```
import pandas as pd
df = pd.read_csv("/content/Obesity prediction.csv")  # Replace with the actual file path
print(df)
```

```
     Gender        Age    Height      Weight family_history FAVC  FCVC  NCP  \
0    Female  21.000000  1.620000   64.000000            yes   no   2.0  3.0
1    Female  21.000000  1.520000   56.000000            yes   no   3.0  3.0
2      Male  23.000000  1.800000   77.000000            yes   no   2.0  3.0
3      Male  27.000000  1.800000   87.000000             no   no   3.0  3.0
4      Male  22.000000  1.780000   89.800000             no   no   2.0  1.0
...     ...        ...       ...         ...            ...  ...   ...  ...
2106 Female  20.976842  1.710730  131.408528            yes  yes   3.0  3.0
2107 Female  21.982942  1.748584  133.742943            yes  yes   3.0  3.0
2108 Female  22.524036  1.752206  133.689352            yes  yes   3.0  3.0
2109 Female  24.361936  1.739450  133.346641            yes  yes   3.0  3.0
2110 Female  23.664709  1.738836  133.472641            yes  yes   3.0  3.0

          CAEC SMOKE      CH2O  SCC       FAF       TUE        CALC  \
0    Sometimes    no  2.000000   no  0.000000  1.000000          no
1    Sometimes   yes  3.000000  yes  3.000000  0.000000   Sometimes
2    Sometimes    no  2.000000   no  2.000000  1.000000  Frequently
3    Sometimes    no  2.000000   no  2.000000  0.000000  Frequently
4    Sometimes    no  2.000000   no  0.000000  0.000000   Sometimes
...        ...   ...       ...  ...       ...       ...         ...
2106 Sometimes    no  1.728139   no  1.676269  0.906247   Sometimes
2107 Sometimes    no  2.005130   no  1.341390  0.599270   Sometimes
2108 Sometimes    no  2.054193   no  1.414209  0.646288   Sometimes
2109 Sometimes    no  2.852339   no  1.139107  0.586035   Sometimes
2110 Sometimes    no  2.863513   no  1.026452  0.714137   Sometimes

                    MTRANS              Obesity
0    Public_Transportation        Normal_Weight
1    Public_Transportation        Normal_Weight
2    Public_Transportation        Normal_Weight
3                  Walking   Overweight_Level_I
4    Public_Transportation  Overweight_Level_II
...                    ...                  ...
2106 Public_Transportation     Obesity_Type_III
2107 Public_Transportation     Obesity_Type_III
2108 Public_Transportation     Obesity_Type_III
2109 Public_Transportation     Obesity_Type_III
2110 Public_Transportation     Obesity_Type_III

[2111 rows x 17 columns]
```

CONVERTING TEXT INTO NUMERICAL

```
from sklearn.preprocessing import LabelEncoder
categorical_cols = ["Gender", "family_history", "FAVC", "CAEC", "SMOKE", "SCC", "CALC", "MTRANS", "Obesity"]

# Apply Label Encoding
label_encoder = LabelEncoder()
for col in categorical_cols:
    df[col] = label_encoder.fit_transform(df[col])

print(df.head())
```

```
   Gender   Age  Height  Weight  family_history  FAVC  FCVC  NCP  CAEC  SMOKE  \
0       0  21.0    1.62    64.0               1     0   2.0  3.0     2      0
1       0  21.0    1.52    56.0               1     0   3.0  3.0     2      1
2       1  23.0    1.80    77.0               1     0   2.0  3.0     2      0
3       1  27.0    1.80    87.0               0     0   3.0  3.0     2      0
4       1  22.0    1.78    89.8               0     0   2.0  1.0     2      0

   CH2O  SCC  FAF  TUE  CALC  MTRANS  Obesity
0   2.0    0  0.0  1.0     3       3        1
1   3.0    1  3.0  0.0     2       3        1
2   2.0    0  2.0  1.0     1       3        1
3   2.0    0  2.0  0.0     1       4        5
4   2.0    0  0.0  0.0     2       3        6
```

y value

```
y = df['Obesity']
print(y)
```

```
0      1
1      1
2      1
3      5
4      6
```

```
        ..
2106    4
2107    4
2108    4
2109    4
2110    4
Name: Obesity, Length: 2111, dtype: int64
```

