1. Important Information

2. Loading The Data

Import all of the necessary libraries:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
```

Load the data from the '.csv' file into a 'pandas dataframe'.

In [2]:
```
data = pd.read_csv('RICE.csv')
print(data.info())
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19404 entries, 0 to 19403
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Observation Year  19404 non-null  int64  
 1   Standard Week     19404 non-null  int64  
 2   Pest Value        19404 non-null  float64
 3   Collection Type   19404 non-null  object 
 4   MaxT              19404 non-null  float64
 5   MinT              19404 non-null  float64
 6   RH1(%)            19404 non-null  float64
 7   RH2(%)            19404 non-null  float64
 8   RF(mm)            19404 non-null  float64
 9   WS(kmph)          19404 non-null  float64
 10  SSH(hrs)          19404 non-null  float64
 11  EVP(mm)           19404 non-null  float64
 12  PEST NAME         19404 non-null  object 
 13  Location          19404 non-null  object 
dtypes: float64(9), int64(2), object(3)
memory usage: 2.1+ MB
None
```

Out[2]:

	Observation Year	Standard Week	Pest Value	Collection Type	MaxT	MinT	RH1(%)	RH2(%)	RF(mm)
0	2003	1	0.0	Number/hill	27.9	14.8	94.7	51.3	0.0
1	2003	2	0.0	Number/hill	27.2	15.0	93.9	53.1	0.0
2	2003	3	0.0	Number/hill	28.7	18.3	94.1	56.7	0.6
3	2003	4	0.0	Number/hill	25.3	16.4	90.9	57.4	0.3
4	2003	5	0.0	Number/hill	28.8	18.7	95.7	55.0	0.0

3. Data Exploration

3.1 Attribute Analysis

Understanding Each Feature

- **Observation Year**: The fiscal year that the observation / reading was made.
- **Standard Week**: The week of the year that the observation / reading was made (Ranging from 1 to 52).
- **Pest Value**: The numerical value of the reading that was taken. This is used to create the target attribute.
- **Collection Type**: The process / procedure used to take the reading.
- **MaxT**: The maximum temperature during the respective week.
- **MinT**: The minimum temperature during the respective week.
- **RH1(%)**: The maximum relative humidity during the respective week.
- **RH2(%)**: The minimum relative humidity during the respective week.
- **RF(mm)**: The rainfall (in mm) during the respective week.
- **WS(kmph)**: The wind speed (in kmph) during the respective week.
- **SSH(hrs)**: The average sunshine hours per day during the respective week.
- **EVP(mm)**: The evaporation (in mm) during the respective week.
- **Pest Name**: The recorded pest's name.
- **Location**: The location where the reading was collected / recorded.

Coverage

- **Temporal Coverage Start Date**: 1960/12/31
- **Temporal Coverage End Date**: 2011/12/31
- **Geospatial Coverage**: India

In [3]:
```
data.describe()
```

Out[3]:

	Observation Year	Standard Week	Pest Value	MaxT	MinT	RH
count	19404.000000	19404.000000	19404.000000	19404.000000	19404.000000	19404.0C
mean	2000.024789	26.473717	807.944081	31.169006	20.404540	82.19
std	9.827306	15.016247	5290.180315	4.904610	5.388381	13.84
min	1959.000000	1.000000	0.000000	10.900000	0.800000	9.30
25%	1996.000000	13.000000	0.000000	28.800000	17.500000	79.10
50%	2001.000000	26.000000	3.000000	30.900000	22.000000	87.30
75%	2007.000000	39.000000	92.000000	33.425000	24.400000	91.00
max	2011.000000	52.000000	311169.000000	71.600000	30.900000	100.00

"Collection Type"

This shows that there are 5 different 'Collection Types' used to collect the 'Pest Value'. There are no deviations or inconsistencies with the names, therefore no normalization is needed.

In [4]:
```
print(data['Collection Type'].describe())
print('')
print(data['Collection Type'].unique())
print('')
print(data['Collection Type'].value_counts())
```

```
count                    19404
unique                       5
top          Number/Light trap
freq                     16430
Name: Collection Type, dtype: object

['Number/hill' 'Number/Light trap' 'Percent Damage'
 'Number/Pheromone trap' 'Percentage']

