```
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 classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

# Load the dataset
df = pd.read_csv("EQ.csv")

def create_three_class_earthquake_target(df):
    """
    Create simplified 3-class earthquake classification target based on
magnitude.

    Classification scheme:
    - Low: mag < 5.0 (Minor to light earthquakes)
    - Moderate: 5.0 <= mag < 7.0 (Moderate to strong earthquakes)
    - High: mag >= 7.0 (Major to great earthquakes)

    Parameters:
    df (pandas.DataFrame): DataFrame with 'mag' column

    Returns:
    pandas.Series: Classification target column
    """

    # Initialize the classification array
    classification = np.full(len(df), 'Unknown', dtype=object)

    # Handle missing values
    valid_mask = ~df['mag'].isna()
```

```python
    valid_df = df[valid_mask]
    valid_indices = df.index[valid_mask]

    # Low magnitude earthquakes (< 4.0)
    low_mask = valid_df['mag'] <= 4.25
    classification[valid_indices[low_mask]] = 'Low'

    # Moderate magnitude earthquakes (5.0 <= mag < 7.0)
    moderate_mask = (valid_df['mag'] > 4.25)
    classification[valid_indices[moderate_mask]] = 'Moderate'

    return pd.Series(classification, index=df.index,
name='earthquake_class')
```

```python
def preprocess_time_features(df, time_column='time'):
    """
    Convert time column to useful numerical features.

    Parameters:
    df (pandas.DataFrame): DataFrame with time column
    time_column (str): Name of the time column

    Returns:
    pandas.DataFrame: DataFrame with time features
    """

    df_copy = df.copy()

    # Convert to datetime if it's not already
    df_copy[time_column] = pd.to_datetime(df_copy[time_column])

    # Extract time features
    df_copy['year'] = df_copy[time_column].dt.year
    df_copy['month'] = df_copy[time_column].dt.month
    df_copy['day'] = df_copy[time_column].dt.day
    df_copy['hour'] = df_copy[time_column].dt.hour
    df_copy['day_of_year'] = df_copy[time_column].dt.dayofyear
    df_copy['day_of_week'] = df_copy[time_column].dt.dayofweek
```

```python
    # Create cyclical features for temporal patterns
    df_copy['month_sin'] = np.sin(2 * np.pi * df_copy['month'] / 12)
    df_copy['month_cos'] = np.cos(2 * np.pi * df_copy['month'] / 12)
    df_copy['hour_sin'] = np.sin(2 * np.pi * df_copy['hour'] / 24)
    df_copy['hour_cos'] = np.cos(2 * np.pi * df_copy['hour'] / 24)
    df_copy['day_of_week_sin'] = np.sin(2 * np.pi * df_copy['day_of_week']
/ 7)
    df_copy['day_of_week_cos'] = np.cos(2 * np.pi * df_copy['day_of_week']
/ 7)

    return df_copy


def prepare_earthquake_data(df):
    """
    Prepare earthquake data for machine learning.

    Parameters:
    df (pandas.DataFrame): Raw earthquake DataFrame

    Returns:
    tuple: (X_train, X_test, y_train, y_test, feature_names,
label_encoder, scaler)
    """

    print("Preparing earthquake data...")
