```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score, roc_curve

# Set plot styles
sns.set(style="whitegrid")
plt.rcParams["figure.figsize"] = (10, 6)

# -------------------------
# Step 1: Load the Dataset
# -------------------------
df = pd.read_csv("/content/dataset.csv")
print("Shape:", df.shape)
print(df.head())
print(df.info())
```

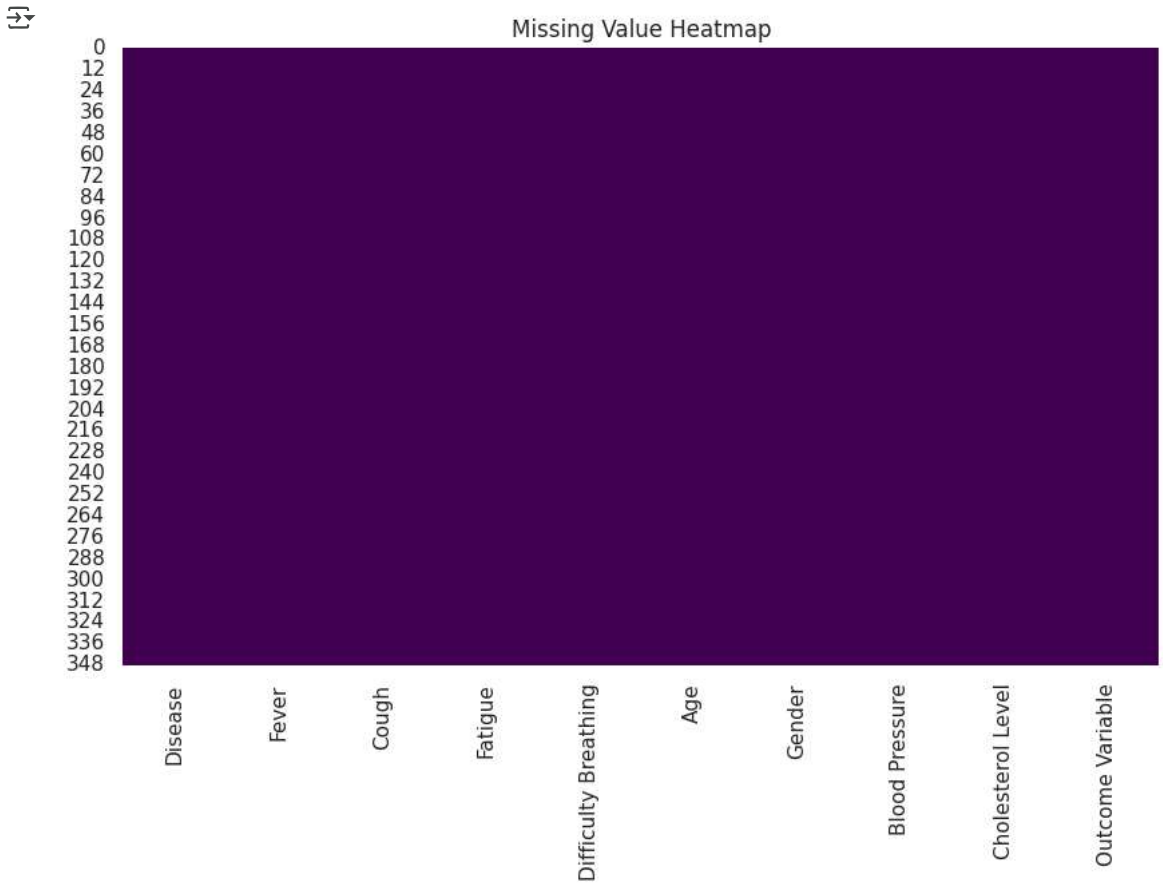
```
Shape: (349, 10)
       Disease Fever Cough Fatigue Difficulty Breathing  Age  Gender  \
0    Influenza   Yes    No     Yes                  Yes   19  Female
1  Common Cold    No   Yes     Yes                   No   25  Female
2       Eczema    No   Yes     Yes                   No   25  Female
3       Asthma   Yes   Yes      No                  Yes   25    Male
4       Asthma   Yes   Yes      No                  Yes   25    Male

  Blood Pressure Cholesterol Level Outcome Variable
0            Low            Normal         Positive
1         Normal            Normal         Negative
2         Normal            Normal         Negative
3         Normal            Normal         Positive
4         Normal            Normal         Positive
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 349 entries, 0 to 348
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Disease               349 non-null    object
 1   Fever                 349 non-null    object
 2   Cough                 349 non-null    object
 3   Fatigue               349 non-null    object
 4   Difficulty Breathing  349 non-null    object
 5   Age                   349 non-null    int64
 6   Gender                349 non-null    object
 7   Blood Pressure        349 non-null    object
 8   Cholesterol Level     349 non-null    object
 9   Outcome Variable      349 non-null    object
dtypes: int64(1), object(9)
memory usage: 27.4+ KB
None
```

```
# Step 2: Data Cleaning
# ----------------------------
# Check for missing values
print("\nMissing values:\n", df.isnull().sum())
```

```
Missing values:
 Disease                 0
Fever                   0
Cough                   0
Fatigue                 0
Difficulty Breathing    0
Age                     0
Gender                  0
Blood Pressure          0
Cholesterol Level       0
Outcome Variable        0
dtype: int64
```