x value

```
x = df.drop('Obesity', axis=1)
print(x)
```

```
      Gender        Age    Height      Weight  family_history  FAVC  FCVC  \
0          0  21.000000  1.620000   64.000000               1     0   2.0
1          0  21.000000  1.520000   56.000000               1     0   3.0
2          1  23.000000  1.800000   77.000000               1     0   2.0
3          1  27.000000  1.800000   87.000000               0     0   3.0
4          1  22.000000  1.780000   89.800000               0     0   2.0
...      ...        ...       ...         ...             ...   ...   ...
2106       0  20.976842  1.710730  131.408528               1     1   3.0
2107       0  21.982942  1.748584  133.742943               1     1   3.0
2108       0  22.524036  1.752206  133.689352               1     1   3.0
2109       0  24.361936  1.739450  133.346641               1     1   3.0
2110       0  23.664709  1.738836  133.472641               1     1   3.0

      NCP  CAEC  SMOKE      CH2O  SCC       FAF       TUE  CALC  MTRANS
0     3.0     2      0  2.000000    0  0.000000  1.000000     3       3
1     3.0     2      1  3.000000    1  3.000000  0.000000     2       3
2     3.0     2      0  2.000000    0  2.000000  1.000000     1       3
3     3.0     2      0  2.000000    0  2.000000  0.000000     1       4
4     1.0     2      0  2.000000    0  0.000000  0.000000     2       3
...   ...   ...    ...       ...  ...       ...       ...   ...     ...
2106  3.0     2      0  1.728139    0  1.676269  0.906247     2       3
2107  3.0     2      0  2.005130    0  1.341390  0.599270     2       3
2108  3.0     2      0  2.054193    0  1.414209  0.646288     2       3
2109  3.0     2      0  2.852339    0  1.139107  0.586035     2       3
2110  3.0     2      0  2.863513    0  1.026452  0.714137     2       3

[2111 rows x 16 columns]
```

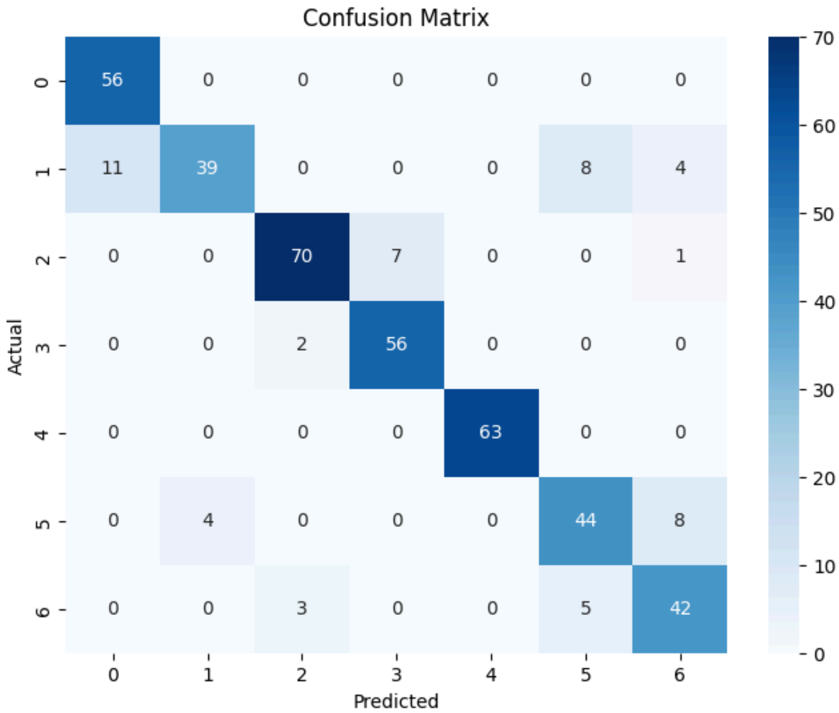
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

LOGISTIC REGRESSION

```
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 StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.compose import ColumnTransformer

# Assuming the last column is the target variable (Adjust accordingly)
X = df.iloc[:, :-1]  # Features (all columns except the last)
y = df.iloc[:, -1]   # Target variable (last column)

# If the target variable is categorical, encode it
if y.dtype == 'object':
    y = pd.factorize(y)[0]

# Splitting dataset 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)

# Identify numerical and categorical features
numerical_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X.select_dtypes(include=['object']).columns

# Create a ColumnTransformer to apply different preprocessing to different columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(sparse_output=False, handle_unknown='ignore'), categorical_features) # One-hot encode categor
    ])

# Fit and transform the data
X_train = preprocessor.fit_transform(X_train)
X_test = preprocessor.transform(X_test)
```

```
# Train Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')  # Adjust for multiclass
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.87
Precision: 0.88
Recall: 0.87
F1-score: 0.87
```