Collection Type
Number/Light trap        16430
Percentage                2298
Number/Pheromone trap      520
Percent Damage             104
Number/hill                 52
Name: count, dtype: int64
```

"Pest Name"

In [5]:
```
print(data['PEST NAME'].describe())
print('')
print(data['PEST NAME'].unique())
print('')
print(data['PEST NAME'].value_counts())
```

```
count                19404
unique                  11
top       Yellowstemborer
freq                  4333
Name: PEST NAME, dtype: object

['Brownplanthopper' 'Gallmidge' 'Greenleafhopper' 'LeafFolder'
 'Yellowstemborer' 'Caseworm' 'Miridbug' 'Whitebackedplanthopper'
 'ZigZagleafhopper' 'LeafBlast' 'NeckBlast']

PEST NAME
Yellowstemborer           4333
Gallmidge                 3016
Greenleafhopper           2287
LeafBlast                 2090
Brownplanthopper          1958
LeafFolder                1716
Whitebackedplanthopper    1248
Miridbug                  1144
Caseworm                   936
ZigZagleafhopper           468
NeckBlast                  208
Name: count, dtype: int64
```

"Location"

In [6]:

```
print(data['Location'].describe())
print('')
print(data['Location'].unique())
print('')
print(data['Location'].value_counts())
```

```
count        19404
unique           6
top       Maruteru
freq          7053
Name: Location, dtype: object

['Cuttack' 'Ludhiana' 'Maruteru' 'Palampur' 'Raipur' 'Rajendranagar']

Location
Maruteru         7053
Rajendranagar    5539
Raipur           2132
Ludhiana         1976
Cuttack          1456
Palampur         1248
Name: count, dtype: int64
```

3.2 Pest Name Analysis

"Brownplanthopper"

In [7]:

```
data_BrownPlanthopper = data[data['PEST NAME'] == 'Brownplanthopper']
print(data_BrownPlanthopper['Collection Type'].value_counts())
print('')
print(data_BrownPlanthopper['Location'].value_counts())
```

```
Collection Type
Number/Light trap    1906
Number/hill            52
Name: count, dtype: int64

Location
Maruteru         918
Rajendranagar    624
Ludhiana         312
Cuttack           52
Raipur            52
Name: count, dtype: int64
```

"Gallmidge"

In [8]:
```
data_Gallmidge = data[data['PEST NAME'] == 'Gallmidge']
print(data_Gallmidge['Collection Type'].value_counts())
print('')
print(data_Gallmidge['Location'].value_counts())
```

```
Collection Type
Number/Light trap    2912
Percent Damage        104
Name: count, dtype: int64

Location
Maruteru         884
Cuttack          832
Rajendranagar    728
Raipur           520
Ludhiana          52
Name: count, dtype: int64
```

"Greenleafhopper"

In [9]:
```
data_Greenleafhopper = data[data['PEST NAME'] == 'Greenleafhopper']
print(data_Greenleafhopper['Collection Type'].value_counts())
print('')
print(data_Greenleafhopper['Location'].value_counts())
```

```
Collection Type
Number/Light trap    2287
Name: count, dtype: int64

Location
Maruteru         831
Rajendranagar    676
Raipur           416
Ludhiana         312
Cuttack           52
Name: count, dtype: int64
```

"LeafFolder"

In [10]:
```
data_LeafFolder = data[data['PEST NAME'] == 'LeafFolder']
print(data_LeafFolder['Collection Type'].value_counts())
print('')
print(data_LeafFolder['Location'].value_counts())
```

```
Collection Type
Number/Light trap    1716
Name: count, dtype: int64

Location
Maruteru         884
Rajendranagar    312
Ludhiana         260
Raipur           208
Cuttack           52
Name: count, dtype: int64
```

"Yellowstemborer"

In [11]:
```
data_Yellowstemborer = data[data['PEST NAME'] == 'Yellowstemborer']
print(data_Yellowstemborer['Collection Type'].value_counts())
print('')
print(data_Yellowstemborer['Location'].value_counts())
```

```
Collection Type
Number/Light trap        3813
Number/Pheromone trap     520
Name: count, dtype: int64

Location
Rajendranagar    1629
Maruteru          936
Ludhiana          676
Raipur            624
Cuttack           468
Name: count, dtype: int64
```

"Caseworm"

In [12]:
```
data_Caseworm = data[data['PEST NAME'] == 'Caseworm']
print(data_Caseworm['Collection Type'].value_counts())
print('')
print(data_Caseworm['Location'].value_counts())
```

