



```
from google.colab import drive
drive.mount('/content/drive')
```

 Mounted at /content/drive

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from imblearn.over_sampling import SMOTE
```

```
# Load the dataset
df = pd.read_csv('/content/drive/MyDrive/bmi (1).csv') # Update with your dataset filename
```

```
# Display the first few rows
print(df.head())
```



	Gender	Height	Weight	Index
0	Male	174	96	4
1	Male	189	87	2
2	Female	185	110	4
3	Female	195	104	3
4	Male	149	61	3

```
# Define the category mapping
category_mapping = {
    0: 'Extremely Weak',
    1: 'Weak',
    2: 'Normal',
    3: 'Overweight',
    4: 'Obesity',
    5: 'Extreme Obesity'
}
```

```
# Prepare features and labels
X = df[['Gender', 'Height', 'Weight']]
y = df['Index']
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

```
# Create preprocessing for categorical and numerical features
categorical_features = ['Gender']
numerical_features = ['Height', 'Weight']
preprocessor = ColumnTransformer(
    transformers=[
        ('num', 'passthrough', numerical_features),
        ('cat', OneHotEncoder(), categorical_features)
    ])

```

```
# Fit the preprocessor on training data to get transformed features
X_train_transformed = preprocessor.fit_transform(X_train)
X_test_transformed = preprocessor.transform(X_test)
```

```
# Handle class imbalance using SMOTE
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train_transformed, y_train)
```

```
# Create and fit the model
model = LogisticRegression(max_iter=1000)
model.fit(X_train_res, y_train_res)
```

⚡ /usr/local/lib/python3.10/dist-packages/sklearn/linear\_model/\_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
```

```
    LogisticRegression
```

```
LogisticRegression(max_iter=1000)
```

```
# Evaluate the model
y_pred = model.predict(X_test_transformed)
```

```
# Print classification report
report = classification_report(y_test, y_pred, target_names=list(category_mapping.values()), output_dict=True)
print(classification_report(y_test, y_pred, target_names=list(category_mapping.values())))
```

⚡

	precision	recall	f1-score	support
Extremely Weak	0.60	1.00	0.75	3
Weak	0.25	0.25	0.25	4
Normal	0.83	0.71	0.77	14
Overweight	0.63	0.86	0.73	14
Obesity	0.76	0.73	0.75	26
Extreme Obesity	0.97	0.87	0.92	39
accuracy			0.79	100
macro avg	0.67	0.74	0.69	100
weighted avg	0.81	0.79	0.79	100

```
# Calculate and print the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
```

⚡ Accuracy: 0.7900

```
# Generate and print the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)
```

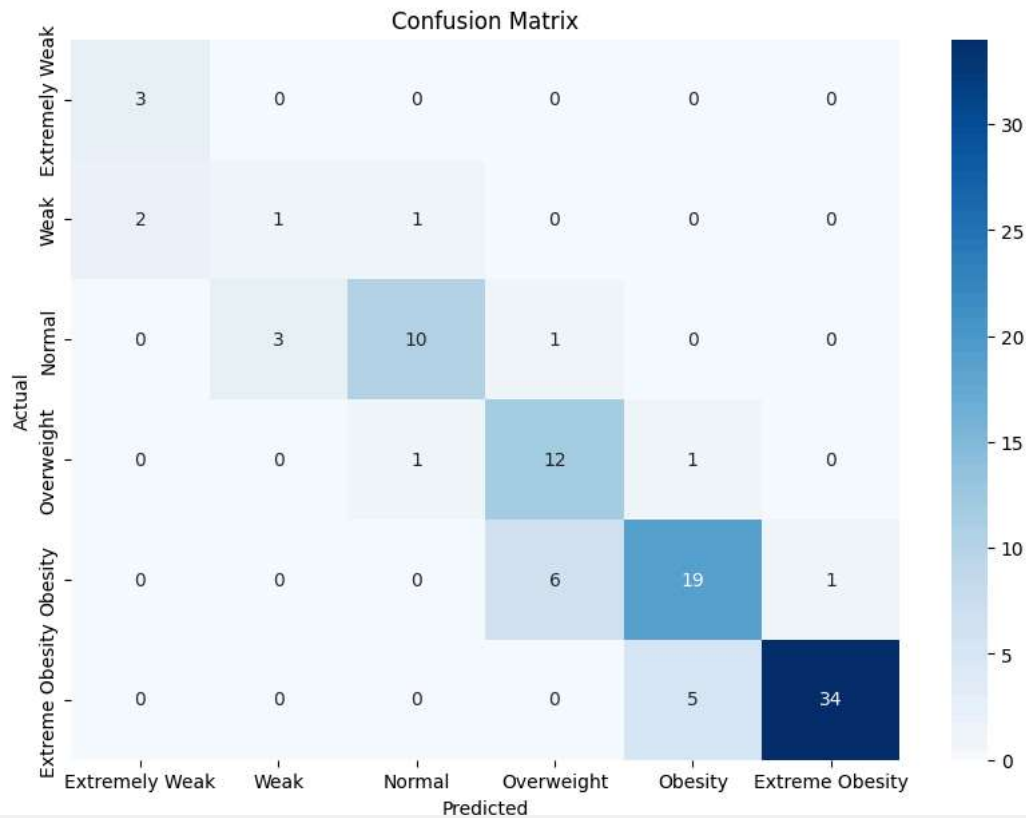
⚡ Confusion Matrix:

