Education Recommendation System

Load Data Set

```
import pandas as pd
df1=pd.read_csv("/content/student-scores.csv")
```

```
Start coding or generate with AI.
```

```
df=df1.copy()
df.head()
```

	id	first_name	last_name	email	gender	part_time_job	absence_days	extracurricular_activities	weekly
0	1	Paul	Casey	paul.casey.1@gslingacademy.com	male	False	3	False	
1	2	Danielle	Sandoval	danielle.sandoval.2@gslingacademy.com	female	False	2	False	
2	3	Tina	Andrews	tina.andrews.3@gslingacademy.com	female	False	9	True	
3	4	Tara	Clark	tara.clark.4@gslingacademy.com	female	False	5	False	
4	5	Anthony	Campos	anthony.campos.5@gslingacademy.com	male	False	5	False	

Drop Irrelevant Columns

```
df.drop(columns=['id','first_name','last_name','email'],axis=1,inplace=True)
```

```
df
```

	gender	part_time_job	absence_days	extracurricular_activities	weekly_self_study_hours	career_aspiration	math_score	history_s
0	male	False	3	False	27	Lawyer	73	
1	female	False	2	False	47	Doctor	90	
2	female	False	9	True	13	Government Officer	81	
3	female	False	5	False	3	Artist	71	
4	male	False	5	False	10	Unknown	84	
...	...	...	...	...	...	...	...	
1995	male	False	2	False	30	Construction Engineer	83	
1996	male	False	2	False	20	Software Engineer	89	
1997	female	False	5	False	14	Software Engineer	97	
1998	female	True	10	True	5	Business Owner	51	
1999	female	False	5	False	27	Accountant	82	

2000 rows × 13 columns

create new features from all score

```
df["total_score"] = df["math_score"] + df["history_score"] + df["physics_score"] + df["chemistry_score"] + df["biology_score"] + df["englisl
df["average_score"] = df["total_score"] / 7
df.head()
```

	gender	part_time_job	absence_days	extracurricular_activities	weekly_self_study_hours	career_aspiration	math_score	history_scor
0	male	False	3	False	27	Lawyer	73	8
1	female	False	2	False	47	Doctor	90	8
2	female	False	9	True	13	Government Officer	81	9
3	female	False	5	False	3	Artist	71	7
4	male	False	5	False	10	Unknown	84	7

```
df['career_aspiration'].value_counts()
```

career_aspiration	count
Software Engineer	315
Business Owner	309
Unknown	223
Banker	169
Lawyer	138
Accountant	126
Doctor	119
Real Estate Developer	83
Stock Investor	73
Construction Engineer	68
Artist	67
Game Developer	63
Government Officer	61
Teacher	59
Designer	56
Scientist	39
Writer	32

```
df['career_aspiration'].unique()
```

```
array(['Lawyer', 'Doctor', 'Government Officer', 'Artist', 'Unknown',
       'Software Engineer', 'Teacher', 'Business Owner', 'Scientist',
       'Banker', 'Writer', 'Accountant', 'Designer',
       'Construction Engineer', 'Game Developer', 'Stock Investor',
       'Real Estate Developer'], dtype=object)
```

```
len(df['career_aspiration'].unique())
```

```
17
```

Encoding Categorical Columns

```
gender_map = {'male': 0, 'female': 1}
part_time_job_map = {False: 0, True: 1}
extracurricular_activities_map = {False: 0, True: 1}
career_aspiration_map = {
        'Lawyer': 0, 'Doctor': 1, 'Government Officer': 2, 'Artist': 3, 'Unknown': 4,
        'Software Engineer': 5, 'Teacher': 6, 'Business Owner': 7, 'Scientist': 8,
        'Banker': 9, 'Writer': 10, 'Accountant': 11, 'Designer': 12,
        'Construction Engineer': 13, 'Game Developer': 14, 'Stock Investor': 15,
        'Real Estate Developer': 16
    }
```

```
# Apply mapping to the DataFrame
df['gender'] = df['gender'].map(gender_map)
df['part_time_job'] = df['part_time_job'].map(part_time_job_map)
df['extracurricular_activities'] = df['extracurricular_activities'].map(extracurricular_activities_map)
df['career_aspiration'] = df['career_aspiration'].map(career_aspiration_map)
```

```
df
```

