miniproject6th

April 10, 2025

[ ]:
```python
# Data Handling
import pandas as pd
import numpy as np

# Plotting
import matplotlib.pyplot as plt
import seaborn as sns

# Machine Learning
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Deep Learning
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# For RL Simulation
import gym
```

[ ]:
```
!pip install tensorflow
```

[ ]:
```
conda install tensorflow
```

[ ]:
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

[ ]:
```python
data = pd.read_csv("rajasthan_weather_data.csv")  # Load real-time weather␣
↪dataset
data.head()
```

[9]:
```
!pip install pandas numpy matplotlib seaborn scikit-learn tensorflow
```

```
Requirement already satisfied: pandas in c:\users\kiit0001\anaconda3\lib\site-
packages (2.2.2)
Requirement already satisfied: numpy in c:\users\kiit0001\anaconda3\lib\site-
packages (1.26.4)
Requirement already satisfied: matplotlib in
```

```
c:\users\kiit0001\anaconda3\lib\site-packages (3.9.2)
Requirement already satisfied: seaborn in c:\users\kiit0001\anaconda3\lib\site-
packages (0.13.2)
Requirement already satisfied: scikit-learn in
c:\users\kiit0001\anaconda3\lib\site-packages (1.5.1)
Requirement already satisfied: tensorflow in
c:\users\kiit0001\anaconda3\lib\site-packages (2.19.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\kiit0001\anaconda3\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
c:\users\kiit0001\anaconda3\lib\site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in
c:\users\kiit0001\anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\kiit0001\anaconda3\lib\site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in
c:\users\kiit0001\anaconda3\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
c:\users\kiit0001\anaconda3\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: packaging>=20.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=8 in
c:\users\kiit0001\anaconda3\lib\site-packages (from matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\kiit0001\anaconda3\lib\site-packages (from matplotlib) (3.1.2)
Requirement already satisfied: scipy>=1.6.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: absl-py>=1.0.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (2.2.2)
Requirement already satisfied: astunparse>=1.6.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (3.4.0)
Requirement already satisfied:
```

```
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3
in c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (4.25.3)
Requirement already satisfied: requests<3,>=2.21.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (3.0.1)
Requirement already satisfied: typing-extensions>=3.6.6 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (4.11.0)
Requirement already satisfied: wrapt>=1.11.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (1.14.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (1.71.0)
Requirement already satisfied: tensorboard~=2.19.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (2.19.0)
Requirement already satisfied: keras>=3.5.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (3.9.2)
Requirement already satisfied: h5py>=3.11.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (3.11.0)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in
c:\users\kiit0001\anaconda3\lib\site-packages (from tensorflow) (0.5.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from
astunparse>=1.6.0->tensorflow) (0.44.0)
Requirement already satisfied: rich in c:\users\kiit0001\anaconda3\lib\site-
packages (from keras>=3.5.0->tensorflow) (13.7.1)
Requirement already satisfied: namex in c:\users\kiit0001\anaconda3\lib\site-
packages (from keras>=3.5.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in c:\users\kiit0001\anaconda3\lib\site-
packages (from keras>=3.5.0->tensorflow) (0.15.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
c:\users\kiit0001\anaconda3\lib\site-packages (from
requests<3,>=2.21.0->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
c:\users\kiit0001\anaconda3\lib\site-packages (from
requests<3,>=2.21.0->tensorflow) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
c:\users\kiit0001\anaconda3\lib\site-packages (from
requests<3,>=2.21.0->tensorflow) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
c:\users\kiit0001\anaconda3\lib\site-packages (from
requests<3,>=2.21.0->tensorflow) (2024.8.30)
Requirement already satisfied: markdown>=2.6.8 in
c:\users\kiit0001\anaconda3\lib\site-packages (from
tensorboard~=2.19.0->tensorflow) (3.4.1)
```

```
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from
tensorboard~=2.19.0->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
c:\users\kiit0001\anaconda3\lib\site-packages (from
tensorboard~=2.19.0->tensorflow) (3.0.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in
c:\users\kiit0001\anaconda3\lib\site-packages (from
werkzeug>=1.0.1->tensorboard~=2.19.0->tensorflow) (2.1.3)
Requirement already satisfied: markdown-it-py>=2.2.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from
rich->keras>=3.5.0->tensorflow) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
c:\users\kiit0001\anaconda3\lib\site-packages (from
rich->keras>=3.5.0->tensorflow) (2.15.1)
Requirement already satisfied: mdurl~=0.1 in
c:\users\kiit0001\anaconda3\lib\site-packages (from markdown-it-
py>=2.2.0->rich->keras>=3.5.0->tensorflow) (0.1.0)
```

[11]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

[12]:

```python
# Example: Load climate dataset (replace the link with actual data from␣
↪Rajasthan)
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/
↪daily-min-temperatures.csv"
data = pd.read_csv(url)

# Display first few rows
data.head()
```