```
Collection Type
Number/Light trap    936
Name: count, dtype: int64

Location
Maruteru         468
Rajendranagar    260
Raipur           156
Ludhiana          52
Name: count, dtype: int64
```

"Miridbug"

In [13]:
```
data_Miridbug = data[data['PEST NAME'] == 'Miridbug']
print(data_Miridbug['Collection Type'].value_counts())
print('')
print(data_Miridbug['Location'].value_counts())
```

```
Collection Type
Number/Light trap    1144
Name: count, dtype: int64

Location
Maruteru         780
Rajendranagar    260
Ludhiana          52
Raipur            52
Name: count, dtype: int64
```

"Whitebackedplanthopper"

In [14]:
```
data_Whitebackedplanthopper = data[data['PEST NAME'] == 'Whitebackedplanthopper'
print(data_Whitebackedplanthopper['Collection Type'].value_counts())
print('')
print(data_Whitebackedplanthopper['Location'].value_counts())
```

```
Collection Type
Number/Light trap    1248
Name: count, dtype: int64

Location
Maruteru    884
Ludhiana    260
Raipur      104
Name: count, dtype: int64
```

"ZigZagleafhopper"

In [15]:
```
data_ZigZagleafhopper = data[data['PEST NAME'] == 'ZigZagleafhopper']
print(data_ZigZagleafhopper['Collection Type'].value_counts())
print('')
print(data_ZigZagleafhopper['Location'].value_counts())
```

```
Collection Type
Number/Light trap    468
Name: count, dtype: int64

Location
Maruteru    468
Name: count, dtype: int64
```

"LeafBlast"

In [16]:
```
data_LeafBlast = data[data['PEST NAME'] == 'LeafBlast']
print(data_LeafBlast['Collection Type'].value_counts())
print('')
print(data_LeafBlast['Location'].value_counts())
```

```
Collection Type
Percentage    2090
Name: count, dtype: int64

Location
Palampur         1092
Rajendranagar     998
Name: count, dtype: int64
```

"NeckBlast"

In [17]:
```
data_NeckBlast = data[data['PEST NAME'] == 'NeckBlast']
print(data_NeckBlast['Collection Type'].value_counts())
print('')
print(data_NeckBlast['Location'].value_counts())
```

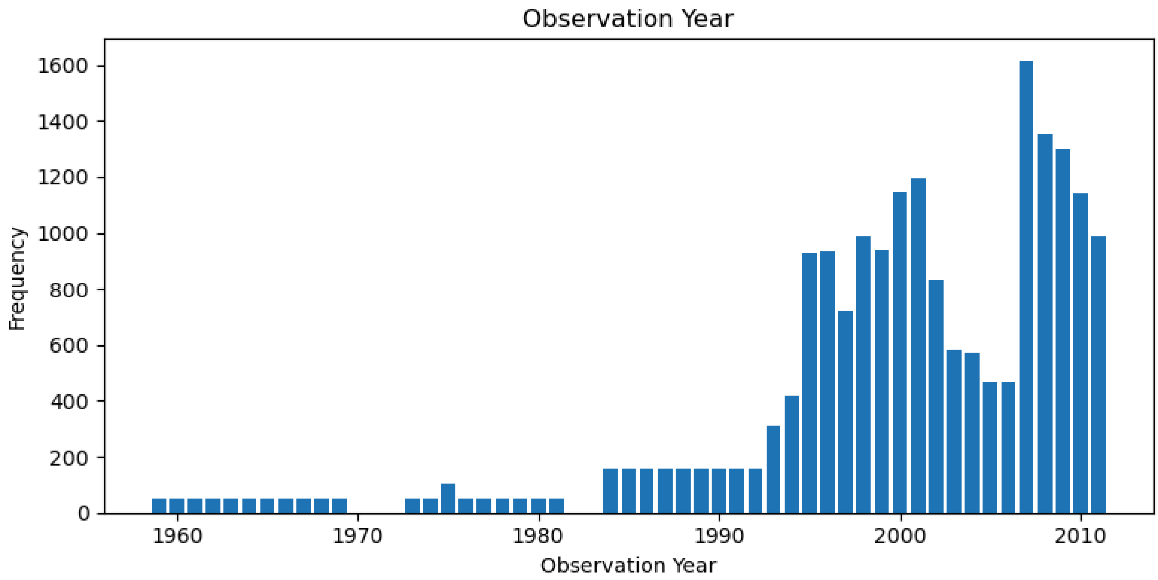
```
Collection Type
Percentage    208
Name: count, dtype: int64

