Section 1: Data Loading and Preprocessing

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import MinMaxScaler


# Load the dataset

# Assuming 'Rainfall_Data_LL.csv' is uploaded to the Colab environment.

df = pd.read_csv('/content/Rainfall_Data_LL.csv')


# Drop irrelevant columns and handle potential missing values

# Note: The provided snippet is clean, but in a real-world scenario, this is crucial.

# Fill NaN with 0 or an appropriate value.

df.fillna(0, inplace=True)


# Define IMD rainfall categories

# Heavy rainfall is >= 64.5 mm (based on the provided IMD definition)

def classify_rainfall(rainfall):

    if rainfall >= 204.5:

        return 'Extremely Heavy'

    elif rainfall >= 115.6:

        return 'Very Heavy'

    elif rainfall >= 64.5:

        return 'Heavy'

    else:

        return 'Normal'


# Create a time-series dataset

# This assumes we are using monthly rainfall data

# for a single subdivision to simplify the example.
# We will use all subdivisions and their lat/long as features.
features = [] # Placeholder: You need to define your features here, e.g., ['SUBDIVISION', 'LAT', 'LON',
'JAN', 'FEB', ...]
target = 'ANNUAL' # For demonstration, we use annual to predict a class

# For the purpose of time-series classification, we must create a sequence of inputs.
# We'll create a simple sliding window approach.
# A full implementation would be more complex, but this demonstrates the principle.
def create_sequences(data, n_past, n_future):
  X, y = [], []
  for i in range(n_past, len(data) - n_future + 1):
    X.append(data[i - n_past:i, :])
    y.append(data[i:i + n_future, 0])  # Assuming rainfall is the first feature
  return np.array(X), np.array(y)

# We'll simplify for a proof of concept with the provided data.
# Let's frame the problem as predicting next month's category
# based on the past 3 months and lat/long.
df_processed = df.copy()
df_processed['JUN'] = df_processed['JUN'].apply(classify_rainfall)
# df_processed = df_processed.drop(columns=[]) # Placeholder: Specify columns to drop if any

# Convert string categories to numerical labels
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df_processed['JUN'] = le.fit_transform(df_processed['JUN'])

# Using a simpler feature set for demonstration
# Changed features_for_model to use existing monthly rainfall columns as features
features_for_model = ['JAN', 'FEB', 'MAR', 'APR', 'MAY']
X = df_processed[features_for_model].values
```

```
y = df_processed['JUN'].values # Assuming 'JUN' is the target for this simplified example


# Scale the data

scaler = MinMaxScaler()

X_scaled = scaler.fit_transform(X)


# Split the data

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


# Reshape for LSTM: [samples, timesteps, features]

# For this example, we treat each row as a single timestep

X_train_reshaped = X_train[:, np.newaxis, :]

X_test_reshaped = X_test[:, np.newaxis, :]
```

Section 2: Model Building and Training

Python

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Conv1D, MaxPooling1D, Flatten, Dropout
from tensorflow.keras.utils import to_categorical
from xgboost import XGBClassifier

# One-hot encode the target variable for deep learning models
y_train_encoded = to_categorical(y_train)
y_test_encoded = to_categorical(y_test)
num_classes = len(le.classes_)

# Build the Hybrid CNN-LSTM Model
# Using a 1D Conv layer since our time series data is 1-dimensional
hybrid_model = Sequential()
hybrid_model.add(Conv1D(filters=64, kernel_size=1, activation='relu',
input_shape=(X_train_reshaped.shape[1], X_train_reshaped.shape[2])))
hybrid_model.add(MaxPooling1D(pool_size=1))
hybrid_model.add(Flatten())
hybrid_model.add(Dropout(0.2))
hybrid_model.add(Dense(100, activation='relu'))
hybrid_model.add(Dropout(0.2))
hybrid_model.add(Dense(50, activation='relu'))
hybrid_model.add(Dense(num_classes, activation='softmax'))

hybrid_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
hybrid_model.fit(X_train_reshaped, y_train_encoded, epochs=50, batch_size=32,
validation_split=0.1, verbose=0)
hybrid_model_predictions = hybrid_model.predict(X_test_reshaped)
hybrid_model_predictions_classes = np.argmax(hybrid_model_predictions, axis=1)

# Build the Standalone LSTM Model
lstm_model = Sequential()
lstm_model.add(LSTM(100, activation='relu', input_shape=(X_train_reshaped.shape[1],
X_train_reshaped.shape[2])))
lstm_model.add(Dense(50, activation='relu'))
lstm_model.add(Dense(num_classes, activation='softmax'))

lstm_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
lstm_model.fit(X_train_reshaped, y_train_encoded, epochs=50, batch_size=32, validation_split=0.1,
verbose=0)
lstm_model_predictions = lstm_model.predict(X_test_reshaped)
lstm_model_predictions_classes = np.argmax(lstm_model_predictions, axis=1)

# Train the XGBoost Model
xgb_model = XGBClassifier(objective='multi:softprob', n_estimators=100, use_label_encoder=False,
eval_metric='mlogloss')
xgb_model.fit(X_train, y_train)
xgb_model_predictions = xgb_model.predict(X_test)
```