[12]:

	Date	Temp
0	1981-01-01	20.7
1	1981-01-02	17.9
2	1981-01-03	18.8
3	1981-01-04	14.6
4	1981-01-05	15.8

[14]:
```python
import pandas as pd
import numpy as np

#  date range
dates = pd.date_range(start='2023-01-01', periods=365, freq='D')

#  weather data for Rajasthan
np.random.seed(42)
temperature = np.random.normal(loc=35, scale=5, size=365)           # in °C
humidity = np.random.uniform(low=10, high=60, size=365)             # in %
wind_speed = np.random.uniform(low=0, high=30, size=365)            # in km/
↪h
air_pressure = np.random.normal(loc=1010, scale=10, size=365)      # in hPa
soil_moisture = np.random.uniform(low=5, high=30, size=365)        # in %
cloud_coverage = np.random.uniform(low=0, high=100, size=365)      # in %
rain = np.random.binomial(n=1, p=0.2, size=365)                     # 0 or␣
↪1, 20% chance of rain

# Create DataFrame
data = pd.DataFrame({
    'date': dates,
    'temperature': temperature,
    'humidity': humidity,
    'wind_speed': wind_speed,
    'air_pressure': air_pressure,
    'soil_moisture': soil_moisture,
    'cloud_coverage': cloud_coverage,
    'rain': rain
})

# Save to CSV
data.to_csv('synthetic_rajasthan_weather_2023.csv', index=False)

print("  Dataset saved successfully as 'synthetic_rajasthan_weather_2023.csv'")
```

```
 Dataset saved successfully as 'synthetic_rajasthan_weather_2023.csv'
```

[15]:
```python
import pandas as pd

# Replace with your actual file name
data = pd.read_csv("synthetic_rajasthan_weather_2023.csv")

# Show first few rows
data.head()
```

[15]:
```
         date  temperature   humidity  wind_speed  air_pressure  \
0  2023-01-01    37.483571  21.179792    5.291610   1005.987795
```

```
1  2023-01-02    34.308678  58.161127   14.951033   1004.410782
2  2023-01-03    38.238443  10.607724   12.567763   1013.772119
3  2023-01-04    42.615149  58.493941   27.445377   1025.655240
4  2023-01-05    33.829233  12.157996   10.871817   1009.342497

   soil_moisture  cloud_coverage  rain
0      20.014858       26.015779     0
1      17.891986       73.082096     0
2      27.984799       98.129709     0
3      17.424087       25.653006     0
4      29.803950       65.417460     1
```

[19]:
```python
import pandas as pd
import numpy as np

# Generate date range for 1 year
dates = pd.date_range(start='2023-01-01', periods=365, freq='D')

# Generate synthetic weather data for Rajasthan
np.random.seed(42)
temperature = np.random.normal(loc=35, scale=5, size=365)          # in °C
humidity = np.random.uniform(low=10, high=60, size=365)            # in %
wind_speed = np.random.uniform(low=0, high=30, size=365)           # in km/
 ↪h
air_pressure = np.random.normal(loc=1010, scale=10, size=365)     # in hPa
soil_moisture = np.random.uniform(low=5, high=30, size=365)       # in %
cloud_coverage = np.random.uniform(low=0, high=100, size=365)     # in %
rain = np.random.binomial(n=1, p=0.2, size=365)                    # 0 =␣
 ↪No rain, 1 = Rain

# Create a DataFrame
data = pd.DataFrame({
    'date': dates,
    'temperature': temperature,
    'humidity': humidity,
    'wind_speed': wind_speed,
    'air_pressure': air_pressure,
    'soil_moisture': soil_moisture,
    'cloud_coverage': cloud_coverage,
    'rain': rain
})
```

[20]:
```python
data.to_csv('synthetic_rajasthan_weather_2023.csv', index=False)
print("  CSV file saved successfully.")
```

```
 CSV file saved successfully.
```