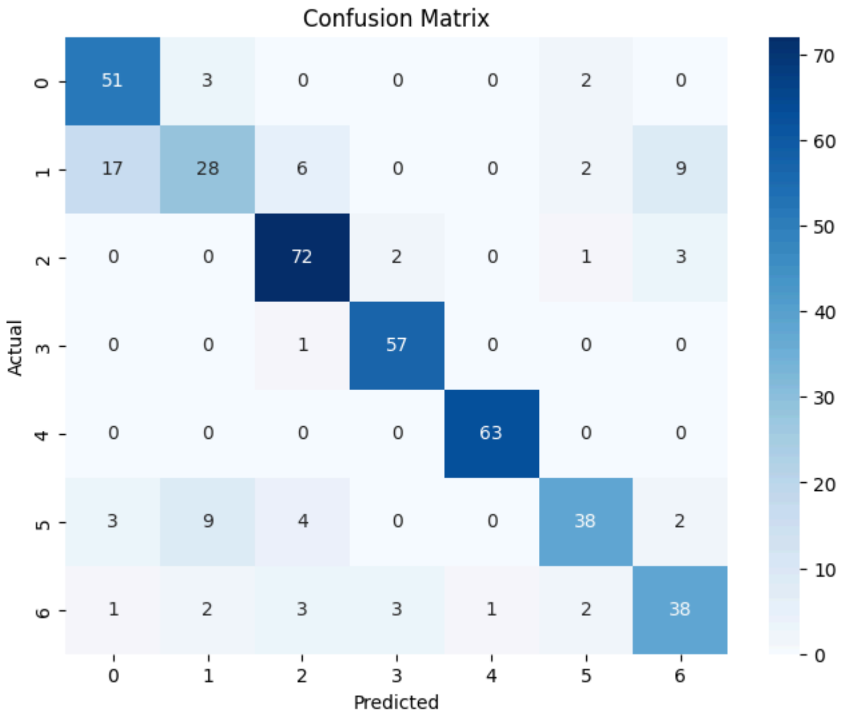


KNeighborsClassifier

```
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 StandardScaler, OneHotEncoder
from sklearn.neighbors import KNeighborsClassifier  # Import KNN Classifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.compose import ColumnTransformer


# Assuming the last column is the target variable (Adjust accordingly)
X = df.iloc[:, :-1]  # Features (all columns except the last)
y = df.iloc[:, -1]   # Target variable (last column)

# If the target variable is categorical, encode it
if y.dtype == 'object':
    y = pd.factorize(y)[0]  # Convert categories to numbers
```

```
# Splitting dataset 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)

# Identify numerical and categorical features
numerical_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X.select_dtypes(include=['object']).columns

# Create a ColumnTransformer to apply different preprocessing to different columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),  # Scale numerical features
        ('cat', OneHotEncoder(sparse_output=False, handle_unknown='ignore'), categorical_features)  # One-hot encode catego
    ])

# Fit and transform the data
X_train = preprocessor.fit_transform(X_train)
X_test = preprocessor.transform(X_test)

# Train KNN model (choosing k=5, you can tune it)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Predictions
y_pred = knn.predict(X_test)

# Evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')  # Adjust for multiclass
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

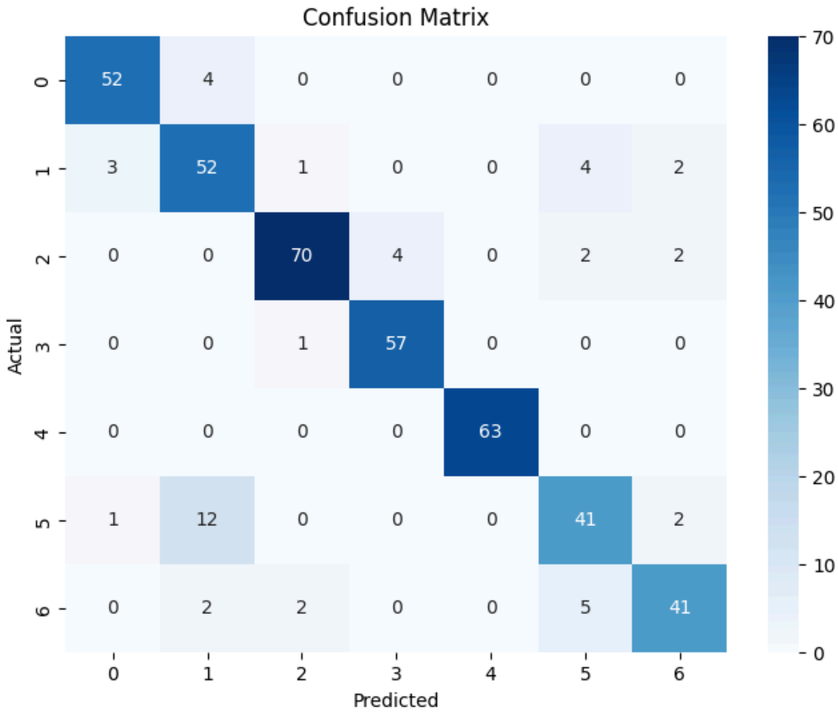
```
Accuracy: 0.82
Precision: 0.82
Recall: 0.82
F1-score: 0.81
```



SVM

```
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 StandardScaler, OneHotEncoder
from sklearn.svm import SVC  # Import SVM Classifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.compose import ColumnTransformer


# Assuming the last column is the target variable (Adjust accordingly)
X = df.iloc[:, :-1]  # Features (all columns except the last)
y = df.iloc[:, -1]   # Target variable (last column)

# If the target variable is categorical, encode it
if y.dtype == 'object':
    y = pd.factorize(y)[0]  # Convert categories to numbers

# Splitting dataset 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)

# Identify numerical and categorical features
numerical_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X.select_dtypes(include=['object']).columns

# Create a ColumnTransformer to apply different preprocessing to different columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),  # Scale numerical features
        ('cat', OneHotEncoder(sparse_output=False, handle_unknown='ignore'), categorical_features)  # One-hot encode catego
    ])