    print(f"Original dataset shape: {df.shape}")

    # Create target variable
    df['earthquake_class'] = create_three_class_earthquake_target(df)

    # Remove rows with unknown classification
    df_clean = df[df['earthquake_class'] != 'Unknown'].copy()
    print(f"After removing unknown classifications: {df_clean.shape}")

    # Preprocess time features
    df_processed = preprocess_time_features(df_clean, 'time')

    # Select feature columns
```

```
    feature_columns = [
        'latitude', 'longitude', 'mag' , 'depth',
        'year', 'month', 'day', 'hour', 'day_of_year', 'day_of_week',
        'month_sin', 'month_cos', 'hour_sin', 'hour_cos',
        'day_of_week_sin', 'day_of_week_cos'
    ]

    # Check if all required columns exist
    missing_cols = [col for col in feature_columns if col not in
df_processed.columns]
    if missing_cols:
        print(f"Warning: Missing columns {missing_cols}")
        feature_columns = [col for col in feature_columns if col in
df_processed.columns]

    # Remove rows with missing values in selected features
    df_final = df_processed[feature_columns +
['earthquake_class']].dropna()
    print(f"After removing missing values: {df_final.shape}")

    # Prepare features and target
    X = df_final[feature_columns]
    y = df_final['earthquake_class']

    # Encode target labels
    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(y)

    # Split the data
    X_train, X_test, y_train, y_test = train_test_split(
        X, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
    )

    # Normalize features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    print(f"Training set shape: {X_train_scaled.shape}")
    print(f"Test set shape: {X_test_scaled.shape}")
```

```
    print(f"Class distribution in training set:")
    unique, counts = np.unique(y_train, return_counts=True)
    for i, (class_idx, count) in enumerate(zip(unique, counts)):
        class_name = label_encoder.inverse_transform([class_idx])[0]
        print(f"  {class_name}: {count} ({count/len(y_train)*100:.1f}%)")

    return X_train_scaled, X_test_scaled, y_train, y_test,
feature_columns, label_encoder, scaler
```

```
def create_tensorflow_model(input_dim, num_classes):
    """
    Create a TensorFlow neural network model for earthquake
classification.

    Parameters:
    input_dim (int): Number of input features
    num_classes (int): Number of output classes

    Returns:
    tensorflow.keras.Model: Compiled model
    """

    model = tf.keras.Sequential([
        tf.keras.layers.Dense(1, activation='relu',
input_shape=(input_dim,)),
        tf.keras.layers.Dense(num_classes, activation='softmax')
    ])

    model.compile(
        optimizer='adam',
        loss=tf.keras.losses.SparseCategoricalCrossentropy(),
        metrics=['accuracy']
    )

    return model
```

```
X_train, X_test, y_train, y_test, feature_names, label_encoder, scaler =
prepare_earthquake_data(df)
```

```
X_train.shape
```

```
num_classes = len(np.unique(y_train))
model = create_tensorflow_model(X_train.shape[1], num_classes)
```

```
num_classes
```

```
X_train.shape
```

```
print(f"\nModel architecture:")
```

```
# model.summary()

# Train model
print("\nTraining model...")
history = model.fit(
    X_train, y_train,
    epochs=10,
    batch_size=32,
    validation_data=(X_test, y_test)
)
```

```
Model architecture:

Training model...
Epoch 1/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.5561
- loss: 0.6899 - val_accuracy: 0.5726 - val_loss: 0.6826
Epoch 2/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5956
- loss: 0.6739 - val_accuracy: 0.6076 - val_loss: 0.6699
Epoch 3/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.6246
- loss: 0.6623 - val_accuracy: 0.6436 - val_loss: 0.6481
Epoch 4/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.6855
- loss: 0.6361 - val_accuracy: 0.7297 - val_loss: 0.6045
Epoch 5/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7428
- loss: 0.5920 - val_accuracy: 0.8258 - val_loss: 0.5315
Epoch 6/10
```

```
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8391
- loss: 0.5175 - val_accuracy: 0.8919 - val_loss: 0.4547
Epoch 7/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8979
- loss: 0.4412 - val_accuracy: 0.9259 - val_loss: 0.3914
Epoch 8/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9236
- loss: 0.3732 - val_accuracy: 0.9369 - val_loss: 0.3428
Epoch 9/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9377
- loss: 0.3259 - val_accuracy: 0.9469 - val_loss: 0.3025
Epoch 10/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9538
- loss: 0.2870 - val_accuracy: 0.9550 - val_loss: 0.2685
```

```
print("\nEvaluating model...")
train_loss, train_accuracy = model.evaluate(X_train, y_train, verbose=0)
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)

print(f"Training Accuracy: {train_accuracy:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")

# Make predictions
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

# Classification report
print("\nClassification Report:")
class_names = label_encoder.classes_
print(classification_report(y_test, y_pred_classes,
target_names=class_names))

# Confusion matrix
plt.figure(figsize=(12, 5))

# Plot training history
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
```

```
plt.legend()

# Plot confusion matrix
plt.subplot(1, 2, 2)
cm = confusion_matrix(y_test, y_pred_classes)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')

plt.tight_layout()
plt.show()
```