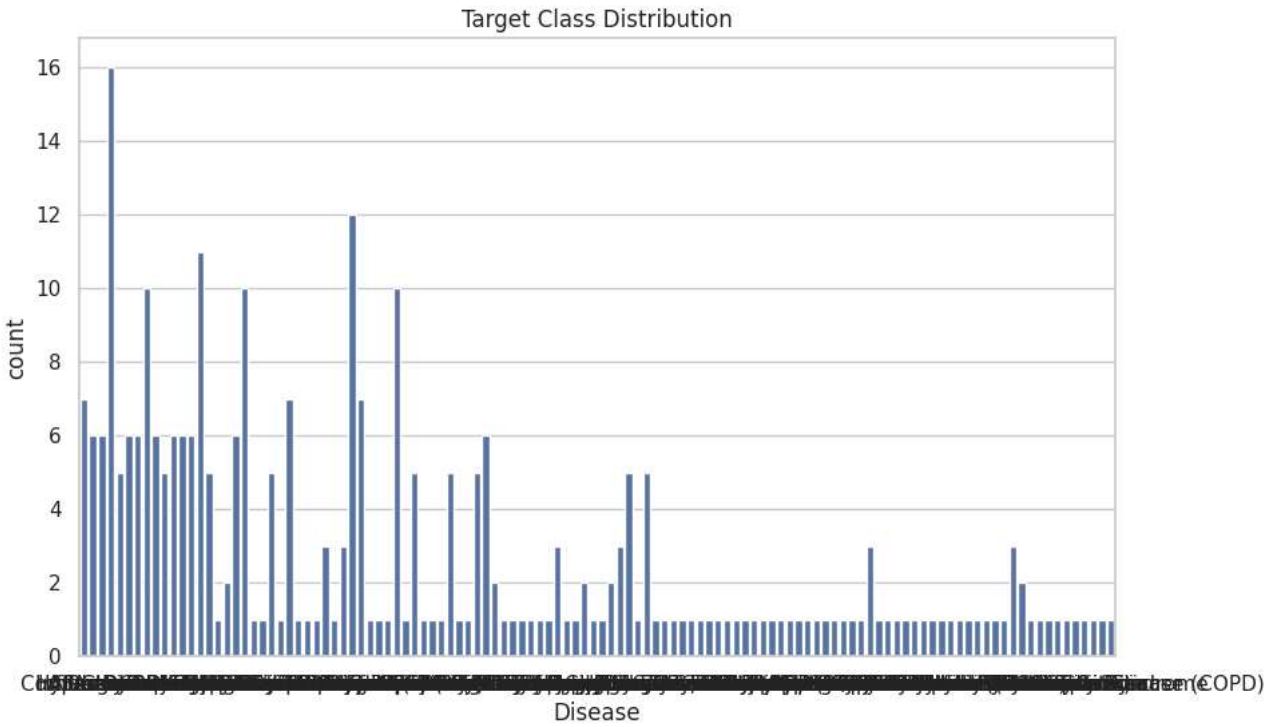
```
# Visualize missing values
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title("Missing Value Heatmap")
plt.show()
```