	gender	part_time_job	absence_days	extracurricular_activities	weekly_self_study_hours	career_aspiration	math_score	history_s
0	0	0	3	0	27	0	73	
1	1	0	2	0	47	1	90	
2	1	0	9	1	13	2	81	
3	1	0	5	0	3	3	71	
4	0	0	5	0	10	4	84	
...	...	...	...	...	...	...	...	
1995	0	0	2	0	30	13	83	
1996	0	0	2	0	20	5	89	
1997	1	0	5	0	14	5	97	
1998	1	1	10	1	5	7	51	
1999	1	0	5	0	27	11	82	

2000 rows × 15 columns

```
df.shape
```

```
(2000, 15)
```

```
df.head()
```

	gender	part_time_job	absence_days	extracurricular_activities	weekly_self_study_hours	career_aspiration	math_score	history_scor
0	0	0	3	0	27	0	73	8
1	1	0	2	0	47	1	90	8
2	1	0	9	1	13	2	81	9
3	1	0	5	0	3	3	71	7
4	0	0	5	0	10	4	84	7

Balance Dataset

```
df['career_aspiration'].unique()
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16])
```

```
df['career_aspiration'].value_counts()
```

career_aspiration	count
5	315
7	309
4	223
9	169
0	138
11	126
1	119
16	83
15	73
13	68
3	67
14	63
2	61
6	59
12	56
8	39
10	32

```
from imblearn.over_sampling import SMOTE

# Create SMOTE object
smote = SMOTE(random_state=42)

# Separate features and target variable
X = df.drop('career_aspiration', axis=1)
y = df['career_aspiration']

# Apply SMOTE to the data
X_resampled, y_resampled = smote.fit_resample(X, y)
```

```
y_resampled.value_counts()
```

	count
career_aspiration	
0	315
1	315
2	315
3	315
4	315
5	315
6	315
7	315
8	315
9	315
10	315
11	315
12	315
13	315
14	315
15	315
16	315

```
y_resampled.shape
```

```
(5355,)
```

Train test Split

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X_resampled,y_resampled,test_size=0.2, random_state=42)
```

```
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

```
((4284, 14), (4284,), (1071, 14), (1071,))
```

Feature Scalling

```
from sklearn.preprocessing import StandardScaler

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler to the training data and transform both training and testing data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
X_train_scaled
```

```
array([[ 1.28299565, -0.25678162, -0.00818145, ...,  0.83256957,
         1.69887402,  1.7074715 ],
       [ 1.28299565, -0.25678162, -0.00818145, ..., -0.6591611 ,
        -0.80753861, -0.80832155],
       [ 1.28299565, -0.25678162, -0.93053284, ...,  1.29873541,
         1.93142777,  1.92251092],
       ...,
       [-0.77942587, -0.25678162,  0.91416993, ..., -0.19299527,
        -0.10987736, -0.09622524],
```

```
       [ 1.28299565, -0.25678162, -0.00818145, ...,  0.64610324,
        -0.2907525 , -0.28493267],
       [-0.77942587, -0.25678162,  0.45299424, ..., -0.28622844,
        -0.93673514, -0.94393154]])
```

```
X_train_scaled.shape
```

```
(4284, 14)
```

Models Training (Multiple Models)

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import warnings
warnings.filterwarnings("ignore")

# Define models
models = {
    "Logistic Regression": LogisticRegression(),
    "Support Vector Classifier": SVC(),
    "Random Forest Classifier": RandomForestClassifier(),
    "K Nearest Neighbors": KNeighborsClassifier(),
    "Decision Tree Classifier": DecisionTreeClassifier(),
    "Gaussian Naive Bayes": GaussianNB(),
    "AdaBoost Classifier": AdaBoostClassifier(),
    "Gradient Boosting Classifier": GradientBoostingClassifier(),
    "XGBoost Classifier": XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
}