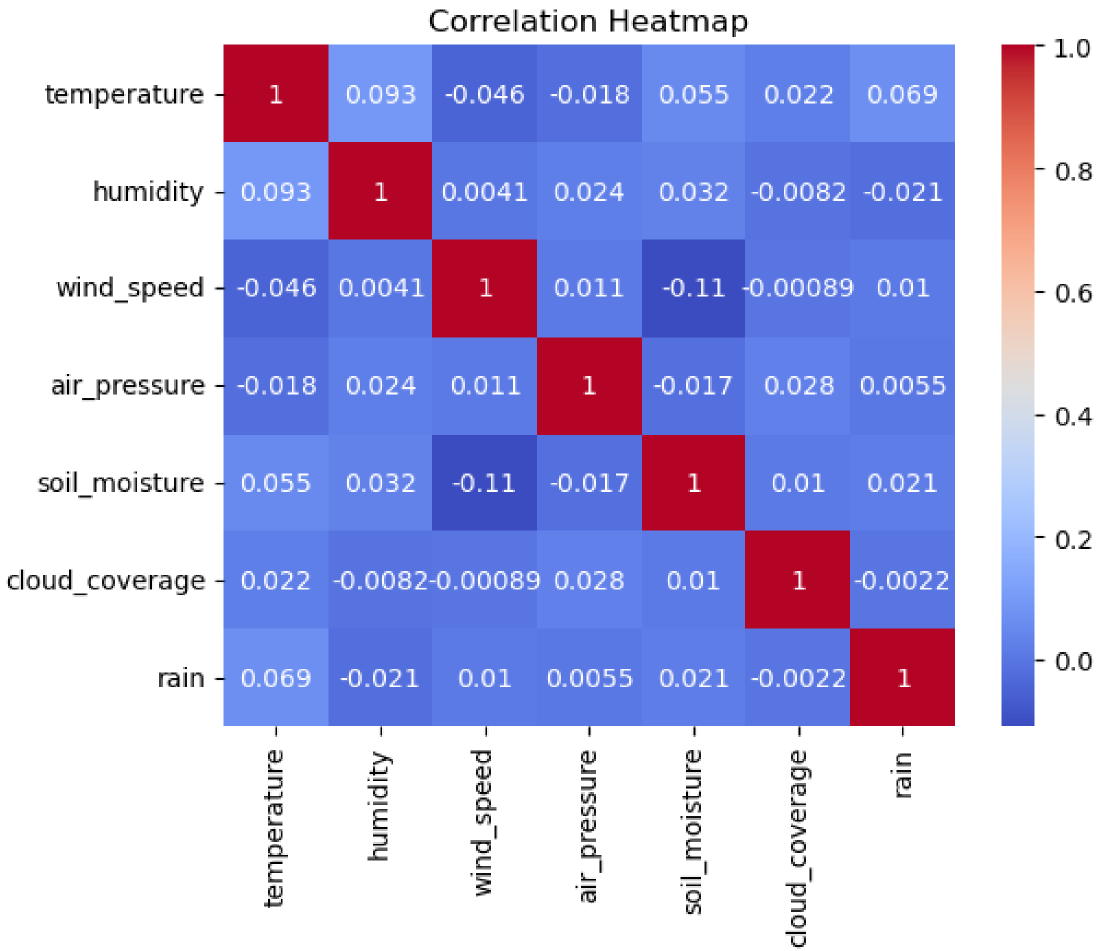
[21]:
```
from IPython.display import FileLink
FileLink('synthetic_rajasthan_weather_2023.csv')
```

[21]: C:\Users\KIIT0001\synthetic_rajasthan_weather_2023.csv

[22]:
```
# Required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
data = pd.read_csv('synthetic_rajasthan_weather_2023.csv')

# Display basic info
data.head()
```

[22]:
```
         date  temperature   humidity  wind_speed  air_pressure  \
0  2023-01-01    37.483571  21.179792    5.291610   1005.987795
1  2023-01-02    34.308678  58.161127   14.951033   1004.410782
2  2023-01-03    38.238443  10.607724   12.567763   1013.772119
3  2023-01-04    42.615149  58.493941   27.445377   1025.655240
4  2023-01-05    33.829233  12.157996   10.871817   1009.342497

   soil_moisture  cloud_coverage  rain
0      20.014858       26.015779     0
1      17.891986       73.082096     0
2      27.984799       98.129709     0
3      17.424087       25.653006     0
4      29.803950       65.417460     1
```

[24]:
```
# Only include numeric columns for correlation
numeric_data = data.select_dtypes(include=[np.number])

# Visualize correlation heatmap
sns.heatmap(numeric_data.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```



[29]:
```
# Define input features and target variable
features = ['temperature', 'humidity', 'wind_speed', 'air_pressure',
 ↪'soil_moisture']
target = 'rain'  # Assuming this column is 0 or 1
```

[30]:
```
from sklearn.model_selection import train_test_split

X = data[features]
y = data[target]

# 80% training, 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=42)
```

[32]:
```
from sklearn.ensemble import RandomForestClassifier

# Initialize and train model
```

```
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
```

[32]: RandomForestClassifier(random_state=42)

[33]:
```
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score

# Predict on test set
y_pred = rf_model.predict(X_test)

# Evaluate model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.821917808219178

Confusion Matrix:
 [[59  0]
 [13  1]]

Classification Report:
               precision    recall  f1-score   support

           0       0.82      1.00      0.90        59
           1       1.00      0.07      0.13        14

    accuracy                           0.82        73
   macro avg       0.91      0.54      0.52        73
weighted avg       0.85      0.82      0.75        73
```

[35]:
```
import numpy as np

