Assignment 2

By: Harman B, Antra N, Ederuvie U, Daniel S, Rahul J

What's the Problem?

In university classrooms, teachers want to know how engaged students are because engaged students learn better. But the current ways of measuring engagement (like surveys) are:

1. Subjective: Based on opinions, not facts.
2. Infrequent: Done only occasionally, not during the class.

We need a better, real-time way to measure engagement.

Machine learning can analyze data from devices (biosensors) that measure:

1. Heart rate: Fast heart rate = stress; Slow heart rate = calm.
2. EEG (brain waves): High activity = focused mind.
3. Skin conductance: Sweaty skin = emotional or stressed.
4. Facial expressions: Smiling = happy; Frowning = upset.

This data can help predict how engaged a student is right now.

What's the Goal?

The goal is to build a machine learning model that takes these physiological signals as inputs and outputs one of three engagement levels:

1. Highly Engaged: Deep focus and positive emotions.
2. Moderately Engaged: Some focus but not fully immersed.
3. Disengaged: Bored, stressed, or distracted.

Code

EDA

In [1]:
```
import pandas as pd
data = pd.read_csv('emo.csv')
data.head()
```

Out[1]:

	HeartRate	SkinConductance	EEG	Temperature	PupilDiameter	SmileIntensity	FrownIntensity	CortisolLevel	ActivityLevel	AmbientNoiseLevel	LightingLevel	EmotionalState	CognitiveStat
0	61	8.937204	11.794946	36.501723	3.330181	0.689238	0.189024	0.603035	136	59	394	engaged	distracte
1	60	12.635397	19.151412	36.618910	3.428995	0.561056	0.091367	0.566671	155	39	479	engaged	distracte
2	81	3.660028	6.226098	36.176898	2.819286	0.417951	0.227355	1.422475	55	30	832	partially engaged	focuse
3	119	0.563070	4.542968	37.205293	2.192961	0.140186	0.502965	1.669045	39	40	602	disengaged	focuse
4	118	0.477378	0.996209	37.248118	2.450139	0.064471	0.695604	1.854076	10	42	908	disengaged	focuse

In [2]:
```
# Check if there are any missing values
print(data.isnull().sum())
```

```
HeartRate            0
SkinConductance      0
EEG                  0
Temperature          0
PupilDiameter        0
SmileIntensity       0
FrownIntensity       0
CortisolLevel        0
ActivityLevel        0
AmbientNoiseLevel    0
LightingLevel        0
EmotionalState       0
CognitiveState       0
EngagementLevel      0
dtype: int64
```

In [3]:
```
# Check how many and which columns are categorical
data.dtypes
```

Out[3]:
```
HeartRate              int64
SkinConductance      float64
EEG                  float64
Temperature          float64
PupilDiameter        float64
SmileIntensity       float64
FrownIntensity       float64
CortisolLevel        float64
ActivityLevel          int64
AmbientNoiseLevel      int64
LightingLevel          int64
EmotionalState        object
CognitiveState        object
EngagementLevel        int64
dtype: object
```

In [4]:
```
# Check the unique variables in the Cognitive State column
data['CognitiveState'].unique()
```

Out[4]:
```
array(['distracted', 'focused'], dtype=object)
```

In [5]:
```
# Check the unique variables in the Emotional State column
data['EmotionalState'].unique()
```

Out[5]:
```
array(['engaged', 'partially engaged', 'disengaged'], dtype=object)
```

Data Preprocessing and more EDA

In [6]:
```
# Do one-hot encoding on the categorical columns which are "EmotionalState" and "CognitiveState"
from sklearn.preprocessing import OneHotEncoder

categorical_columns = data.select_dtypes(include = ['object']).columns
```

```
encoder = OneHotEncoder(sparse_output=False)
one_hot_encoded = encoder.fit_transform(data[categorical_columns])
data_encoded = pd.DataFrame(one_hot_encoded, columns=encoder.get_feature_names_out(categorical_columns))
data= pd.concat([data.drop(categorical_columns, axis=1), data_encoded], axis=1)
data
```

Out[6]:

	HeartRate	SkinConductance	EEG	Temperature	PupilDiameter	SmileIntensity	FrownIntensity	CortisolLevel	ActivityLevel	AmbientNoiseLevel	LightingLevel	EngagementLevel	Emotion
0	61	8.937204	11.794946	36.501723	3.330181	0.689238	0.189024	0.603035	136	59	394	3	
1	60	12.635397	19.151412	36.618910	3.428995	0.561056	0.091367	0.566671	155	39	479	1	
2	81	3.660028	6.226098	36.176898	2.819286	0.417951	0.227355	1.422475	55	30	832	3	
3	119	0.563070	4.542968	37.205293	2.192961	0.140186	0.502965	1.669045	39	40	602	3	
4	118	0.477378	0.996209	37.248118	2.450139	0.064471	0.695604	1.854076	10	42	908	3	
...	...	...	...	...	...	...	...	...	...	...	...	...	
995	98	3.897648	7.681519	36.274526	2.624275	0.404309	0.204719	1.215872	65	50	913	2	
996	109	0.439062	0.352790	37.173929	2.489483	0.070776	0.638161	1.826544	23	43	642	2	
997	108	1.077287	1.836462	37.073454	2.370298	0.011001	0.595518	1.781096	8	43	620	2	
998	76	14.260010	19.309704	36.708047	3.393744	0.653693	0.171151	0.783958	110	38	779	1	
999	85	4.676070	8.612581	36.053343	2.527295	0.460965	0.361972	1.060261	73	57	797	3	