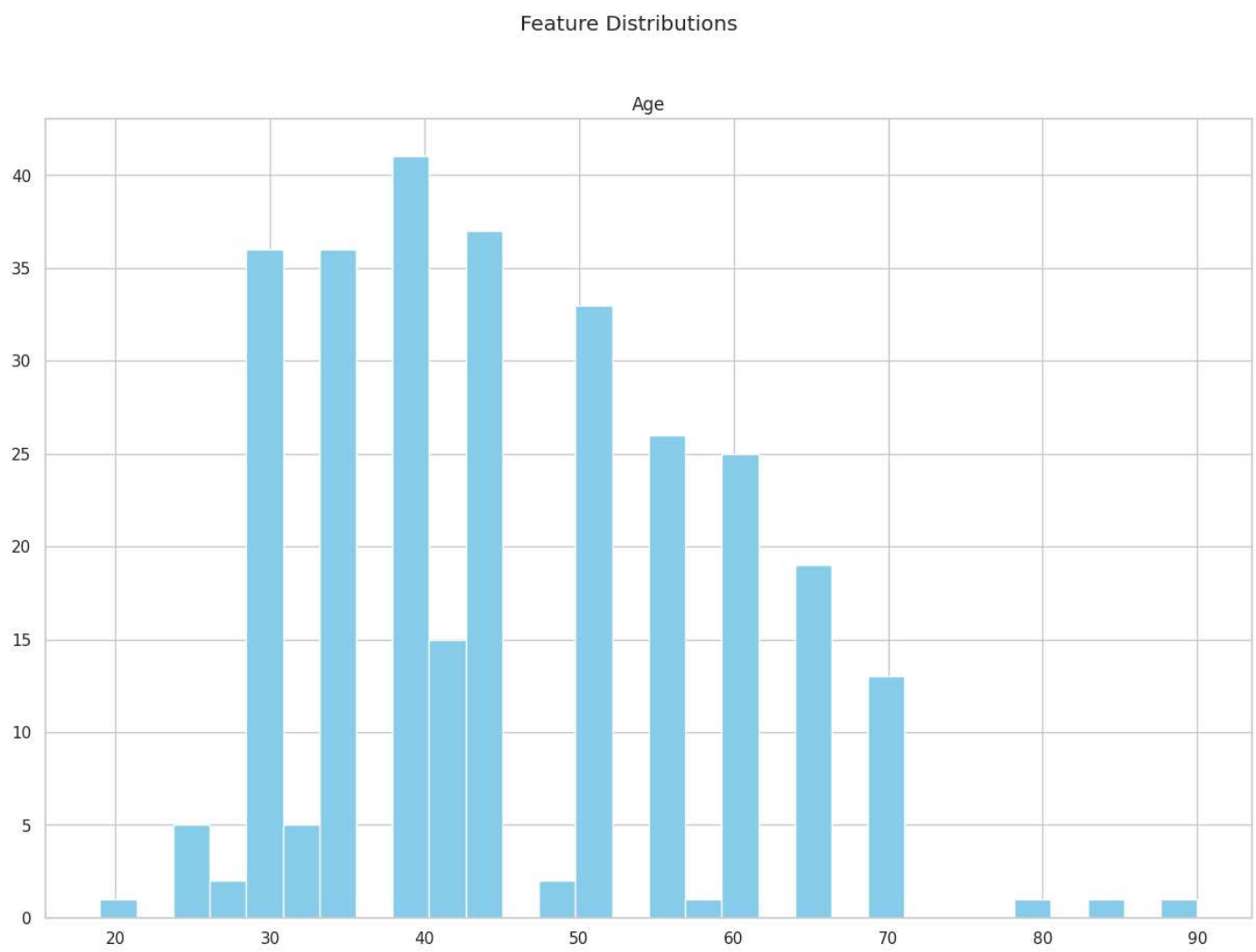

```
# Drop rows with missing values
df = df.dropna()


# Drop duplicates
df = df.drop_duplicates()


# Step 3: Exploratory Data Analysis (EDA)
# ----------------------------
# Distribution of target
if 'Disease' in df.columns:
    sns.countplot(x='Disease', data=df)
    plt.title("Target Class Distribution")
    plt.show()
```




```
# Histograms for numeric features
df.select_dtypes(include=[np.number]).hist(bins=30, figsize=(15, 10), color='skyblue')
plt.suptitle("Feature Distributions")
plt.show()
```