# Fit and transform the data
X_train = preprocessor.fit_transform(X_train)
X_test = preprocessor.transform(X_test)

# Train SVM model (using RBF kernel, you can change to 'linear' or 'poly' if needed)
svm_model = SVC(kernel='rbf', C=1.0, random_state=42)
svm_model.fit(X_train, y_train)

# Predictions
y_pred = svm_model.predict(X_test)

# Evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')  # Adjust for multiclass
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.89
Precision: 0.89
Recall: 0.89
F1-score: 0.89
```

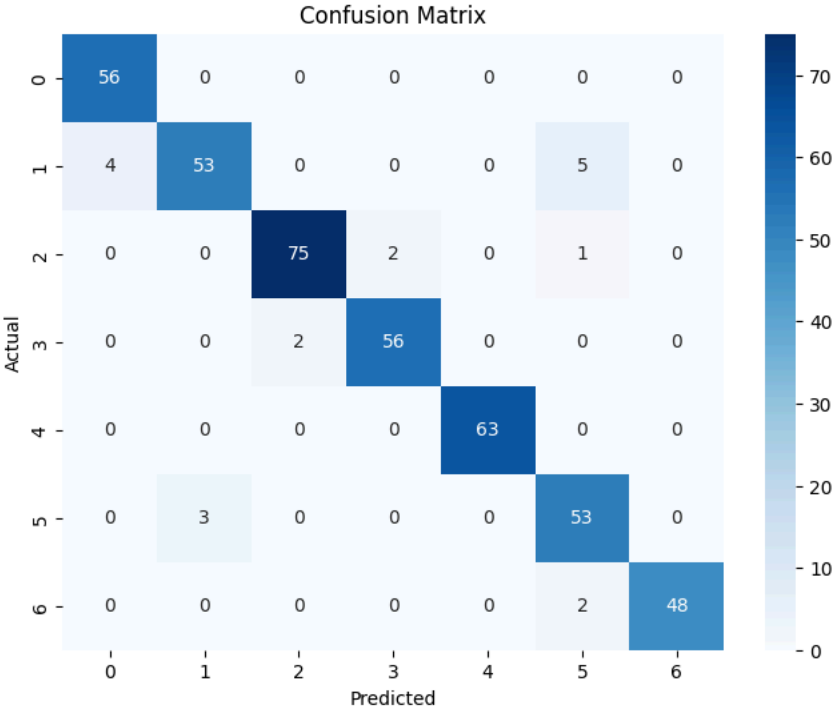


XGBoost Classifier

```
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 StandardScaler, OneHotEncoder
from xgboost import XGBClassifier  # Import XGBoost Classifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.compose import ColumnTransformer


# Assuming the last column is the target variable (Adjust accordingly)
X = df.iloc[:, :-1]  # Features (all columns except the last)
y = df.iloc[:, -1]   # Target variable (last column)

# If the target variable is categorical, encode it
if y.dtype == 'object':
    y = pd.factorize(y)[0]  # Convert categories to numbers

# Splitting dataset 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)

# Identify numerical and categorical features
numerical_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X.select_dtypes(include=['object']).columns

# Create a ColumnTransformer to apply different preprocessing to different columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),  # Scale numerical features
        ('cat', OneHotEncoder(sparse_output=False, handle_unknown='ignore'), categorical_features)  # One-hot encode categ
    ])

# Fit and transform the data
X_train = preprocessor.fit_transform(X_train)
X_test = preprocessor.transform(X_test)

# Train XGBoost Classifier model
xgb_model = XGBClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_model.fit(X_train, y_train)

# Predictions
y_pred = xgb_model.predict(X_test)

# Evaluation metrics
```

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')  # Adjust for multiclass
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.96
Precision: 0.96
Recall: 0.96
F1-score: 0.95
```



CatBoost Classifier

```
!pip install catboost -q
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.7/98.7 MB 8.2 MB/s eta 0:00:00
```

```
import subprocess
import sys

# Function to install CatBoost if not installed
def install_catboost():
    try:
        import catboost
    except ModuleNotFoundError:
        print("Installing CatBoost...")
        subprocess.check_call([sys.executable, "-m", "pip", "install", "catboost"])

# Install CatBoost if missing
install_catboost()

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from catboost import CatBoostClassifier
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matr:


# Select categorical columns for encoding
categorical_cols = ["Gender", "family_history", "FAVC", "CAEC", "SMOKE", "SCC", "CALC", "MTRANS", "Obesity"]

# Apply Label Encoding to categorical columns
label_encoder = LabelEncoder()
for col in categorical_cols:
    df[col] = label_encoder.fit_transform(df[col])

# Define features (X) and target (y)
X = df.drop(columns=["Obesity"])  # Features
y = df["Obesity"]  # Target variable (Categorical)

# Split into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Standardize numerical features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the CatBoost Classifier
model = CatBoostClassifier(iterations=1000, learning_rate=0.05, depth=8, loss_function='MultiClass', verbose=100)

# Fit the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
y_pred = np.round(y_pred).astype(int)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')  # Weighted for multiclass
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}\n")

# Classification Report
print("Classification Report:\n", classification_report(y_test, y_pred))

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot Confusion Matrix inside a box
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False, linewidths=1, linecolor='black', square=True,
            xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()

# Plot feature importance
feature_importance = model.get_feature_importance()
feature_names = X.columns

plt.figure(figsize=(10, 6))
plt.barh(feature_names, feature_importance, color='skyblue')
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Feature Importance in CatBoost')
plt.show()
```

```
0:  learn: 1.8344600    total: 174ms    remaining: 2m 53s
100:    learn: 0.2519881    total: 12.1s    remaining: 1m 47s
200:    learn: 0.1176434    total: 24.6s    remaining: 1m 37s
300:    learn: 0.0693122    total: 38.1s    remaining: 1m 28s
400:    learn: 0.0494986    total: 48.8s    remaining: 1m 12s
500:    learn: 0.0378369    total: 57.4s    remaining: 57.2s
600:    learn: 0.0301517    total: 1m 5s    remaining: 43.2s
```