1000 rows × 17 columns

Checking whether the dataset is balanced or not

In [7]:
```
class_counts = data['EmotionalState_disengaged'].value_counts()
print(class_counts)
```

```
EmotionalState_disengaged
0.0    679
1.0    321
Name: count, dtype: int64
```

In [9]:
```
class_counts = data['EmotionalState_engaged'].value_counts()
print(class_counts)
```

```
EmotionalState_engaged
0.0    668
1.0    332
Name: count, dtype: int64
```

In [10]:
```
class_counts = data['EmotionalState_partially engaged'].value_counts()
print(class_counts)
```

```
EmotionalState_partially engaged
0.0    653
1.0    347
Name: count, dtype: int64
```

Correlation matrix

In [11]:
```
import pandas as pd
import plotly.express as px

correlation_matrix = data.corr()


fig = px.imshow(
    correlation_matrix,
    text_auto=True,
    color_continuous_scale='Viridis',
    title="Correlation Matrix Heatmap"
)

fig.update_layout(
    coloraxis_colorbar=dict(
        title="Correlation",
        thickness=15,
        len=0.7,
        x=1.05,
        y=0.5,
    ),
    width=800,
    height=800,
    margin=dict(l=20, r=20, t=50, b=20),
)

fig.show()
```

In [12]:
```
# Split the data with Engagement Level as the target variable
from sklearn.model_selection import train_test_split

X = data.drop('EngagementLevel', axis=1)
y = data['EngagementLevel']
```

In [13]:
```
# train test split and validation split
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

In [14]:
```
# scalling train,test,val dataset
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

Training and Testing

Random Forest

In [16]:
```
# randomforest model

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

In [17]:
```
rf_model = RandomForestClassifier(random_state=42, class_weight='balanced')
rf_model.fit(X_train, y_train)
```

```
val_predictions = rf_model.predict(X_val)
print("Validation Set Performance:")
print(classification_report(y_val, val_predictions))
```

```
Validation Set Performance:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        53
           3       1.00      1.00      1.00        88

    accuracy                           1.00       150
   macro avg       1.00      1.00      1.00       150
weighted avg       1.00      1.00      1.00       150
```

In [18]:
```
test_predictions = rf_model.predict(X_test)
print("Test Set Performance:")
print(classification_report(y_test, test_predictions))
```

```
Test Set Performance:
              precision    recall  f1-score   support

           1       1.00      0.96      0.98        25
           2       1.00      1.00      1.00        42
           3       0.99      1.00      0.99        83

    accuracy                           0.99       150
   macro avg       1.00      0.99      0.99       150
weighted avg       0.99      0.99      0.99       150
```

In [19]:
```
cm = confusion_matrix(y_test, test_predictions)
print("Confusion Matrix (Test Set):")
print(cm)
```

```
Confusion Matrix (Test Set):
[[24  0  1]
 [ 0 42  0]
 [ 0  0 83]]
```

Decision Tree

In [21]:
```
from sklearn.metrics import accuracy_score, classification_report
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(class_weight='balanced',random_state=42)
dt.fit(X_train, y_train)
val_predictions = dt.predict(X_val)
print("Validation Set Performance:")
print(classification_report(y_val, val_predictions))
```

```
Validation Set Performance:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        53
           3       1.00      1.00      1.00        88

    accuracy                           1.00       150
   macro avg       1.00      1.00      1.00       150
weighted avg       1.00      1.00      1.00       150
```

In [22]:
```
dt_y_pred = dt.predict(X_test)

dt_accuracy = accuracy_score(y_test, dt_y_pred)
print(f"Decision Tree Model Accuracy: {dt_accuracy*100:.2f}%\n")
print(classification_report(y_test, dt_y_pred))
```

```
Decision Tree Model Accuracy: 99.33%

              precision    recall  f1-score   support

           1       1.00      0.96      0.98        25
           2       1.00      1.00      1.00        42
           3       0.99      1.00      0.99        83

    accuracy                           0.99       150
   macro avg       1.00      0.99      0.99       150
weighted avg       0.99      0.99      0.99       150
```

Which Model is Better?

Both models performed very well, with their difference being minute. Although the Decision Tree had slightly lower metrics, we believe it is the better model. The RF model takes longer to make predictions because it takes a vote from all estimators, where as the DT model has just one estimator. We think the offered by the DT model outweighs the accuracy difference of the two.