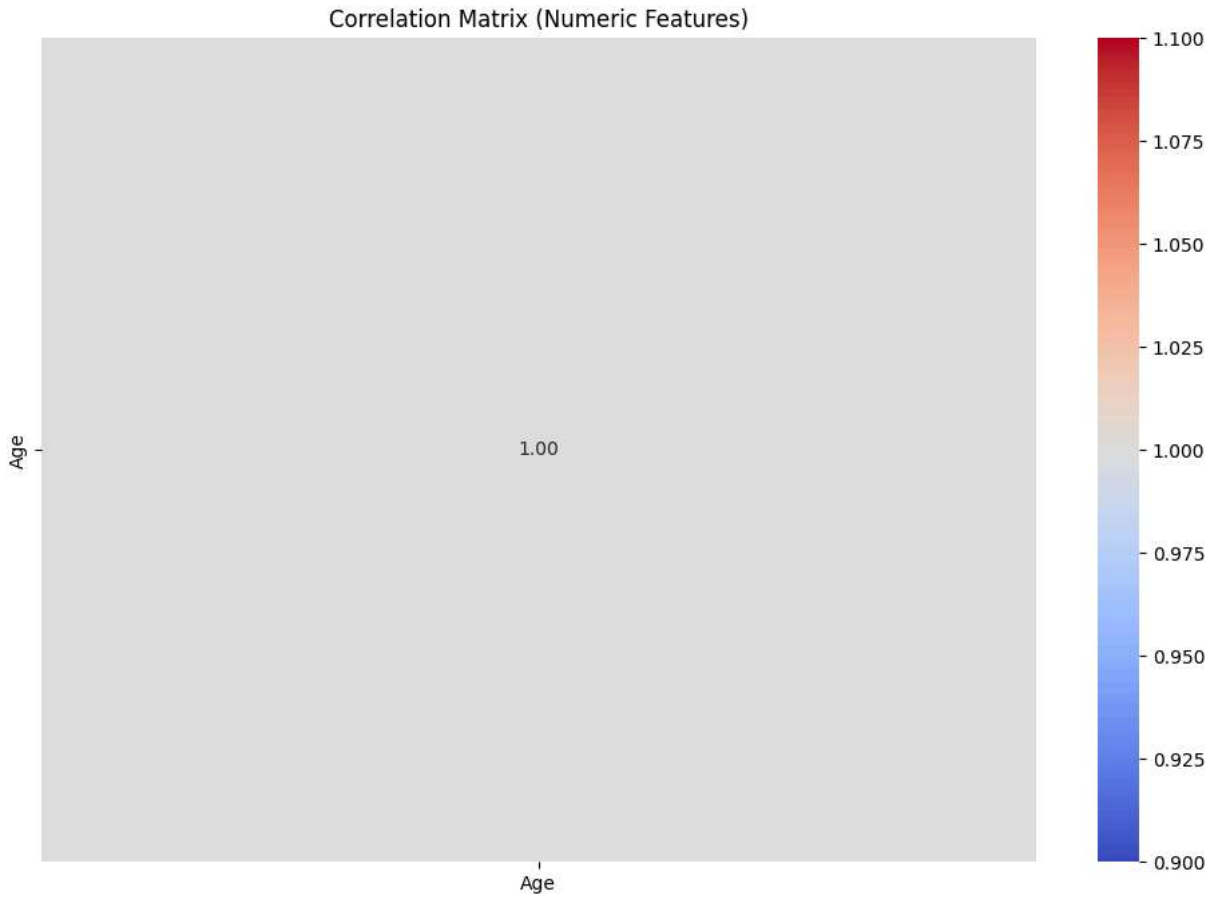



```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt # This line imports the necessary library
import seaborn as sns
import pickle

# Load the dataset (Make sure the path is correct)
df = pd.read_csv("/content/dataset.csv")

# ... (rest of the code) ...

# Correlation heatmap (numeric only to avoid string conversion errors)
plt.figure(figsize=(12, 8)) # Now 'plt' is defined and can be used
numeric_df = df.select_dtypes(include=[np.number])
sns.heatmap(numeric_df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix (Numeric Features)")
plt.show()
```




```
# Step 4: Preprocessing
# ----------------------------
from sklearn.preprocessing import LabelEncoder # Import LabelEncoder here
label_encoders = {}
for col in df.select_dtypes(include='object').columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le


# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler # Import StandardScaler here
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score, roc_curve

if 'Disease' not in df.columns:
    raise ValueError("Target column not found in dataset. Please ensure it's named 'Disease'.")

X = df.drop('Disease', axis=1)
y = df['Disease']

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

# Split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)


# Step 5: Model Training
# -----------------------------
# Logistic Regression
```

```
lr = LogisticRegression()
lr.fit(X_train, y_train)

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

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
# Step 6: Evaluation
# ----------------------------
models = {'Logistic Regression': lr, 'Random Forest': rf}

for name, model in models.items():
    y_pred = model.predict(X_test)
    print(f"\n{name} Results")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred, zero_division=0))

    # ROC AUC (only for binary classification)
    if hasattr(model, "predict_proba") and len(np.unique(y_test)) == 2:
        y_probs = model.predict_proba(X_test)[:, 1]
        fpr, tpr, _ = roc_curve(y_test, y_probs)
        auc_score = roc_auc_score(y_test, y_probs)
        plt.plot(fpr, tpr, label=f"{name} (AUC = {auc_score:.2f})")

# Plot ROC curve only if applicable
if len(np.unique(y_test)) == 2:
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curves")
    plt.legend()
    plt.show()
else:
    print("ROC curve is not applicable for multiclass classification.")
```

```
          91       1.00      1.00      1.00         1
          93       0.00      0.00      0.00         2
          97       0.00      0.00      0.00         1
         100       0.00      0.00      0.00         1
         101       0.00      0.00      0.00         4
         106       0.00      0.00      0.00         1
         107       0.00      0.00      0.00         1
         108       0.00      0.00      0.00         0
         111       0.00      0.00      0.00         3
         112       0.00      0.00      0.00         1
         113       0.00      0.00      0.00         0
         114       0.00      0.00      0.00         0
         115       1.00      1.00      1.00         1

    accuracy                           0.30        70
   macro avg       0.19      0.20      0.19        70
weighted avg       0.30      0.30      0.30        70

ROC curve is not applicable for multiclass classification
```

```
# Step 7: Save the Best Model
# ----------------------------
# Save Random Forest model
with open("heart_model.pkl", "wb") as f:
    pickle.dump(rf, f)

# Save encoders
with open("label_encoders.pkl", "wb") as f:
    pickle.dump(label_encoders, f)

# Save scaler
with open("scaler.pkl", "wb") as f:
    pickle.dump(scaler, f)

print("\nModel, encoders, and scaler saved successfully.")
```

