'G3'])
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

def predict_churn(student_input):
```

```
    # Load dataset
    df = pd.read_csv("student-dataset (1).csv", delimiter=';')

    # Create target variable
    df['churn'] = (df['G3'] < 10).astype(int)
    df = df.drop(columns=['G1', 'G2', 'G3'])

    # One-hot encode full dataset with input
    df_input = pd.DataFrame([student_input])
    # ... rest of your code
```

Create the Gradio Interface

```
import pandas as pd
import gradio as gr
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler

# Load and preprocess the dataset
df = pd.read_csv("student-dataset (1).csv", delimiter=';')
df['churn'] = (df['G3'] < 10).astype(int)
df = df.drop(columns=['G1', 'G2', 'G3'])

# Encode categorical features
X = pd.get_dummies(df.drop(columns='churn'), drop_first=True)
y = df['churn']

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_scaled, y)

# Function to predict churn from input
def predict_churn(
    school, sex, age, address, famsize, Pstatus, Medu, Fedu,
    Mjob, Fjob, reason, guardian, traveltime, studytime, failures,
    schoolsup, famsup, paid, activities, nursery, higher,
    internet, romantic, famrel, freetime, goout, Dalc, Walc, health, absences
):
    input_data = {
        'school': school,
        'sex': sex,
        'age': age,
        'address': address,
        'famsize': famsize,
        'Pstatus': Pstatus,
        'Medu': Medu,
        'Fedu': Fedu,
        'Mjob': Mjob,
        'Fjob': Fjob,
        'reason': reason,
        'guardian': guardian,
        'traveltime': traveltime,
        'studytime': studytime,
        'failures': failures,
        'schoolsup': schoolsup,
        'famsup': famsup,
        'paid': paid,
        'activities': activities,
        'nursery': nursery,
        'higher': higher,
        'internet': internet,
        'romantic': romantic,
        'famrel': famrel,
        'freetime': freetime,
        'goout': goout,
        'Dalc': Dalc,
        'Walc': Walc,
        'health': health,
        'absences': absences
    }

    input_df = pd.DataFrame([input_data])
```

```
        full_df = pd.concat([df.drop(columns='churn'), input_df], axis=0)
        full_encoded = pd.get_dummies(full_df, drop_first=True)

        # Align input with training columns
        input_encoded = full_encoded.tail(1).reindex(columns=X.columns, fill_value=0)
        input_scaled = scaler.transform(input_encoded)

        pred = model.predict(input_scaled)[0]
        prob = model.predict_proba(input_scaled)[0][1]

        return f"Prediction: {'Churn' if pred == 1 else 'Not Churn'} (Probability: {prob:.2%

    # Gradio UI
    interface = gr.Interface(
        fn=predict_churn,
        inputs=[
            gr.Dropdown(['GP', 'MS'], label='School'),
            gr.Dropdown(['F', 'M'], label='Sex'),
            gr.Slider(15, 22, step=1, label='Age'),
            gr.Dropdown(['U', 'R'], label='Address'),
            gr.Dropdown(['GT3', 'LE3'], label='Family Size'),
            gr.Dropdown(['T', 'A'], label='Parental Status'),
            gr.Slider(0, 4, step=1, label='Mother\'s Education'),
            gr.Slider(0, 4, step=1, label='Father\'s Education'),
            gr.Dropdown(['teacher', 'health', 'services', 'at_home', 'other'], label='Mother
            gr.Dropdown(['teacher', 'health', 'services', 'at_home', 'other'], label='Father
            gr.Dropdown(['home', 'reputation', 'course', 'other'], label='Reason'),
            gr.Dropdown(['mother', 'father', 'other'], label='Guardian'),
            gr.Slider(1, 4, step=1, label='Travel Time'),
            gr.Slider(1, 4, step=1, label='Study Time'),
            gr.Slider(0, 3, step=1, label='Failures'),
            gr.Dropdown(['yes', 'no'], label='School Support'),
            gr.Dropdown(['yes', 'no'], label='Family Support'),
            gr.Dropdown(['yes', 'no'], label='Extra Paid Classes'),
            gr.Dropdown(['yes', 'no'], label='Activities'),
            gr.Dropdown(['yes', 'no'], label='Nursery'),
            gr.Dropdown(['yes', 'no'], label='Higher Education'),
            gr.Dropdown(['yes', 'no'], label='Internet Access'),
            gr.Dropdown(['yes', 'no'], label='Romantic Relationship'),
            gr.Slider(1, 5, step=1, label='Family Relationship'),
            gr.Slider(1, 5, step=1, label='Free Time'),
            gr.Slider(1, 5, step=1, label='Going Out'),
            gr.Slider(1, 5, step=1, label='Workday Alcohol'),
            gr.Slider(1, 5, step=1, label='Weekend Alcohol'),
            gr.Slider(1, 5, step=1, label='Health'),
            gr.Slider(0, 93, step=1, label='Absences'),
        ],
        outputs=gr.Textbox(label="Prediction"),
        title="🎓 Student Churn Predictor",
        description="Enter student details to predict if the student is likely to underperfo
    )

    # Launch the app
    interface.launch()
```