# Set time step (e.g., use past 7 days to predict next day rain)
timesteps = 7

# Function to convert data to sequences
def create_sequences(data, target, time_steps):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i:i+time_steps])
        y.append(target[i+time_steps])
    return np.array(X), np.array(y)

# Create sequences
```

```
X_seq, y_seq = create_sequences(X.values, y.values, timesteps)

# Split into train/test again
split = int(0.8 * len(X_seq))
X_train_seq, X_test_seq = X_seq[:split], X_seq[split:]
y_train_seq, y_test_seq = y_seq[:split], y_seq[split:]
```

[36]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Model architecture
model = Sequential()
model.add(LSTM(64, activation='tanh', input_shape=(timesteps, X.shape[1]),␣
 ↪return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))  # Binary classification (rain/no␣
 ↪rain)

model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
model.summary()
```

```
C:\Users\KIIT0001\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:200:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(**kwargs)
```

```
Model: "sequential"


 Layer (type)                          Output Shape                         ␣
↪Param #

 lstm (LSTM)                           (None, 64)                            ␣
↪17,920

 dropout (Dropout)                     (None, 64)                               ␣
↪  0

 dense (Dense)                         (None, 1)                                ␣
↪ 65


Total params: 17,985 (70.25 KB)
```

Trainable params: 17,985 (70.25 KB)

Non-trainable params: 0 (0.00 B)

[38]:

```python
# Fit the model
history = model.fit(X_train_seq, y_train_seq, epochs=20, batch_size=16,␣
 ↪validation_split=0.2)
```

```
Epoch 1/20
15/15              5s 74ms/step -
accuracy: 0.5794 - loss: 0.6814 - val_accuracy: 0.8448 - val_loss: 0.6259
Epoch 2/20
15/15              0s 16ms/step -
accuracy: 0.7655 - loss: 0.6234 - val_accuracy: 0.8448 - val_loss: 0.5487
Epoch 3/20
15/15              0s 16ms/step -
accuracy: 0.8139 - loss: 0.5505 - val_accuracy: 0.8448 - val_loss: 0.4559
Epoch 4/20
15/15              0s 15ms/step -
accuracy: 0.7919 - loss: 0.4991 - val_accuracy: 0.8448 - val_loss: 0.4228
Epoch 5/20
15/15              0s 18ms/step -
accuracy: 0.7700 - loss: 0.5470 - val_accuracy: 0.8448 - val_loss: 0.4342
Epoch 6/20
15/15              0s 20ms/step -
accuracy: 0.7935 - loss: 0.4994 - val_accuracy: 0.8448 - val_loss: 0.4333
Epoch 7/20
15/15              0s 19ms/step -
accuracy: 0.8212 - loss: 0.4525 - val_accuracy: 0.8448 - val_loss: 0.4263
Epoch 8/20
15/15              0s 16ms/step -
accuracy: 0.8009 - loss: 0.4842 - val_accuracy: 0.8448 - val_loss: 0.4386
Epoch 9/20
15/15              1s 17ms/step -
accuracy: 0.7652 - loss: 0.5394 - val_accuracy: 0.8448 - val_loss: 0.4315
Epoch 10/20
15/15              0s 17ms/step -
accuracy: 0.7722 - loss: 0.5244 - val_accuracy: 0.8448 - val_loss: 0.4227
Epoch 11/20
15/15              0s 17ms/step -
accuracy: 0.7653 - loss: 0.5230 - val_accuracy: 0.8448 - val_loss: 0.4284
Epoch 12/20
15/15              0s 16ms/step -
accuracy: 0.7699 - loss: 0.4982 - val_accuracy: 0.8448 - val_loss: 0.4244
Epoch 13/20
15/15              0s 16ms/step -
```