Section 3: Model Evaluation

Python

```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, classification_report

def calculate_threat_score(y_true, y_pred, pos_label=2): # Assuming 'Heavy' is label 2
    cm = confusion_matrix(y_true, y_pred)
    # N11: hits (correctly predicted positive)
    N11 = cm[pos_label, pos_label]
    # N10: misses (actual positive, predicted negative)
    N10 = sum(cm[:, pos_label]) - N11
    # N01: false alarms (actual negative, predicted positive)
    N01 = sum(cm[pos_label, :]) - N11

    # Threat score formula: N11 / (N11 + N10 + N01)
    if (N11 + N10 + N01) == 0:
        return 0
    ts = N11 / (N11 + N10 + N01)
    return ts

print("Hybrid CNN-LSTM Model Performance:")
print(f"Accuracy: {accuracy_score(y_test, hybrid_model_predictions_classes):.4f}")
print(f"F1-Score: {f1_score(y_test, hybrid_model_predictions_classes, average='weighted'):.4f}")
print(f"Threat Score: {calculate_threat_score(y_test, hybrid_model_predictions_classes,
pos_label=le.transform(['Heavy'])):.4f}")
print("\n" + classification_report(y_test, hybrid_model_predictions_classes,
target_names=le.classes_))

print("Standalone LSTM Model Performance:")
print(f"Accuracy: {accuracy_score(y_test, lstm_model_predictions_classes):.4f}")
print(f"F1-Score: {f1_score(y_test, lstm_model_predictions_classes, average='weighted'):.4f}")
print(f"Threat Score: {calculate_threat_score(y_test, lstm_model_predictions_classes,
pos_label=le.transform(['Heavy'])):.4f}")
print("\n" + classification_report(y_test, lstm_model_predictions_classes,
target_names=le.classes_))

print("XGBoost Model Performance:")
print(f"Accuracy: {accuracy_score(y_test, xgb_model_predictions):.4f}")
print(f"F1-Score: {f1_score(y_test, xgb_model_predictions, average='weighted'):.4f}")
print(f"Threat Score: {calculate_threat_score(y_test, xgb_model_predictions,
pos_label=le.transform(['Heavy'])):.4f}")
print("\n" + classification_report(y_test, xgb_model_predictions, target_names=le.classes_))
```

Section 4: Explainable AI

Python

```
import shap
import matplotlib.pyplot as plt

# Explain the XGBoost Model with SHAP
explainer_xgb = shap.TreeExplainer(xgb_model)
shap_values_xgb = explainer_xgb.shap_values(X_test)
shap.summary_plot(shap_values_xgb, X_test, feature_names=features_for_model, plot_type='bar')
plt.show()
```

```
# Explain the Hybrid Model with SHAP
# This is a complex example, but DeepExplainer is the right tool
explainer_hybrid = shap.DeepExplainer(hybrid_model, X_train_reshaped)
shap_values_hybrid = explainer_hybrid.shap_values(X_test_reshaped)
# For multi-class, shap_values is a list. We plot for the 'Heavy' class.
heavy_class_index = le.transform(['Heavy'])
shap.summary_plot(shap_values_hybrid[heavy_class_index], X_test_reshaped.squeeze(),
feature_names=features_for_model, plot_type='beeswarm')
plt.show()

# Explain an individual prediction with SHAP
# Get the first instance from the test set
sample_instance = X_test_reshaped
shap_values_instance = explainer_hybrid.shap_values(sample_instance.reshape(1, 1, -1))
shap.initjs()
# Note: force_plot for DeepExplainer can be complex to visualize
# A beeswarm plot is more practical for a report
print("SHAP values for a single instance prediction (Hybrid Model):")
print(f"Predicted Class: {le.inverse_transform([hybrid_model_predictions_classes])}")
# shap.force_plot(explainer_hybrid.expected_value[heavy_class_index],
shap_values_instance[heavy_class_index], sample_instance.squeeze(),
feature_names=features_for_model)

# Demonstrate LIME on a single instance for the hybrid model
# Note: LIME is more complex for deep learning and requires a wrapper
from lime.lime_tabular import LimeTabularExplainer

explainer_lime = LimeTabularExplainer(
   X_train,
   class_names=le.classes_,
   feature_names=features_for_model,
   discretize_continuous=True
)

# Function to get hybrid model's prediction for LIME
def predict_fn(data):
   return hybrid_model.predict(data[:, np.newaxis, :])

# Explain a random instance
idx = np.random.randint(0, len(X_test))
exp = explainer_lime.explain_instance(X_test[idx], predict_fn, num_features=5, top_labels=1)
print(f"\nLIME explanation for instance {idx}:")
print(f"Predicted class:
{le.inverse_transform([np.argmax(hybrid_model.predict(X_test_reshaped[idx][np.newaxis, :, :]))])}")
print(f"Actual class: {le.inverse_transform([y_test[idx]])}")
exp.show_in_notebook(show_all=False)
```