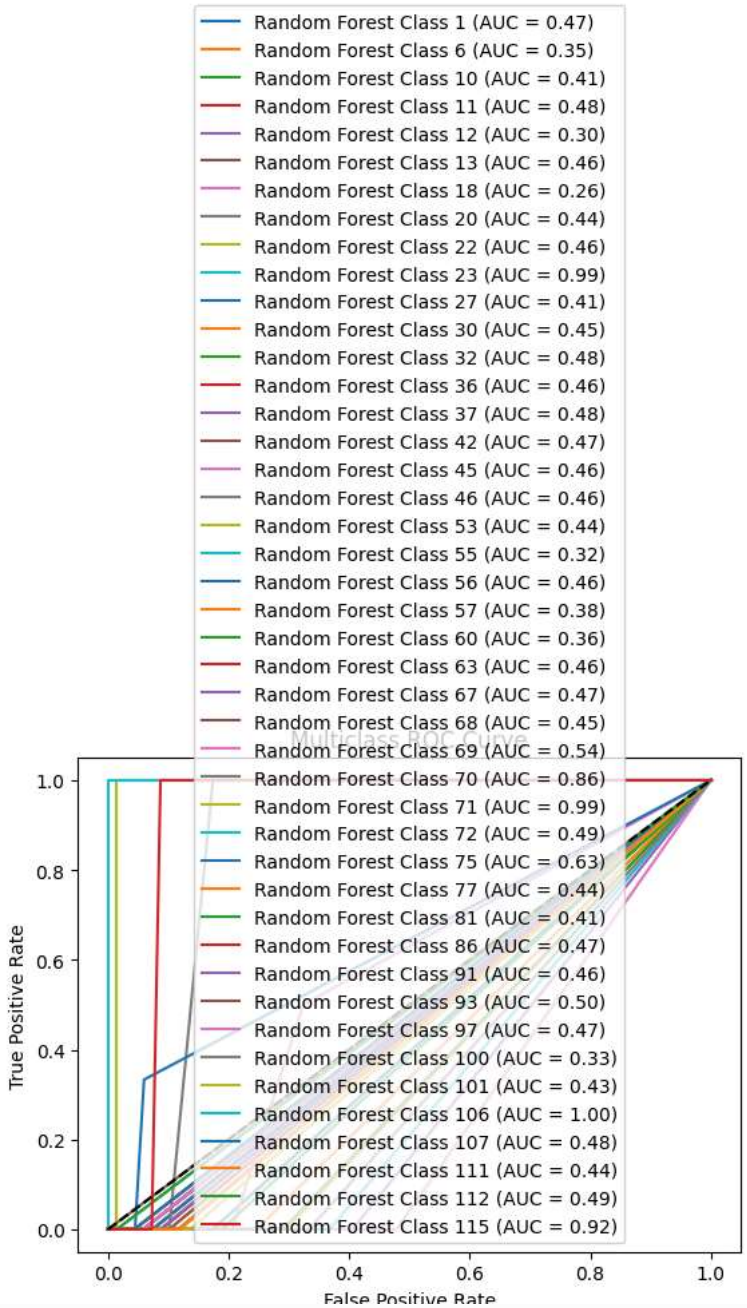
```
Model, encoders, and scaler saved successfully.
```

```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import auc

# Binarize y_test for multiclass ROC
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
y_score = model.predict_proba(X_test)

# Compute ROC curve and AUC for each class
for i in range(len(classes)):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f"{name} Class {classes[i]} (AUC = {roc_auc:.2f})")

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC Curve")
plt.legend()
plt.show()
```




```
import pandas as pd
df = pd.read_csv('/content/dataset.csv')
df.head()
```