Adam (Adaptive Moment Estimation) with epoch value 30

```
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt

# Load the dataset
file_path = "/content/Obesity prediction.csv"
df = pd.read_csv(file_path)

# Check and clean column names
df.columns = df.columns.str.strip()  # Remove leading/trailing spaces
print("Columns in dataset:", df.columns)  # Debugging step

# Ensure correct target column name
target_column = "Obesity"  # Changed from "NObeyesdad" to "Obesity"
if target_column not in df.columns:
    raise ValueError(f"Target column '{target_column}' not found in dataset. Available columns: {df.columns}")

# Encode categorical variables
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le  # Save for future decoding if needed

# Separate features and target
X = df.drop(columns=[target_column])  # Ensure correct target column
y = df[target_column]

# Normalize features (Standardization)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Convert target to categorical (if classification)
num_classes = len(np.unique(y))
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

# Build the neural network
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')  # Output layer for multi-class classification
])

# Compile model with Adam optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model with actual epochs
epochs = 30  # Set the number of epochs for training
history = model.fit(X_train, y_train, epochs=epochs, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc * 100:.2f}%")

# Generate predictions for the test set
y_pred = model.predict(X_test)

# Convert one-hot encoded predictions and true labels to their original classes
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

# Generate confusion matrix
cm = confusion_matrix(y_test_classes, y_pred_classes)

# Calculate Precision, Recall, and F1-score
```

```
precision = precision_score(y_test_classes, y_pred_classes, average='weighted')
recall = recall_score(y_test_classes, y_pred_classes, average='weighted')
f1 = f1_score(y_test_classes, y_pred_classes, average='weighted')

print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")

# Display confusion matrix using ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=label_encoders[target_column].classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()

# Plot the graph for epochs vs accuracy
plt.plot(range(epochs), history.history['accuracy'], label='Training Accuracy')
plt.plot(range(epochs), history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy vs Epochs')
plt.legend()
plt.grid(True)
plt.show()

# Save the model (Optional)
model.save("obesity_prediction_model.h5")
```

```
Columns in dataset: Index(['Gender', 'Age', 'Height', 'Weight', 'family_history', 'FAVC', 'FCVC',
       'NCP', 'CAEC', 'SMOKE', 'CH2O', 'SCC', 'FAF', 'TUE', 'CALC', 'MTRANS',
       'Obesity'],
      dtype='object')
Epoch 1/30
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
53/53 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.2838 - loss: 1.8251 - val_accuracy: 0.5012 - val_loss: 1.4431
Epoch 2/30
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5696 - loss: 1.3300 - val_accuracy: 0.6596 - val_loss: 1.0968
Epoch 3/30
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.6377 - loss: 1.0405 - val_accuracy: 0.6856 - val_loss: 0.8846
Epoch 4/30
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.7193 - loss: 0.8251 - val_accuracy: 0.7400 - val_loss: 0.7426
Epoch 5/30
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.7727 - loss: 0.6750 - val_accuracy: 0.7636 - val_loss: 0.6424
Epoch 6/30
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.8083 - loss: 0.5543 - val_accuracy: 0.8038 - val_loss: 0.5586
Epoch 7/30
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.8605 - loss: 0.4626 - val_accuracy: 0.8369 - val_loss: 0.4953
Epoch 8/30
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8547 - loss: 0.4452 - val_accuracy: 0.8534 - val_loss: 0.4455
Epoch 9/30
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.8915 - loss: 0.3666 - val_accuracy: 0.8747 - val_loss: 0.4042
Epoch 10/30
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.8874 - loss: 0.3375 - val_accuracy: 0.8676 - val_loss: 0.3713
Epoch 11/30
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9144 - loss: 0.2911 - val_accuracy: 0.8842 - val_loss: 0.3483
Epoch 12/30
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9237 - loss: 0.2599 - val_accuracy: 0.8983 - val_loss: 0.3233
Epoch 13/30
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9292 - loss: 0.2288 - val_accuracy: 0.9102 - val_loss: 0.3038
Epoch 14/30
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9472 - loss: 0.2104 - val_accuracy: 0.9102 - val_loss: 0.2907
Epoch 15/30
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9469 - loss: 0.2005 - val_accuracy: 0.9031 - val_loss: 0.2740
Epoch 16/30
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9484 - loss: 0.1815 - val_accuracy: 0.9125 - val_loss: 0.2614
Epoch 17/30
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9576 - loss: 0.1608 - val_accuracy: 0.9267 - val_loss: 0.2502
```