Location
Palampur         156
Rajendranagar     52
Name: count, dtype: int64
```

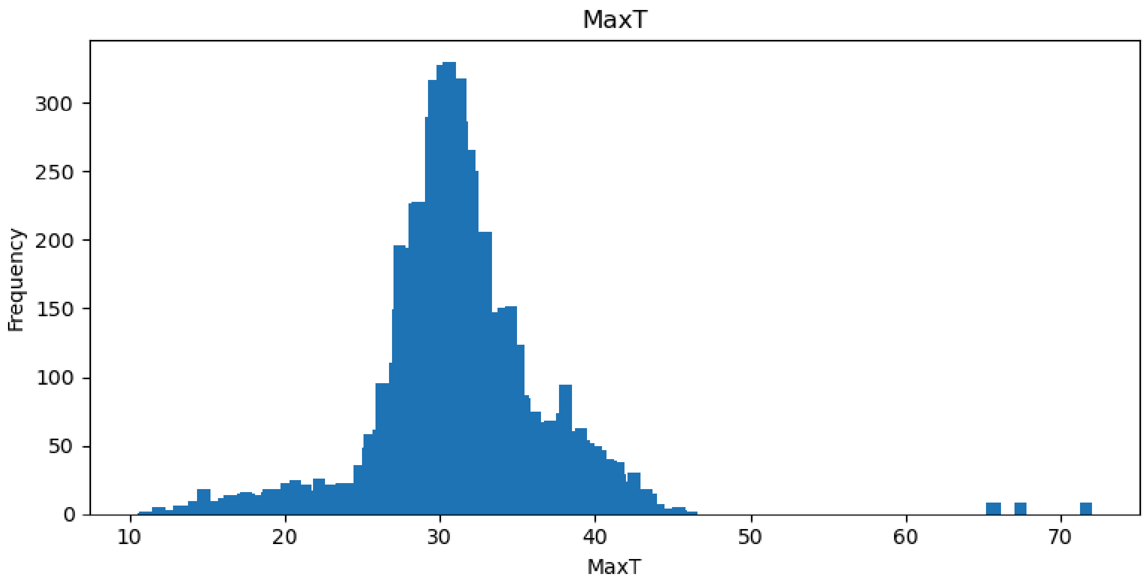
3.3 Data Visualization

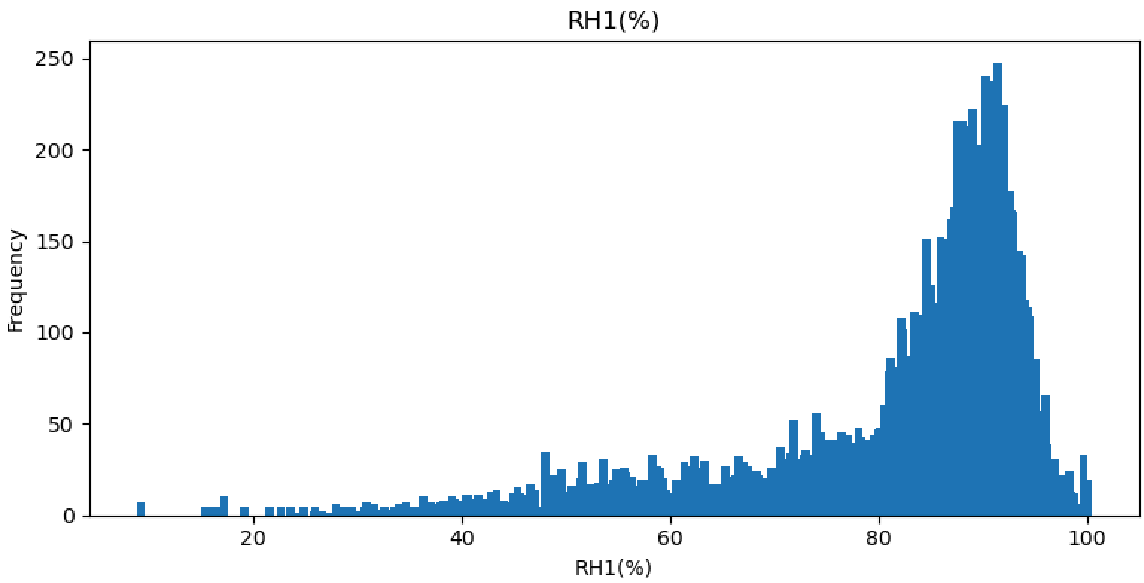
Attribute Frequency