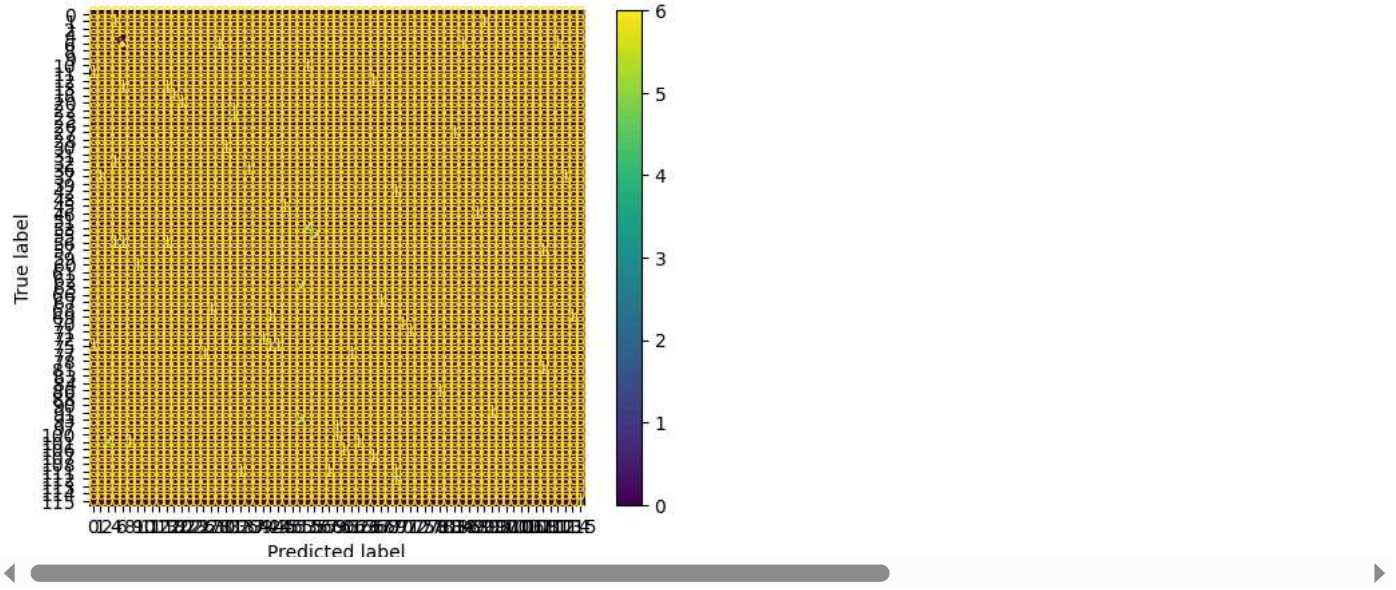
	Disease	Fever	Cough	Fatigue	Difficulty Breathing	Age	Gender	Blood Pressure	Cholesterol Level	Outcome Variable
0	Influenza	Yes	No	Yes	Yes	19	Female	Low	Normal	Positive
1	Common Cold	No	Yes	Yes	No	25	Female	Normal	Normal	Negative
2	Eczema	No	Yes	Yes	No	25	Female	Normal	Normal	Negative
3	Asthma	Yes	Yes	No	Yes	25	Male	Normal	Normal	Positive
4	Asthma	Yes	Yes	No	Yes	25	Male	Normal	Normal	Positive