# Train and evaluate each model
for name, model in models.items():
    print("="*50)
    print("Model:", name)
    # Train the model
    model.fit(X_train_scaled, y_train)

    # Predict on test set
    y_pred = model.predict(X_test_scaled)

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    classification_rep = classification_report(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Print metrics
    print("Accuracy:", accuracy)
    print("Classification Report:\n", classification_rep)
    print("Confusion Matrix:\n", conf_matrix)
```

```
==================================================
Model: Logistic Regression
Accuracy: 0.48739495798319327
Classification Report:
               precision    recall  f1-score   support

           0       0.45      0.54      0.49        68
           1       0.49      0.62      0.55        72
           2       0.42      0.44      0.43        57
           3       0.52      0.57      0.55        58
           4       0.31      0.17      0.22        66
           5       0.32      0.32      0.32        76
           6       0.58      0.92      0.71        71
           7       0.83      0.90      0.87        61
           8       0.41      0.45      0.43        53
           9       0.29      0.10      0.15        61
          10       0.59      0.71      0.65        63
          11       0.44      0.45      0.45        53
          12       0.31      0.16      0.21        68
```

```
          13       0.38      0.49      0.43        55
          14       0.61      0.93      0.74        57
          15       0.37      0.24      0.29        63
          16       0.55      0.32      0.40        69

    accuracy                           0.49      1071
   macro avg       0.46      0.49      0.46      1071
weighted avg       0.46      0.49      0.46      1071

Confusion Matrix:
 [[37  4  0  0  0  7  0  0  4  1 10  3  0  2  0  0  0]
 [ 2 45  0  0  0  7  0  0 13  0  0  0  0  5  0  0  0]
 [ 0  0 25  5  1  1  9  1  0  0  2  0  4  1  2  2  4]
 [ 0  0  2 33  0  0  2  1  0  0  0  0  0  0 11  0  9]
 [ 6  5  7  3 11  9  7  1  2  3  0  3  3  2  1  2  1]
 [ 8  9  0  0  1 24  1  0  1  7  1  5  3 12  0  4  0]
 [ 0  0  0  0  1  2 65  0  0  1  2  0  0  0  0  0  0]
 [ 0  0  0  3  0  0  0 55  0  0  0  0  0  0  3  0  0]
 [ 4 18  0  0  0  1  0  0 24  0  6  0  0  0  0  0  0]
 [10  1  0  0  3  8  8  0  1  6  2  8  6  6  0  2  0]
 [ 8  2  0  0  1  0  4  0  2  1 45  0  0  0  0  0  0]
 [ 0  1  0  0  4  6  3  0  4  1  0 24  1  3  0  6  0]
 [ 2  2  8  3  5  2  7  0  3  0  4  0 11  6  8  5  2]
 [ 1  2  2  0  3  0  0  0  5  1  1  4  4 27  0  5  0]
 [ 0  0  0  3  0  0  0  0  0  0  0  0  0  0 53  0  1]
 [ 4  3  3  0  5  8  1  2  0  0  2  7  4  7  1 15  1]
 [ 0  0 13 13  0  1  5  6  0  0  1  0  0  0  8  0 22]]
==================================================
Model: Support Vector Classifier
Accuracy: 0.6470588235294118
Classification Report:
               precision    recall  f1-score   support

           0       0.55      0.62      0.58        68
           1       0.60      0.83      0.70        72
           2       0.60      0.74      0.66        57
           3       0.69      0.86      0.77        58
           4       0.55      0.18      0.27        66
           5       0.41      0.32      0.36        76
```

Model Selection (Random Forest)

```
model = RandomForestClassifier()

model.fit(X_train_scaled, y_train)
# Predict on test set
y_pred = model.predict(X_test_scaled)

# Calculate metrics
print("Accuracy: ",accuracy_score(y_test, y_pred))
print("Report: ",classification_report(y_test, y_pred))
print("Confusion Matrix: ",confusion_matrix(y_test, y_pred))
```

