# Weather Forecasting

# Part 1 -

In this project, we developed a machine learning model to predict whether it will rain or not based on historical weather data. We employed three different approaches to obtain results and achieved moderately accurate predictions.

# Preprocessing the Dataset

As part of the preprocessing step, we first encoded the target variable `rain_or_not` as a binary value, as it needs to be numeric for prediction.

```python
# Encode the target variable
df['rain_or_not']=df['rain_or_not'].map({'Rain':1, 'No Rain':0})
```

Next, we removed any rows with missing values to ensure data consistency.

```python
# Remove rows with missing values
df_cleaned = df.dropna()
```

# Exploratory Data Analysis (EDA)

We conducted various exploratory data analysis (EDA) techniques to better understand the dataset. One of the first steps was to examine the class imbalance in the data.

### Initial Rain/No Rain Distribution:

```python
# Calculate initial rain/no rain counts and percentages
initial_rain_count = (df['rain_or_not'] == 1).sum()
initial_no_rain_count = (df['rain_or_not'] == 0).sum()
initial_total = len(df)
```

```
initial_rain_percentage = (initial_rain_count / initial_total) *
100
initial_no_rain_percentage = (initial_no_rain_count /
initial_total) * 100
```

- **Rain:** 198 (63.67%)
- **No Rain:** 113 (36.33%)
- **Total:** 311

### Distribution After Removing Missing Values:

```python
# Calculate cleaned rain/no rain counts and percentages
cleaned_rain_count = (df_cleaned['rain_or_not'] == 1).sum()
cleaned_no_rain_count = (df_cleaned['rain_or_not'] == 0).sum()
cleaned_total = len(df_cleaned)
cleaned_rain_percentage = (cleaned_rain_count / cleaned_total) *
100
cleaned_no_rain_percentage = (cleaned_no_rain_count /
cleaned_total) * 100
```

- **Rain:** 189 (63.85%)
- **No Rain:** 107 (36.15%)
- **Total:** 296

Our conclusion was that removing missing values did not significantly impact the class imbalance.

To further analyze the data, we utilized various visualization techniques, including:

- **Histograms** – to understand the distribution of continuous variables.
- **Box Plots & Violin Plots** – to identify outliers and compare distributions.
- **Point Biserial Correlation** – to measure the relationship between continuous features and the binary target variable.

Additionally, we implemented a custom method to calculate the probability of rain based on the weather patterns of the last *n* days, providing deeper insights into seasonal variations and trends in rainfall patterns.

# Machine Learning Models

To train and obtain results, we experimented with three different machine learning models: Long Short-Term Memory (LSTM), Artificial Neural Networks (ANN), and Hidden Markov Models (HMM). Below, we provide a brief introduction to each model and explain why they were chosen for this problem.

## Method 1 - Long Short-Term Memory (LSTM)

LSTM is a type of recurrent neural network (RNN) designed to capture long-term dependencies in sequential data. Given that weather patterns are highly dependent on historical trends, LSTMs are a natural fit for this problem.

### LSTM Model Implementation

```python
model = Sequential()
model.add(LSTM(units=64, return_sequences=True,
input_shape=(X_train.shape[1], 1)))
model.add(LSTM(units=32, return_sequences=False))
model.add(Dense(1, activation='sigmoid'))
```

### LSTM Model Performance

The LSTM model produced decent results, but they were not fully satisfactory:

- **Accuracy:** 0.7143
- **Precision:** 0.6875
- **Recall:** 0.9167
- **F1-score:** 0.7857
- **AUC:** 0.6111

## Method 2 - Artificial Neural Network (ANN) Approach

ANNs are designed to detect patterns in data and are commonly used for classification tasks. Given the complexity of weather prediction, we implemented a deep ANN to explore its effectiveness.

### ANN Model Implementation

```python
model = Sequential()
```

```python
model.add(Dense(units=32, kernel_initializer='uniform',
activation='relu', input_dim=look_back))
model.add(Dense(units=32, kernel_initializer='uniform',
activation='relu'))
model.add(Dense(units=16, kernel_initializer='uniform',
activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(units=8, kernel_initializer='uniform',
activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=1, kernel_initializer='uniform',
activation='sigmoid'))

opt = Adam(learning_rate=0.00009)
model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])
```

#### ANN Model Performance

Despite being a more complex method, ANN did not perform as well as LSTM:

- **Accuracy:** 0.6667
- **Precision:** 0.7778
- **Recall:** 0.5833
- **F1-score:** 0.6667
- **AUC:** 0.6481

### Method 3 - Hidden Markov Model (HMM)

HMMs are probabilistic models used for sequential data. They can capture hidden states in time-series data, making them suitable for weather prediction where conditions evolve over time.

#### HMM Model Implementation

Python
```python
# Initialize the HMM (using a GaussianHMM for simplicity)
```

```
model = hmm.GaussianHMM(n_components=2, covariance_type="full",
n_iter=80)
```

**HMM Model Performance**

The HMM model was somewhat unpredictable and did not produce consistent results:

- **Accuracy:** 0.6667
- **Precision:** 0.6923
- **Recall:** 0.7500
- **F1-score:** 0.7200

## Final Conclusion

After evaluating the results of all three models, we concluded that LSTM was the most suitable choice for this problem. It performed the best in terms of recall and F1-score, making it the most effective at predicting rainfall.