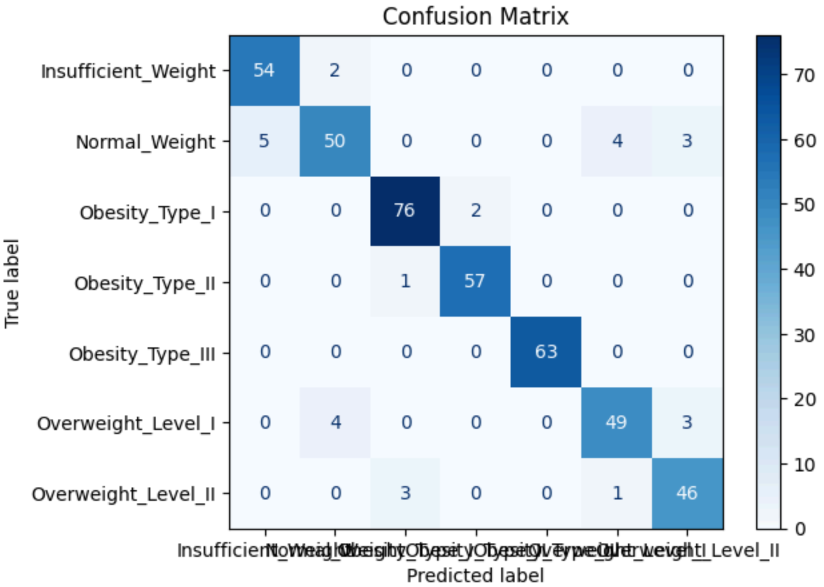
EPOCH VALUE 60

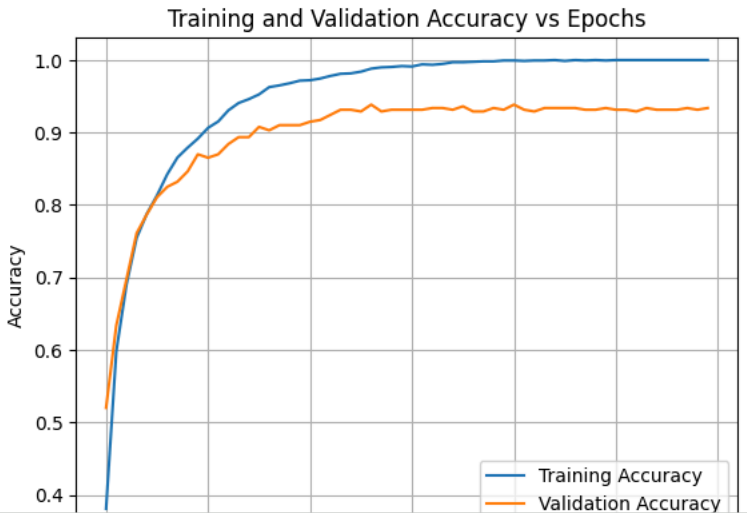
```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt

# Load the dataset
file_path = "/content/Obesity prediction.csv"
df = pd.read_csv(file_path)

# Check and clean column names
df.columns = df.columns.str.strip()  # Remove leading/trailing spaces
print("Columns in dataset:", df.columns)  # Debugging step

# Ensure correct target column name
target_column = "Obesity"  # Changed from "NObeyesdad" to "Obesity"
if target_column not in df.columns:
    raise ValueError(f"Target column '{target_column}' not found in dataset. Available columns: {df.columns}")

# Encode categorical variables
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le  # Save for future decoding if needed

# Separate features and target
X = df.drop(columns=[target_column])  # Ensure correct target column
y = df[target_column]

# Normalize features (Standardization)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Convert target to categorical (if classification)
num_classes = len(np.unique(y))
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

# Build the neural network
```