```
[[ 3  0  0  0  0  0]
 [ 2  1  1  0  0  0]
 [ 0  3 10  1  0  0]
 [ 0  0  1 12  1  0]
 [ 0  0  0  6 19  1]
 [ 0  0  0  0  5 34]]
```

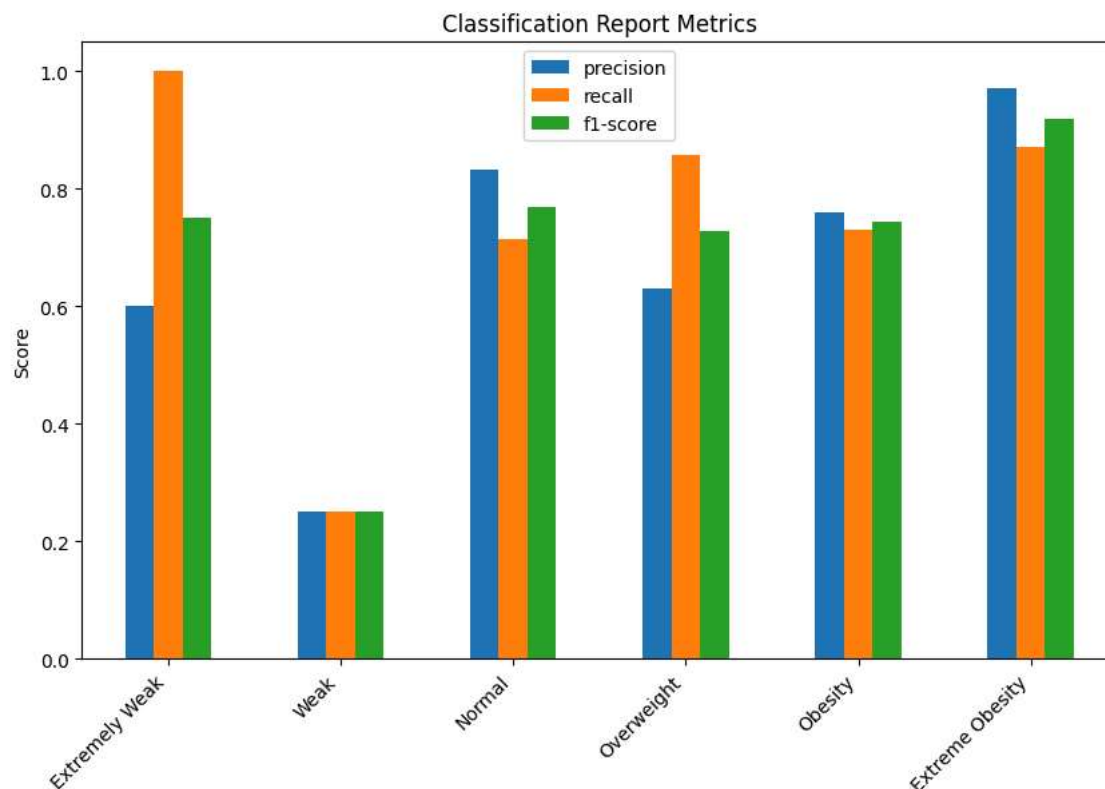
```
# --- Visualization Section ---
```

```
# Plot confusion matrix as a heatmap
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=list(category_mapping.values()), yticklabels=list(category_mapping.va
plt.title('Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

```
# Plot classification report metrics
metrics_df = pd.DataFrame(report).transpose()
```



```
# Bar plot for precision, recall, f1-score
metrics_df[['precision', 'recall', 'f1-score']].iloc[:3].plot(kind='bar', figsize=(10, 6))
plt.title('Classification Report Metrics')
plt.ylabel('Score')
plt.xticks(rotation=45, ha='right')
plt.show()
```



```
# Function to predict BMI index based on user input
def predict_bmi_index(gender, height, weight):
```

```
new_data = pd.DataFrame({
    'Gender': [gender],
    'Height': [height],
    'Weight': [weight]
})

# Transform the input data
new_data_transformed = preprocessor.transform(new_data)


# Predict the index
predicted_index = model.predict(new_data_transformed)[0]
predicted_category = category_mapping[predicted_index]

return predicted_index, predicted_category


# User input
gender = input("Enter gender (Male/Female): ")
height = float(input("Enter height in centimeters (e.g., 175): "))
weight = float(input("Enter weight in kilograms (e.g., 70): "))

# Predict BMI index
predicted_index, predicted_category = predict_bmi_index(gender, height, weight)

print(f'Predicted BMI Index: {predicted_index}, Category: {predicted_category}')
```

 Enter gender (Male/Female): Female  
Enter height in centimeters (e.g., 175): 150  
Enter weight in kilograms (e.g., 70): 40  
Predicted BMI Index: 0, Category: Extremely Weak