```
Accuracy:  0.8403361344537815
Report:                precision    recall  f1-score   support

           0       0.80      0.84      0.82        68
           1       0.81      0.99      0.89        72
           2       0.80      1.00      0.89        57
           3       0.90      0.95      0.92        58
           4       0.80      0.42      0.55        66
           5       0.65      0.46      0.54        76
           6       0.92      0.99      0.95        71
           7       0.95      0.93      0.94        61
           8       0.79      0.98      0.87        53
           9       0.74      0.69      0.71        61
          10       0.93      0.98      0.95        63
          11       0.84      0.72      0.78        53
          12       0.88      0.87      0.87        68
          13       0.74      0.96      0.83        55
          14       0.89      0.98      0.93        57
          15       0.91      0.79      0.85        63
          16       0.92      0.84      0.88        69

    accuracy                           0.84      1071
   macro avg       0.84      0.85      0.83      1071
weighted avg       0.84      0.84      0.83      1071

Confusion Matrix:  [[57  5  0  0  1  0  0  0  1  2  1  1  0  0  0  0  0]
 [ 0 71  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0]
```

```
 [ 0  0 57  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 55  0  0  0  1  0  0  0  0  0  0  2  0  0]
 [ 4  3  8  0 28  7  0  0  1  3  1  3  3  0  1  1  3]
 [ 5  4  0  0  3 35  2  0  3  6  0  0  2 13  0  3  0]
 [ 0  0  0  0  0  0 70  0  0  1  0  0  0  0  0  0  0]
 [ 0  0  0  1  0  0  0 57  0  0  0  0  0  0  2  0  1]
 [ 0  1  0  0  0  0  0  0 52  0  0  0  0  0  0  0  0]
 [ 3  1  1  0  0  4  2  0  2 42  2  2  2  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0 62  0  0  0  0  0  0]
 [ 0  1  0  0  1  4  1  0  3  1  0 38  1  2  0  0  1]
 [ 0  0  2  0  1  1  1  0  1  0  1  0 59  0  2  0  0]
 [ 0  0  0  0  0  0  0  0  1  0  0  0  0 53  0  1  0]
 [ 0  0  0  0  0  0  0  1  0  0  0  0  0  0 56  0  0]
 [ 2  2  0  0  1  1  0  0  0  2  0  1  0  4  0 50  0]
 [ 0  0  3  5  0  1  0  1  1  0  0  0  0  0  0  0 58]]
```

Single Input Predictions

```
X_test_scaled[10]
```

```
array([ 1.28299565, -0.25678162,  1.37534562, -0.33832543, -0.20860229,
       -0.29448353,  0.84117249,  1.40056418,  0.52088807,  0.49520339,
        0.98111738,  1.67166808,  1.69887402,  1.69331945])
```

```
X_test_scaled[10].reshape(1,-1)
```

```
array([[ 1.28299565, -0.25678162,  1.37534562, -0.33832543, -0.20860229,
        -0.29448353,  0.84117249,  1.40056418,  0.52088807,  0.49520339,
         0.98111738,  1.67166808,  1.69887402,  1.69331945]])
```

```
model.predict(X_test_scaled[10].reshape(1,-1))
```

```
array([12])
```

```
model.predict(X_test_scaled[10].reshape(1,-1))[0]
```

```
np.int64(12)
```

```
# test 1
print("Actual Label :", y_test.iloc[10])
print("Model Prediction :",model.predict(X_test_scaled[10].reshape(1,-1))[0])
if y_test.iloc[10]==model.predict(X_test_scaled[10].reshape(1,-1)):
    print("Wow! Model doing well.....")
else:
    print("not sure......")
```

```
Actual Label : 12
Model Prediction : 12
Wow! Model doing well.....
```

```
# test 2
print("Actual Label :", y_test.iloc[300])
print("Model Prediction :",model.predict(X_test_scaled[300].reshape(1,-1))[0])
if y_test.iloc[10]==model.predict(X_test_scaled[10].reshape(1,-1)):
    print("Wow! Model doing well.....")
else:
    print("not sure......")
```

