Below is the revised report with placeholders for result images. You can later insert your actual result images where indicated.

**Detailed Report on ET and Rainfall Forecasting Process**

**1. Introduction**

This report outlines a comprehensive approach to forecasting evapotranspiration (ET) and rainfall using meteorological data obtained from an external API. The project is structured into three main components: data collection and preprocessing, ET prediction using a multi-step LSTM model, and rainfall prediction using a similar LSTM-based approach. Each section of the code is discussed in detail, including the purpose of key functions and the rationale behind design choices.

**2. Data Collection and Preprocessing (main.py)**

**2.1 Overview**

The main.py script is responsible for retrieving daily meteorological data (rainfall, maximum temperature, and minimum temperature) from the HCDP API for multiple farm locations (stations). It then computes ET using the Hargreaves–Samani method and stores the resulting data in a structured folder format.

**2.2 API Configuration and Station Setup**

The script begins by configuring the API token, listing farm stations with their coordinates, and defining the date range for data collection:

# Configuration

API\_TOKEN = "1b8a6439c85b8e42e211b68ea68ac198" # Replace with your actual HCDP API token

farms = [

{"name": "Kahuku Farm", "lat": 21.6832, "lng": -157.9604},

{"name": "Nozawa Farms", "lat": 21.688, "lng": -157.9648},

# ... additional farms ...

]

START\_YEAR = 2000

END\_YEAR = 2025

PARTIAL\_END\_DATE = "2025-03-12"

This section defines the scope of data collection across multiple years and sets the geographical context for each station.

**2.3 Calculating Extraterrestrial Radiation**

A crucial step in the ET computation is determining extraterrestrial radiation (Ra). The function below implements the FAO-56 method:

def extraterrestrial\_radiation\_mm(doy, latitude\_degs):

lat = math.radians(latitude\_degs)

Gsc = 0.082 # Solar constant (MJ/m^2/min)

dr = 1 + 0.033 \* math.cos((2 \* math.pi / 365) \* doy)

delta = 0.409 \* math.sin((2 \* math.pi / 365) \* doy - 1.39)

omega\_s = math.acos(-math.tan(lat) \* math.tan(delta))

Ra\_MJ = (24 \* 60 / math.pi) \* Gsc \* dr \* (

omega\_s \* math.sin(lat) \* math.sin(delta) +

math.cos(lat) \* math.cos(delta) \* math.sin(omega\_s)

)

Ra\_mm = 0.408 \* Ra\_MJ

return Ra\_mm

This function converts the latitude into radians and calculates the daily extraterrestrial radiation, an essential input for the ET calculation.

**2.4 Data Fetching and Merging**

The following function retrieves data for each parameter (rainfall, temperature) by querying the API and then converts the response into a pandas DataFrame:

def fetch\_daily\_data\_for\_year(lat, lng, datatype, start\_date, end\_date, aggregation=None):

params = {

"start": start\_date,

"end": end\_date,

"lat": lat,

"lng": lng,

"extent": "statewide",

"datatype": datatype,

"period": "day"

}

if datatype == "rainfall":

params["production"] = "new"

if datatype == "temperature" and aggregation:

params["aggregation"] = aggregation

response = requests.get(API\_URL, headers=headers, params=params)

if response.status\_code != 200:

print(f"Error fetching data ({datatype}, aggregation={aggregation}, range={start\_date}-{end\_date}):")

print(response.status\_code, response.text)

return pd.DataFrame(columns=["Date", "Value"])

else:

data = response.json()

df = pd.DataFrame(list(data.items()), columns=["Date", "Value"])

df["Date"] = pd.to\_datetime(df["Date"])

return df

The script then merges data for rainfall, maximum temperature, and minimum temperature by matching dates, computes the daily mean temperature, and finally calculates ET using the Hargreaves–Samani equation:

def compute\_et(row):

if pd.notnull(row["Tmax (°C)"]) and pd.notnull(row["Tmin (°C)"]) and pd.notnull(row["Tmean"]):

diff = row["Tmax (°C)"] - row["Tmin (°C)"]

if diff < 0:

return None

return 0.0023 \* (row["Tmean"] + 17.8) \* np.sqrt(diff) \* row["Ra\_mm"]

else:

return None

df\_merge["ET (mm/day)"] = df\_merge.apply(compute\_et, axis=1)

Finally, the computed data is saved into CSV files, organized by station and year, with a combined file for all years per station.