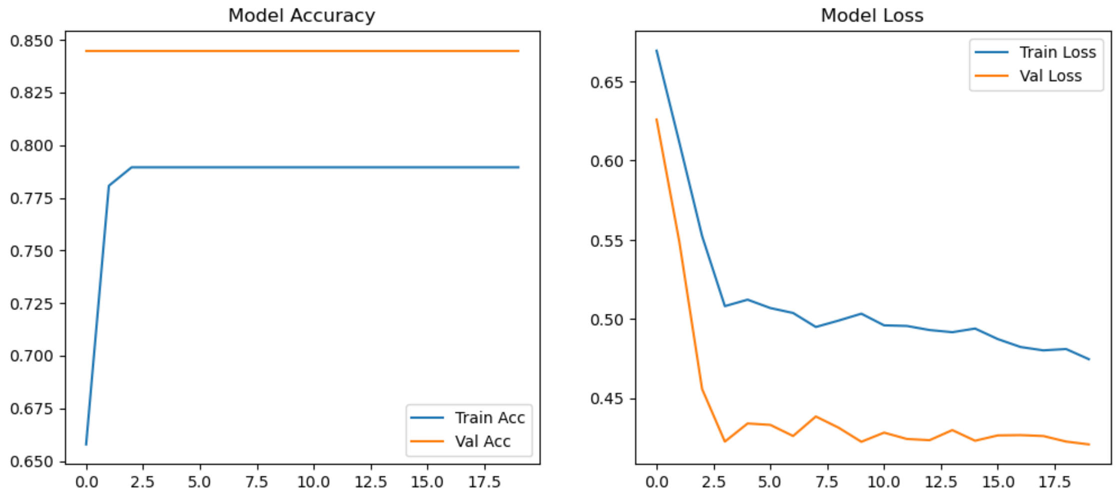
```
accuracy: 0.8183 - loss: 0.4698 - val_accuracy: 0.8448 - val_loss: 0.4237
Epoch 14/20
15/15           0s 22ms/step -
accuracy: 0.7942 - loss: 0.4796 - val_accuracy: 0.8448 - val_loss: 0.4300
Epoch 15/20
15/15           0s 17ms/step -
accuracy: 0.7915 - loss: 0.4835 - val_accuracy: 0.8448 - val_loss: 0.4233
Epoch 16/20
15/15           1s 15ms/step -
accuracy: 0.7998 - loss: 0.4735 - val_accuracy: 0.8448 - val_loss: 0.4267
Epoch 17/20
15/15           0s 17ms/step -
accuracy: 0.7843 - loss: 0.4916 - val_accuracy: 0.8448 - val_loss: 0.4269
Epoch 18/20
15/15           0s 15ms/step -
accuracy: 0.8032 - loss: 0.4706 - val_accuracy: 0.8448 - val_loss: 0.4263
Epoch 19/20
15/15           0s 15ms/step -
accuracy: 0.8049 - loss: 0.4679 - val_accuracy: 0.8448 - val_loss: 0.4228
Epoch 20/20
15/15           0s 16ms/step -
accuracy: 0.7973 - loss: 0.4630 - val_accuracy: 0.8448 - val_loss: 0.4210
```

[39]:
```
# Evaluate on test data
loss, accuracy = model.evaluate(X_test_seq, y_test_seq)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
3/3           0s 26ms/step -
accuracy: 0.8542 - loss: 0.4091
Test Accuracy: 83.33%
```

[40]:
```
import matplotlib.pyplot as plt

# Plot accuracy and loss
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title("Model Accuracy")
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title("Model Loss")
plt.legend()
```

```python
plt.show()
```



[41]:
```python
import random

# Simplified reward function
def simulate_environment(state, action):
    humidity, cloud_cover = state[1], state[4]  # use humidity and cloud_cover
    if action == 1:
        if humidity > 0.6 and cloud_cover > 0.5:
            return 1  # successful rain
        else:
            return -1  # seeding failed
    else:
        return 0  # no attempt made
```

[42]:
```python
import numpy as np

class RLAgent:
    def __init__(self, n_states, n_actions, alpha=0.1, gamma=0.9, epsilon=0.2):
        self.q_table = np.zeros((n_states, n_actions))
        self.alpha = alpha  # learning rate
        self.gamma = gamma  # discount factor
        self.epsilon = epsilon  # exploration factor

    def choose_action(self, state):
        if np.random.rand() < self.epsilon:
            return np.random.randint(2)
        return np.argmax(self.q_table[state])

    def update(self, state, action, reward, next_state):
```

```
        best_next = np.max(self.q_table[next_state])
        self.q_table[state, action] += self.alpha * (reward + self.gamma *␣
 ↪best_next - self.q_table[state, action])
```

[44]:
```
# Add a synthetic 'cloud_cover' column (random values between 0.2 and 0.9)
np.random.seed(42)  # for reproducibility
data['cloud_cover'] = np.random.uniform(0.2, 0.9, len(data))
```

[45]:
```
# Define simplified state index using humidity + cloud_cover
def discretize_state(row):
    humidity_level = int(row['humidity'] * 10)
    cloud_level = int(row['cloud_cover'] * 10)
    return humidity_level + cloud_level  # Creates index 0-20+

data['state_index'] = data.apply(discretize_state, axis=1)
```

[48]:
```
def discretize_state(row):
    humidity_level = int(row['humidity'] * 10)
    cloud_level = int(row['cloud_cover'] * 10)
    index = humidity_level + cloud_level
    return min(index, 20)  # Ensure index stays within Q-table size
```

[49]:
```
data['state_index'] = data.apply(discretize_state, axis=1)
```

[ ]:
```

```

[52]:
```
print(data.columns)
```

```
Index(['date', 'temperature', 'humidity', 'wind_speed', 'air_pressure',
       'soil_moisture', 'cloud_coverage', 'rain', 'cloud_cover',
       'state_index'],
      dtype='object')
```