```
Actual Label : 0
Model Prediction : 0
Wow! Model doing well.....
```

```
# test 3
print("Actual Label :", y_test.iloc[23])
print("Model Prediction :",model.predict(X_test_scaled[23].reshape(1,-1))[0])
if y_test.iloc[10]==model.predict(X_test_scaled[10].reshape(1,-1)):
    print("Wow! Model doing well.....")
else:
    print("not sure......")
```

```
Actual Label : 3
Model Prediction : 3
Wow! Model doing well.....
```

Saving & Load Files

```
import pickle

# SAVE FILES
pickle.dump(scaler,open("scaler.pkl",'wb'))
pickle.dump(model,open("model.pkl",'wb'))


# Load the scaler, label encoder, and model
scaler = pickle.load(open("scaler.pkl", 'rb'))
model = pickle.load(open("model.pkl", 'rb'))
```

Recommendation System

```
import pickle
import numpy as np

# Load the scaler, label encoder, model, and class names
scaler = pickle.load(open("scaler.pkl", 'rb'))
model = pickle.load(open("model.pkl", 'rb'))
class_names = ['Lawyer', 'Doctor', 'Government Officer', 'Artist', 'Unknown',
               'Software Engineer', 'Teacher', 'Business Owner', 'Scientist',
               'Banker', 'Writer', 'Accountant', 'Designer',
               'Construction Engineer', 'Game Developer', 'Stock Investor',
               'Real Estate Developer']

def Recommendations(gender, part_time_job, absence_days, extracurricular_activities,
                    weekly_self_study_hours, math_score, history_score, physics_score,
                    chemistry_score, biology_score, english_score, geography_score,
                    total_score,average_score):

    # Encode categorical variables
    gender_encoded = 1 if gender.lower() == 'female' else 0
    part_time_job_encoded = 1 if part_time_job else 0
    extracurricular_activities_encoded = 1 if extracurricular_activities else 0

    # Create feature array
    feature_array = np.array([[gender_encoded, part_time_job_encoded, absence_days, extracurricular_activities_encoded,
                               weekly_self_study_hours, math_score, history_score, physics_score,
                               chemistry_score, biology_score, english_score, geography_score,total_score,average_score]])

    # Scale features
    scaled_features = scaler.transform(feature_array)

    # Predict using the model
    probabilities = model.predict_proba(scaled_features)

    # Get top five predicted classes along with their probabilities
    top_classes_idx = np.argsort(-probabilities[0])[:5]
    top_classes_names_probs = [(class_names[idx], probabilities[0][idx]) for idx in top_classes_idx]

    return top_classes_names_probs


# Example usage 1
final_recommendations = Recommendations(gender='female',
                                        part_time_job=False,
                                        absence_days=2,
                                        extracurricular_activities=False,
                                        weekly_self_study_hours=7,
                                        math_score=65,
                                        history_score=60,
                                        physics_score=97,
                                        chemistry_score=94,
                                        biology_score=71,
                                        english_score=81,
                                        geography_score=66,
                                        total_score=534,
                                        average_score=76.285714)
```

```
print("Top recommended studies with probabilities:")
print("="*50)
for class_name, probability in final_recommendations:
    print(f"{class_name} with probability {probability}")
```

```
Top recommended studies with probabilities:
==================================================
Teacher with probability 0.61
Unknown with probability 0.16
Government Officer with probability 0.08
Real Estate Developer with probability 0.07
Designer with probability 0.02
```

```
# Example usage 2
final_recommendations = Recommendations(gender='female',
                                        part_time_job=False,
                                        absence_days=2,
                                        extracurricular_activities=False,
                                        weekly_self_study_hours=4,
                                        math_score=87,
                                        history_score=73,
                                        physics_score=98,
                                        chemistry_score=91,
                                        biology_score=79,
                                        english_score=60,
                                        geography_score=77,
                                        total_score=583,
                                        average_score=83.285714)

print("Top recommended studies with probabilities:")
print("="*50)
```