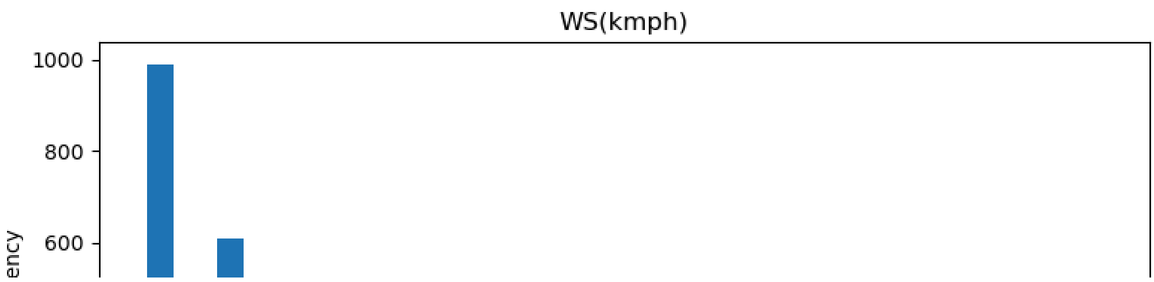
Bar Charts

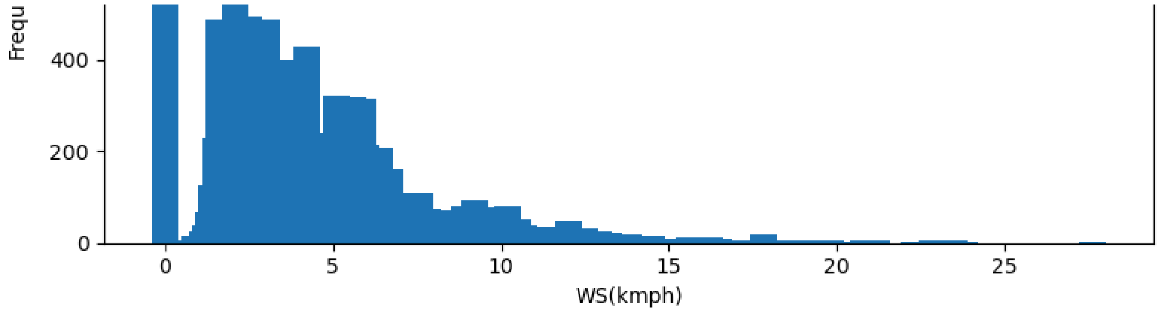
In [18]:
```
# Select the columns for bar plots
columns = ['Observation Year', 'MaxT', 'RH1(%)', 'WS(kmph)', 'PEST NAME']