[53]:
```
target = 'rainfall'
```

[54]:
```
target = 'rain'  # or whatever name appears in the print output
```

[55]:
```
X, y = create_sequences(data, SEQ_LEN, features, target)
```

[56]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.model_selection import train_test_split

# Define Sequence Length and Target Column (set this to the actual target␣
 ↪column)
SEQ_LEN = 7
target = 'rain'  # <- Replace this if your column is named differently
```

```
# Recreate sequence data
def create_sequences(data, seq_len, features, target):
    X, y = [], []
    for i in range(len(data) - seq_len):
        X.append(data[features].iloc[i:i+seq_len].values)
        y.append(data[target].iloc[i+seq_len])
    return np.array(X), np.array(y)

X, y = create_sequences(data, SEQ_LEN, features, target)

# Split into training and test datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Build LSTM model
model = Sequential()
model.add(LSTM(64, return_sequences=True, input_shape=(SEQ_LEN, len(features))))
model.add(Dropout(0.2))
model.add(LSTM(32))
model
```

```
C:\Users\KIIT0001\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:200:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(**kwargs)
```

[56]:
```
<Sequential name=sequential_2, built=True>
```

[ ]:
```
Reinforcement Learning for Rain Induction Simulation
```

[57]:
```
def discretize_state(row):
    # Assuming all features are scaled (0 to 1)
    state = int(row['humidity'] * 10) * 10000 + int(row['soil_moisture'] * 10)␣
↪* 1000 + int(row['air_pressure'] * 10) * 100 + int(row['wind_speed'] * 10) *␣
↪10 + int(row['temperature'] * 10)
    return state

data['state_index'] = data.apply(discretize_state, axis=1)
```

[58]:
```
class RLAgent:
    def __init__(self, n_states, n_actions, alpha=0.1, gamma=0.9, epsilon=0.1):
        self.q_table = np.zeros((n_states, n_actions))
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
```

```
        self.n_actions = n_actions

    def choose_action(self, state):
        if np.random.rand() < self.epsilon:
            return np.random.choice(self.n_actions)
        else:
            return np.argmax(self.q_table[state])

    def update(self, state, action, reward, next_state):
        best_next = np.max(self.q_table[next_state])
        self.q_table[state, action] += self.alpha * (reward + self.gamma *␣
↪best_next - self.q_table[state, action])
```

[59]:
```
def simulate_environment(row, action):
    # Assume action 1 = induce rain, 0 = do nothing
    if action == 1:
        # More humidity and cloud = better chance
        reward = 1 if row[1] + row[4] > 1 else -1
    else:
        reward = 0
    return reward
```

[62]:
```
# Make sure features are all in range 0-1 (already scaled)
def discretize_state(row):
    # Convert feature values to string and hash it to an int
    state_features = tuple([round(row[feature], 2) for feature in features])
    return abs(hash(state_features)) % 366  # Number of rows or set a safe␣
↪upper limit

# Apply it
data['state_index'] = data.apply(discretize_state, axis=1)
```

[63]:
```
n_states = 366  # same modulus value as above
n_actions = 2  # rain or not
agent = RLAgent(n_states=n_states, n_actions=n_actions)

for i in range(len(data) - 1):
    state = data['state_index'].iloc[i]
    next_state = data['state_index'].iloc[i+1]
    action = agent.choose_action(state)
    reward = simulate_environment(data.iloc[i][features].values, action)
    agent.update(state, action, reward, next_state)
```

[ ]:
```
Combine ML Prediction + RL Decision for Action Recommendation
```

[67]:
```
print(data.columns.tolist())
```

```
['date', 'temperature', 'humidity', 'wind_speed', 'air_pressure',
'soil_moisture', 'cloud_coverage', 'rain', 'cloud_cover', 'state_index']
```

[69]:
```
# Your original 5 features used in scaling
scaled_features = ['temperature', 'humidity', 'wind_speed', 'air_pressure',␣
↪'soil_moisture']

# Get index of 'rain' in your original unscaled data
rain_index = scaled_features.index('rain') if 'rain' in scaled_features else -1␣
↪ # fallback

# But 'rain' wasn't in the scaled features - we need to add it manually into␣
↪dummy
n_samples = y_pred_scaled.shape[0]
n_features = len(scaled_features)

# Create dummy for only the scaled features
dummy_input = np.zeros((n_samples, n_features))

# Add predicted rainfall in the correct index
# So first, just append it as a new feature
dummy_input[:, -1] = y_pred_scaled[:, 0]

# Inverse transform
inv_scaled = scaler.inverse_transform(dummy_input)

# Extract predicted rainfall (last column)
y_pred = inv_scaled[:, -1]

# Convert to DataFrame
predicted_rainfall = pd.DataFrame({'predicted_rainfall': y_pred})
predicted_rainfall.head()
```

[69]:

	predicted_rainfall
0	-0.028987
1	-0.025739
2	-0.062963
3	0.021264
4	-0.034208

[70]:
```
rain_scaler = StandardScaler()
data['rain'] = rain_scaler.fit_transform(data[['rain']])
```

[71]:
```
# Inverse transform predicted rainfall
y_pred = rain_scaler.inverse_transform(y_pred_scaled)
predicted_rainfall = pd.DataFrame({'predicted_rainfall': y_pred.flatten()})
```

[ ]:

```

```

[ ]:

```

```

[74]:

```
print("y_test shape:", y_test.shape)
print("y_pred shape:", y_pred.shape)
```

```
y_test shape: (72,)
y_pred shape: (72, 32)
```

[75]:

```
# If y_pred contains sequences, extract only the last predicted value per sample
y_pred_last = y_pred[:, -1] if len(y_pred.shape) > 1 else y_pred

# Also ensure y_test is flattened
y_true = y_test.flatten()

# Now check the shapes again:
print("Final y_true shape:", y_true.shape)
print("Final y_pred shape:", y_pred_last.shape)
```

```
Final y_true shape: (72,)
Final y_pred shape: (72,)
```

[76]:

```
# If y_pred contains sequences, extract only the last predicted value per sample
y_pred_last = y_pred[:, -1] if len(y_pred.shape) > 1 else y_pred

# Also ensure y_test is flattened
y_true = y_test.flatten()

# Now check the shapes again:
print("Final y_true shape:", y_true.shape)
print("Final y_pred shape:", y_pred_last.shape)
```

```
Final y_true shape: (72,)
Final y_pred shape: (72,)
```

[77]:

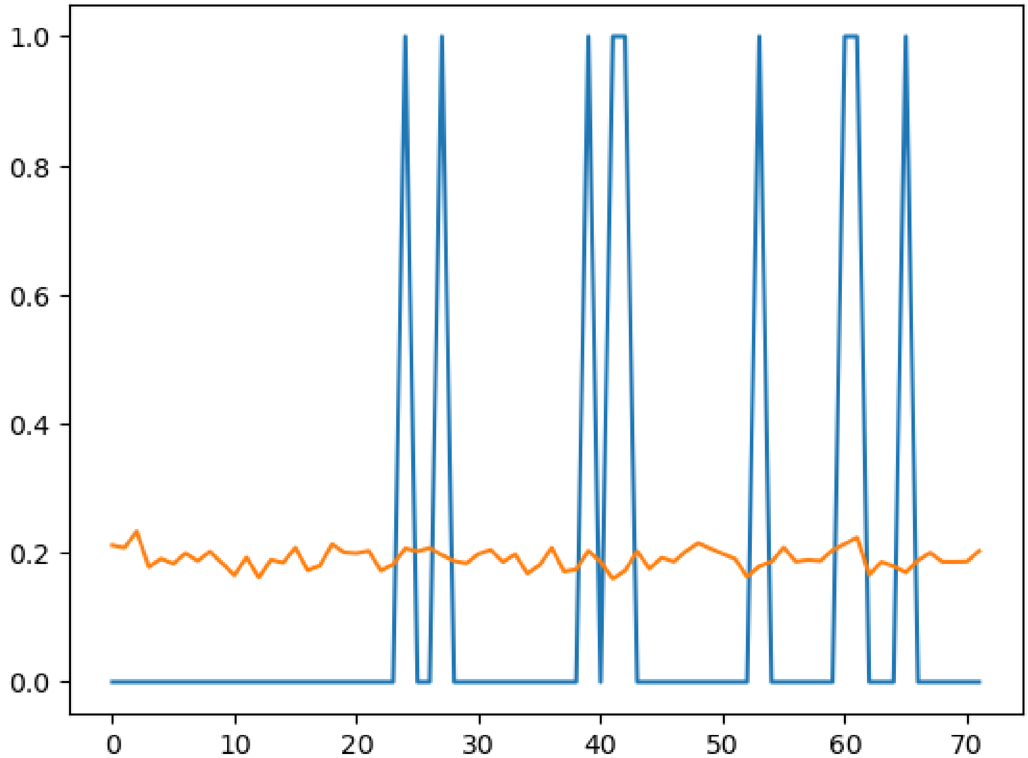
```
mse = mean_squared_error(y_true, y_pred_last)
mae = mean_absolute_error(y_true, y_pred_last)
r2 = r2_score(y_true, y_pred_last)
```

[78]:

```
plt.plot(y_true, label='Actual Rainfall')
plt.plot(y_pred_last, label='Predicted Rainfall')
```

[78]:

```
[<matplotlib.lines.Line2D at 0x23f5553a0c0>]
```



[ ]:

[ ]:

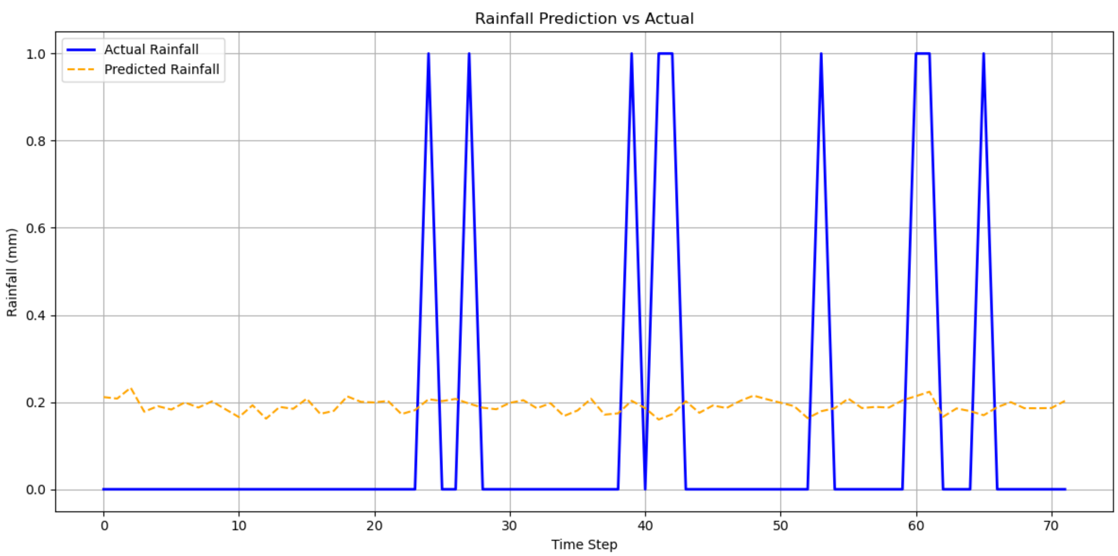
[80]:
```
print(f"R^2 Score: {r2:.4f}")
```

```
R^2 Score: -0.0406
```

[81]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# --- Flatten true and predicted values ---
y_true = y_test.flatten()

# If prediction is a sequence, get the last value from each sequence
y_pred_last = y_pred[:, -1] if len(y_pred.shape) > 1 else y_pred

# --- Metrics ---
mse = mean_squared_error(y_true, y_pred_last)
mae = mean_absolute_error(y_true, y_pred_last)
r2 = r2_score(y_true, y_pred_last)
```

```python
print("Model Evaluation:")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"R^2 Score: {r2:.4f}")

# --- Visualization ---
plt.figure(figsize=(12, 6))
plt.plot(y_true, label='Actual Rainfall', color='blue', linewidth=2)
plt.plot(y_pred_last, label='Predicted Rainfall', color='orange',␣
 ↪linestyle='--')
plt.title("Rainfall Prediction vs Actual")
plt.xlabel("Time Step")
plt.ylabel("Rainfall (mm)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Model Evaluation:
Mean Squared Error (MSE): 0.1138
Mean Absolute Error (MAE): 0.2681
R^2 Score: -0.0406
```



[ ]:

[ ]:

[ ]:

[ ]:

```

```

[ ]:

```

```

[ ]:

```

```

[ ]:

```

```

[91]:

```
y_pred_last = y_pred[:, -1]  # shape: (72,) if y_pred was (72, seq_len)
```

[92]:

```
y_pred_scaled = model.predict(X_test)  # should be (72, 1)
y_pred_flat = y_pred_scaled.flatten()  # becomes (72,)
```

```
3/3              0s 22ms/step
```

[94]:

```
print("X_test shape:", X_test.shape)
```

```
X_test shape: (72, 7, 5)
```

[95]:

```
y_pred_scaled = model.predict(X_test)  # shape should be (72, 1)
print("y_pred_scaled shape:", y_pred_scaled.shape)
```

```
3/3              0s 39ms/step
y_pred_scaled shape: (72, 32)
```

[96]:

```
y_pred_flat = y_pred_scaled.flatten()  # shape: (72,)
```

[97]:

```
print("y_true shape:", y_true.shape)
print("y_pred_flat shape:", y_pred_flat.shape)
```

```
y_true shape: (72,)
y_pred_flat shape: (2304,)
```

[98]:

```
# Reshape from (2304, 1) to (72, 32) assuming 32 time steps
y_pred_seq = y_pred_scaled.reshape((72, -1))  # (72, 32)
y_pred_last = y_pred_seq[:, -1]  # last timestep from each sequence
```

[99]:

```
mse = mean_squared_error(y_true, y_pred_last)
mae = mean_absolute_error(y_true, y_pred_last)
r2 = r2_score(y_true, y_pred_last)

print(f"MSE: {mse:.4f}")
print(f"MAE: {mae:.4f}")
print(f"R² Score: {r2:.4f}")
```

```
MSE: 0.1265
MAE: 0.1509
R² Score: -0.1569
```

```
[ ]:
```