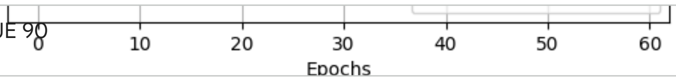
```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')  # Output layer for multi-class classification
])

# Compile model with Adam optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model with actual epochs
epochs = 60  # Set the number of epochs for training
history = model.fit(X_train, y_train, epochs=epochs, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc * 100:.2f}%")

# Generate predictions for the test set
y_pred = model.predict(X_test)

# Convert one-hot encoded predictions and true labels to their original classes
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

# Generate confusion matrix
cm = confusion_matrix(y_test_classes, y_pred_classes)

# Calculate Precision, Recall, and F1-score
precision = precision_score(y_test_classes, y_pred_classes, average='weighted')
recall = recall_score(y_test_classes, y_pred_classes, average='weighted')
f1 = f1_score(y_test_classes, y_pred_classes, average='weighted')

print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")

# Display confusion matrix using ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=label_encoders[target_column].classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()

# Plot the graph for epochs vs accuracy
plt.plot(range(epochs), history.history['accuracy'], label='Training Accuracy')
plt.plot(range(epochs), history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy vs Epochs')
plt.legend()
plt.grid(True)
plt.show()

# Save the model (Optional)
model.save("obesity_prediction_model.h5")
```

```
Columns in dataset: Index(['Gender', 'Age', 'Height', 'Weight', 'family_history', 'FAVC', 'FCVC',
       'NCP', 'CAEC', 'SMOKE', 'CH2O', 'SCC', 'FAF', 'TUE', 'CALC', 'MTRANS',
       'Obesity'],
      dtype='object')
Epoch 1/60
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
53/53 ━━━━━━━━━━━━━━━━━━━━ 6s 38ms/step - accuracy: 0.2505 - loss: 1.8860 - val_accuracy: 0.5201 - val_loss: 1.4675
Epoch 2/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.5748 - loss: 1.3496 - val_accuracy: 0.6336 - val_loss: 1.0547
Epoch 3/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.6561 - loss: 1.0059 - val_accuracy: 0.6974 - val_loss: 0.8478
Epoch 4/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.7514 - loss: 0.7796 - val_accuracy: 0.7612 - val_loss: 0.7211
Epoch 5/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - accuracy: 0.7938 - loss: 0.6458 - val_accuracy: 0.7872 - val_loss: 0.6327
Epoch 6/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.7952 - loss: 0.5862 - val_accuracy: 0.8109 - val_loss: 0.5630
Epoch 7/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.8514 - loss: 0.4759 - val_accuracy: 0.8251 - val_loss: 0.5083
Epoch 8/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.8630 - loss: 0.4322 - val_accuracy: 0.8322 - val_loss: 0.4621
Epoch 9/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.8637 - loss: 0.4031 - val_accuracy: 0.8463 - val_loss: 0.4350
Epoch 10/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.8821 - loss: 0.3644 - val_accuracy: 0.8700 - val_loss: 0.3967
Epoch 11/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9009 - loss: 0.3177 - val_accuracy: 0.8652 - val_loss: 0.3780
Epoch 12/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.9115 - loss: 0.2889 - val_accuracy: 0.8700 - val_loss: 0.3476
Epoch 13/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 22ms/step - accuracy: 0.9352 - loss: 0.2622 - val_accuracy: 0.8842 - val_loss: 0.3243
Epoch 14/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - accuracy: 0.9443 - loss: 0.2346 - val_accuracy: 0.8936 - val_loss: 0.3112
Epoch 15/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9503 - loss: 0.2083 - val_accuracy: 0.8936 - val_loss: 0.2900
Epoch 16/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9567 - loss: 0.1889 - val_accuracy: 0.9078 - val_loss: 0.2800
Epoch 17/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.9679 - loss: 0.1723 - val_accuracy: 0.9031 - val_loss: 0.2728
Epoch 18/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9727 - loss: 0.1593 - val_accuracy: 0.9102 - val_loss: 0.2613
Epoch 19/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9639 - loss: 0.1483 - val_accuracy: 0.9102 - val_loss: 0.2492
Epoch 20/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9739 - loss: 0.1309 - val_accuracy: 0.9102 - val_loss: 0.2432
Epoch 21/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9677 - loss: 0.1322 - val_accuracy: 0.9149 - val_loss: 0.2382
Epoch 22/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9786 - loss: 0.1132 - val_accuracy: 0.9173 - val_loss: 0.2380
Epoch 23/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9758 - loss: 0.1103 - val_accuracy: 0.9243 - val_loss: 0.2271
Epoch 24/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9849 - loss: 0.1033 - val_accuracy: 0.9314 - val_loss: 0.2219
Epoch 25/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9813 - loss: 0.0909 - val_accuracy: 0.9314 - val_loss: 0.2185
Epoch 26/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9828 - loss: 0.0837 - val_accuracy: 0.9291 - val_loss: 0.2134
Epoch 27/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9908 - loss: 0.0808 - val_accuracy: 0.9385 - val_loss: 0.2076
Epoch 28/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9944 - loss: 0.0703 - val_accuracy: 0.9291 - val_loss: 0.2168
Epoch 29/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9910 - loss: 0.0666 - val_accuracy: 0.9314 - val_loss: 0.2085
Epoch 30/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9938 - loss: 0.0593 - val_accuracy: 0.9314 - val_loss: 0.2135
Epoch 31/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9935 - loss: 0.0596 - val_accuracy: 0.9314 - val_loss: 0.2047
Epoch 32/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9963 - loss: 0.0538 - val_accuracy: 0.9314 - val_loss: 0.2066
Epoch 33/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9928 - loss: 0.0513 - val_accuracy: 0.9338 - val_loss: 0.2023
Epoch 34/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9976 - loss: 0.0451 - val_accuracy: 0.9338 - val_loss: 0.1967
Epoch 35/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9986 - loss: 0.0445 - val_accuracy: 0.9314 - val_loss: 0.1982
Epoch 36/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9985 - loss: 0.0407 - val_accuracy: 0.9362 - val_loss: 0.2045
Epoch 37/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9978 - loss: 0.0413 - val_accuracy: 0.9291 - val_loss: 0.2021
Epoch 38/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9975 - loss: 0.0312 - val_accuracy: 0.9291 - val_loss: 0.1938
Epoch 39/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9996 - loss: 0.0353 - val_accuracy: 0.9338 - val_loss: 0.1980
Epoch 40/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9982 - loss: 0.0301 - val_accuracy: 0.9314 - val_loss: 0.2026
Epoch 41/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9993 - loss: 0.0293 - val_accuracy: 0.9385 - val_loss: 0.1998
Epoch 42/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9986 - loss: 0.0274 - val_accuracy: 0.9314 - val_loss: 0.1976
```

```
Epoch 43/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9996 - loss: 0.0276 - val_accuracy: 0.9291 - val_loss: 0.1998
Epoch 44/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9995 - loss: 0.0236 - val_accuracy: 0.9338 - val_loss: 0.1992
Epoch 45/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0237 - val_accuracy: 0.9338 - val_loss: 0.2051
Epoch 46/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9993 - loss: 0.0194 - val_accuracy: 0.9338 - val_loss: 0.1985
Epoch 47/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0215 - val_accuracy: 0.9338 - val_loss: 0.2037
Epoch 48/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9998 - loss: 0.0190 - val_accuracy: 0.9314 - val_loss: 0.2080
Epoch 49/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0205 - val_accuracy: 0.9314 - val_loss: 0.1975
Epoch 50/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9994 - loss: 0.0166 - val_accuracy: 0.9338 - val_loss: 0.1987
Epoch 51/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 1.0000 - loss: 0.0174 - val_accuracy: 0.9314 - val_loss: 0.2001
Epoch 52/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 1.0000 - loss: 0.0145 - val_accuracy: 0.9314 - val_loss: 0.1973
Epoch 53/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 1.0000 - loss: 0.0139 - val_accuracy: 0.9291 - val_loss: 0.2011
Epoch 54/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 1.0000 - loss: 0.0135 - val_accuracy: 0.9338 - val_loss: 0.1990
Epoch 55/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 1.0000 - loss: 0.0133 - val_accuracy: 0.9314 - val_loss: 0.2053
Epoch 56/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0113 - val_accuracy: 0.9314 - val_loss: 0.2004
Epoch 57/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0123 - val_accuracy: 0.9314 - val_loss: 0.2042
Epoch 58/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0109 - val_accuracy: 0.9338 - val_loss: 0.2084
Epoch 59/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0102 - val_accuracy: 0.9314 - val_loss: 0.2061
Epoch 60/60
53/53 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0102 - val_accuracy: 0.9338 - val_loss: 0.2082
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9323 - loss: 0.1908
Test Accuracy: 93.38%
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
Precision: 0.9333
Recall: 0.9338
F1-score: 0.9330
```