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_estimator(model, X_test, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f34a5a27c90>
```




```
import joblib
joblib.dump(model, 'xgboost_model.pkl')
```

```
['xgboost_model.pkl']
```

```
!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-packages (from streamlit) (5.5.0)
Requirement already satisfied: blinker<2,>=1.5.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (1.9.0)
Requirement already satisfied: cachetools<6,>=4.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (5.5.2)
Requirement already satisfied: click<9,>=7.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (8.1.8)
Requirement already satisfied: numpy<3,>=1.23 in /usr/local/lib/python3.11/dist-packages (from streamlit) (2.0.2)
Requirement already satisfied: packaging<25,>=20 in /usr/local/lib/python3.11/dist-packages (from streamlit) (24.2)
Requirement already satisfied: pandas<3,>=1.4.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (2.2.2)
Requirement already satisfied: pillow<12,>=7.1.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (11.2.1)
Requirement already satisfied: protobuf<7,>=3.20 in /usr/local/lib/python3.11/dist-packages (from streamlit) (5.29.4)
Requirement already satisfied: pyarrow>=7.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (18.1.0)
Requirement already satisfied: requests<3,>=2.27 in /usr/local/lib/python3.11/dist-packages (from streamlit) (2.32.3)
Requirement already satisfied: tenacity<10,>=8.1.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (9.1.2)
Requirement already satisfied: toml<2,>=0.10.1 in /usr/local/lib/python3.11/dist-packages (from streamlit) (0.10.2)
Requirement already satisfied: typing-extensions<5,>=4.4.0 in /usr/local/lib/python3.11/dist-packages (from streamlit) (4.13.2)
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/python3.11/dist-packages (from streamlit) (3.1.44)
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-packages (from streamlit) (6.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->streamlit) (3.1.6)
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->streamlit) (4.23.0)
Requirement already satisfied: narwhals>=1.14.2 in /usr/local/lib/python3.11/dist-packages (from altair<6,>=4.0->streamlit) (1.37.1)
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.11/dist-packages (from gitpython!=3.1.19,<4,>=3.0.7->streamlit)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->streamlit) (2.9
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->streamlit) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3,>=1.4.0->streamlit) (2025.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->streamlit) (
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->streamlit) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->streamlit) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.27->streamlit) (2025.4
Requirement already satisfied: smmap<6,>=3.0.1 in /usr/local/lib/python3.11/dist-packages (from gitdb<5,>=4.0.1->gitpython!=3.1.19,<4,>=
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->altair<6,>=4.0->streamlit) (3.0.
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->altair<6,>=4.0->streamlit
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->al
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->altair<6,>=4.0->str
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->altair<6,>=4.0->streamli
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas<3,>=1.4.0->strea
```

```
Downloading streamlit-1.45.0-py3-none-any.whl (9.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.9/9.9 MB 48.4 MB/s eta 0:00:00
Downloading pydeck-0.9.1-py2.py3-none-any.whl (6.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 84.1 MB/s eta 0:00:00
Downloading watchdog-6.0.0-py3-none-manylinux2014_x86_64.whl (79 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.1/79.1 kB 8.3 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
```

```
import streamlit as st
import joblib
import numpy as np

model = joblib.load('xgboost_model.pkl')

st.title("AI Disease Prediction")

glucose = st.number_input("Glucose Level")
bmi = st.number_input("BMI")
bp = st.number_input("Blood Pressure")
age = st.slider("Age", 20, 100)

if st.button("Predict"):
    prediction = model.predict([[glucose, bp, bmi, age]])
    st.success(f"Prediction: {'Positive' if prediction[0]==1 else 'Negative'}")
```

```
2025-05-10 09:26:43.370 WARNING streamlit.runtime.scriptrunner_utils.script_run_context: Thread 'MainThread': missing ScriptRunContext!
2025-05-10 09:26:43.554
  Warning: to view this Streamlit app on a browser, run it with the following
  command:

    streamlit run /usr/local/lib/python3.11/dist-packages/colab_kernel_launcher.py [ARGUMENTS]
2025-05-10 09:26:43.556 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.559 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.561 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.564 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.566 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.569 Session state does not function when running a script without `streamlit run`
2025-05-10 09:26:43.571 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.574 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.577 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.580 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.581 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.582 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.583 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.585 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.587 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.590 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.592 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.594 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.597 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.599 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.601 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.602 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.604 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.605 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.607 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.608 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.611 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.612 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.613 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.615 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-05-10 09:26:43.616 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder # Import LabelEncoder
from xgboost import XGBClassifier
import joblib