**3. Multi-Step LSTM Model for ET Prediction (model.py)**

**3.1 Overview**

The model.py script trains a multi-step LSTM neural network to forecast ET. The model uses historical data from multiple stations to predict ET for the next three days, employing a sliding window of 24 days as input.

**3.2 Data Loading and Preparation**

Data is loaded from the all\_years\_data.csv files for each station:

def load\_station\_data(station\_folder):

csv\_path = os.path.join(BASE\_DIR, station\_folder, "all\_years\_data.csv")

if not os.path.exists(csv\_path):

print(f"Warning: {csv\_path} not found. Station: {station\_folder}")

return pd.DataFrame()

df = pd.read\_csv(csv\_path)

if "Date" in df.columns:

df["Date"] = pd.to\_datetime(df["Date"])

df.sort\_values("Date", inplace=True)

df.reset\_index(drop=True, inplace=True)

return df

Data is then split into training and testing sets based on a designated test period for one station, while other stations contribute solely to the training data. Feature engineering (e.g., extracting day and month) is applied to refine the dataset.

**3.3 Sequence Creation**

The time-series sequences for model input are created using a sliding window approach. Each sample uses 24 days of historical data to forecast ET for the next three days:

def create\_sequences(df, window\_size, horizon, target\_col):

data = df.values

X, y = [], []

for i in range(len(df) - window\_size - horizon + 1):

X\_i = data[i : i + window\_size, :]

y\_i = data[i + window\_size : i + window\_size + horizon, df.columns.get\_loc(target\_col)]

X.append(X\_i)

y.append(y\_i)

return np.array(X), np.array(y)

**3.4 Model Architecture**

The LSTM model is built with one LSTM layer followed by a Dense layer that outputs a three-day forecast:

model = Sequential()

model.add(LSTM(64, activation='tanh', input\_shape=(WINDOW\_SIZE, num\_features)))

model.add(Dense(HORIZON))

model.compile(

loss='mse',

optimizer=Adam(learning\_rate=0.001),

metrics=['mae', r2\_keras]

)

This model uses mean squared error (MSE) as the loss function and includes a custom R² metric (r2\_keras) to evaluate performance.

**3.5 Training and Evaluation**

The model is trained with early stopping to prevent overfitting, using an 80/20 split for training and validation:

history = model.fit(

X\_train, y\_train,

validation\_data=(X\_val, y\_val),

epochs=50,

batch\_size=32,

callbacks=[early\_stop],

verbose=1

)

Post-training, the model is evaluated on the test dataset, and key performance metrics (MAE, MSE, RMSE, and R²) are computed. Additionally, the script generates time-series plots and scatter plots to visualize the comparison between actual and predicted ET values.

**Result Image Placeholder:**  
*[Insert your ET prediction result image here]*

**4. Multi-Step LSTM Model for Rainfall Prediction (model\_rainfall.py)**

**4.1 Overview**

The model\_rainfall.py script mirrors the approach used in model.py but targets rainfall prediction instead of ET. The methodology remains consistent, including data loading, sequence creation, and model training, with the primary change being the target variable.

**4.2 Key Adjustments**

The main difference is in the target column and corresponding adjustments in plotting:

TARGET\_COL = "Rainfall (mm)"

All other processes—feature engineering, sequence generation, model training, and evaluation—are analogous to the ET prediction model.

**Result Image Placeholder:**  
*[Insert your Rainfall prediction result image here]*

**5. Conclusion**

This report demonstrates a robust methodology for forecasting ET and rainfall using deep learning techniques. The process involves:

1. **Data Collection and Preprocessing:**
   * Retrieving and merging weather data from multiple farms.
   * Computing ET using the Hargreaves–Samani method.
   * Organizing the data into a structured folder format.
2. **ET Forecasting with LSTM:**
   * Loading and preprocessing the historical data.
   * Creating time-series sequences using a 24-day window to predict ET for the following three days.
   * Training a multi-step LSTM model and evaluating its performance using key metrics and visual plots.
3. **Rainfall Forecasting with LSTM:**
   * Applying the same methodology for rainfall prediction with adjusted parameters.
   * Evaluating the rainfall model with similar performance metrics and visualizations.

This detailed approach provides a clear framework for forecasting key agricultural parameters, with ample opportunities for further model tuning and performance improvement. The placeholders for result images indicate where visual evidence of the model's performance can be incorporated.

*End of Report*