EPOCH VALUE 90

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt

# Load the dataset
file_path = "/content/Obesity prediction.csv"
df = pd.read_csv(file_path)

# Check and clean column names
df.columns = df.columns.str.strip()  # Remove leading/trailing spaces
print("Columns in dataset:", df.columns)  # Debugging step

# Ensure correct target column name
target_column = "Obesity"  # Changed from "NObeyesdad" to "Obesity"
if target_column not in df.columns:
    raise ValueError(f"Target column '{target_column}' not found in dataset. Available columns: {df.columns}")

# Encode categorical variables
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le  # Save for future decoding if needed

# Separate features and target
X = df.drop(columns=[target_column])  # Ensure correct target column
y = df[target_column]

# Normalize features (Standardization)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Convert target to categorical (if classification)
num_classes = len(np.unique(y))
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

# Build the neural network
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')  # Output layer for multi-class classification
])

# Compile model with Adam optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model with actual epochs
epochs = 90  # Set the number of epochs for training
history = model.fit(X_train, y_train, epochs=epochs, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc * 100:.2f}%")

# Generate predictions for the test set
y_pred = model.predict(X_test)

# Convert one-hot encoded predictions and true labels to their original classes
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

# Generate confusion matrix
cm = confusion_matrix(y_test_classes, y_pred_classes)

# Calculate Precision, Recall, and F1-score
precision = precision_score(y_test_classes, y_pred_classes, average='weighted')
recall = recall_score(y_test_classes, y_pred_classes, average='weighted')
f1 = f1_score(y_test_classes, y_pred_classes, average='weighted')
```

```
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")

# Display confusion matrix using ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=label_encoders[target_column].classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()

# Plot the graph for epochs vs accuracy
plt.plot(range(epochs), history.history['accuracy'], label='Training Accuracy')
plt.plot(range(epochs), history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy vs Epochs')
plt.legend()
plt.grid(True)
plt.show()

# Save the model (Optional)
model.save("obesity_prediction_model.h5")
```