# Load data
df = pd.read_csv("/content/dataset.csv")  # Change filename if needed

# Define target and features
X = df.drop('Outcome Variable', axis=1)  # Assuming 'Outcome Variable' is the target
```

```
y = df['Outcome Variable']

# ----> Encode categorical features to numerical using LabelEncoder
# ----> Create a LabelEncoder instance for each categorical column
label_encoders = {}
for col in X.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])
    label_encoders[col] = le  # Store the encoder for later use

# 🔁 Step 1: Encode labels to 0, 1, 2, ...
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# 🔢 Get number of classes
num_classes = len(np.unique(y_encoded))

# 🔄 Step 2: Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)  # Now X should only contain numerical features

# 🧪 Step 3: Split data (stratified)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
)

# 🚀 Step 4: Train XGBoost for multiclass
model = XGBClassifier(objective='multi:softmax', num_class=num_classes, random_state=42)
model.fit(X_train, y_train)

# 💾 Step 5: Save model and label encoder
joblib.dump(model, 'xgboost_model.pkl')
joblib.dump(le, 'label_encoder.pkl')
# ----> Save the label encoders for the categorical features
joblib.dump(label_encoders, 'feature_label_encoders.pkl')

print("✅ Model trained and saved successfully.")
```

```
✅ Model trained and saved successfully.
```

```
!pip install streamlit
!pip install xgboost
!pip install scikit-learn
!pip install joblib
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from xgboost import XGBClassifier
import joblib
import streamlit as st

# ----> Training and saving the model (Run this cell once) <----

# Load data
df = pd.read_csv("/content/dataset.csv")  # Update filename if needed

# Define target and features
X = df.drop('Outcome Variable', axis=1)
y = df['Outcome Variable']

# Encode categorical features to numerical using LabelEncoder
label_encoders = {}
for col in X.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])
    label_encoders[col] = le  # Store the encoder for later use

# Encode target variable
le = LabelEncoder()
y_encoded = le.fit_transform(y)
num_classes = len(np.unique(y_encoded))

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

```
# Split data (stratified)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
)

# Train XGBoost
model = XGBClassifier(objective='multi:softmax', num_class=num_classes, random_state=42)
model.fit(X_train, y_train)

# Save model, encoders, and scaler
joblib.dump(model, 'xgboost_model.pkl')
joblib.dump(le, 'label_encoder.pkl')
joblib.dump(label_encoders, 'feature_label_encoders.pkl')
joblib.dump(scaler, 'scaler.pkl') # Added this to save scaler


print("✅ Model trained and saved successfully.")

# ----> Streamlit App (Run this cell to start the app) <----

# Load trained components
model = joblib.load('xgboost_model.pkl')
target_encoder = joblib.load('label_encoder.pkl')
feature_encoders = joblib.load('feature_label_encoders.pkl')
scaler = joblib.load('scaler.pkl')  # Load the saved scaler

# Streamlit UI
st.title("AI-Powered Disease Prediction")
st.markdown("Enter patient details to predict possible disease classification.")

# Example inputs - Update based on your dataset
fever = st.selectbox("Fever", ["Yes", "No"])
cough = st.selectbox("Cough", ["Yes", "No"])
fatigue = st.selectbox("Fatigue", ["Yes", "No"])
breathing = st.selectbox("Difficulty Breathing", ["Yes", "No"])
age = st.slider("Age", 0, 120)
gender = st.selectbox("Gender", ["Male", "Female"])
bp = st.number_input("Blood Pressure", min_value=0)
chol = st.number_input("Cholesterol Level", min_value=0)

# Predict button
if st.button("Predict Disease"):
    input_dict = {
        'Fever': fever,
        'Cough': cough,
        'Fatigue': fatigue,
        'Difficulty Breathing': breathing,
        'Age': age,
        'Gender': gender,
        'Blood Pressure': bp,
        'Cholesterol Level': chol
    }

    input_df = pd.DataFrame([input_dict])

    # Encode categorical features
    for col in input_df.columns:
        if col in feature_encoders:
            input_df[col] = feature_encoders[col].transform(input_df[col])

    # Scale the input features
    input_scaled = scaler.transform(input_df)

    # Predict
    prediction = model.predict(input_scaled)
    predicted_disease = target_encoder.inverse_transform(prediction)[0]

    st.success(f"Predicted Disease: {predicted_disease}")
```