# Create subplots for bar plots
fig, axs = plt.subplots(len(columns), 1, figsize=(8, 4*len(columns)))

# Create bar plots for each column
for i, column in enumerate(columns):
    # Count the frequency of each unique value in the column
    value_counts = data[column].value_counts()

    # Plotting the bar graph
    axs[i].bar(value_counts.index, value_counts.values)
    axs[i].set_title(column)
    axs[i].set_xlabel(column)
    axs[i].set_ylabel("Frequency")

# Adjust the spacing between subplots
plt.tight_layout()

# Display the plots
plt.show()
```

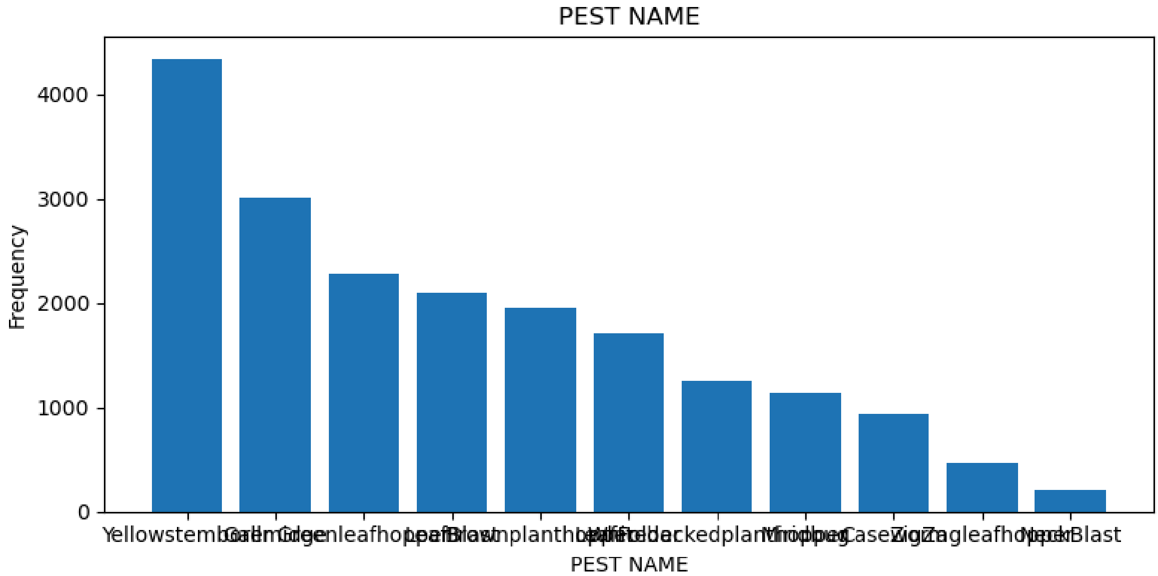

Observation Year
Frequency
0
200
400
600
800
1000
1200
1400
1600
1960
1970
1980
1990
2000
2010
Observation Year


MaxT
Frequency
0
50
100
150
200
250
300
10
20
30
40
50
60
70
MaxT


RH1(%)
Frequency
0
50
100
150
200
250
20
40
60
80
100
RH1(%)


WS(kmph)
600
800
1000

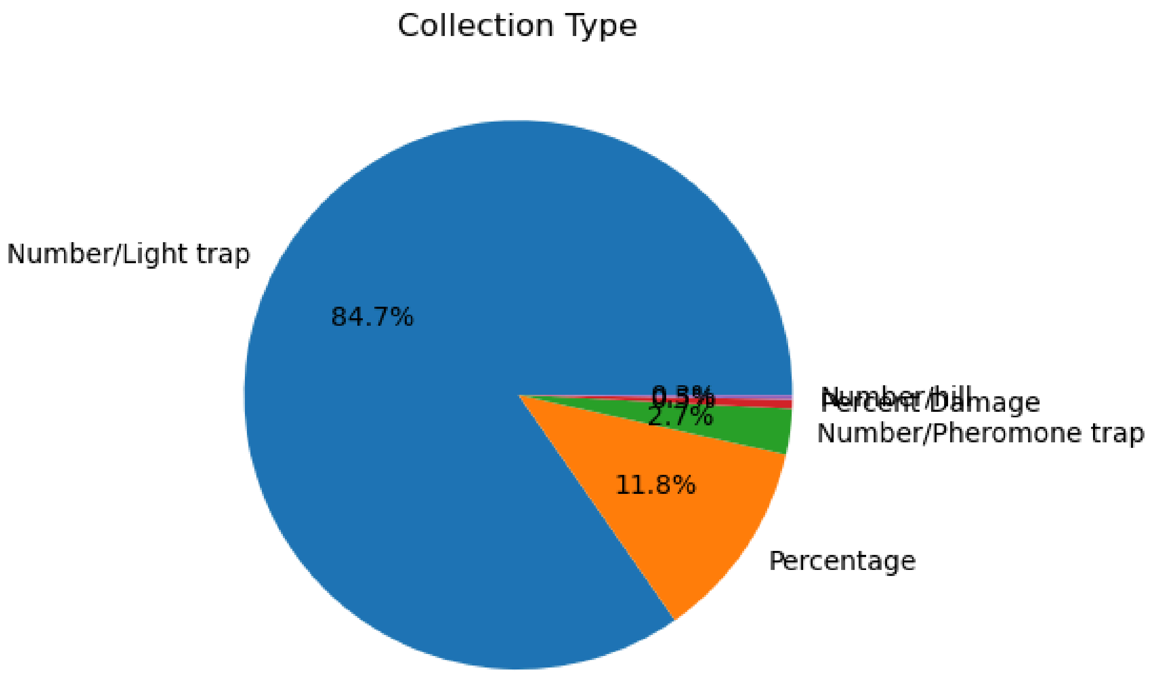


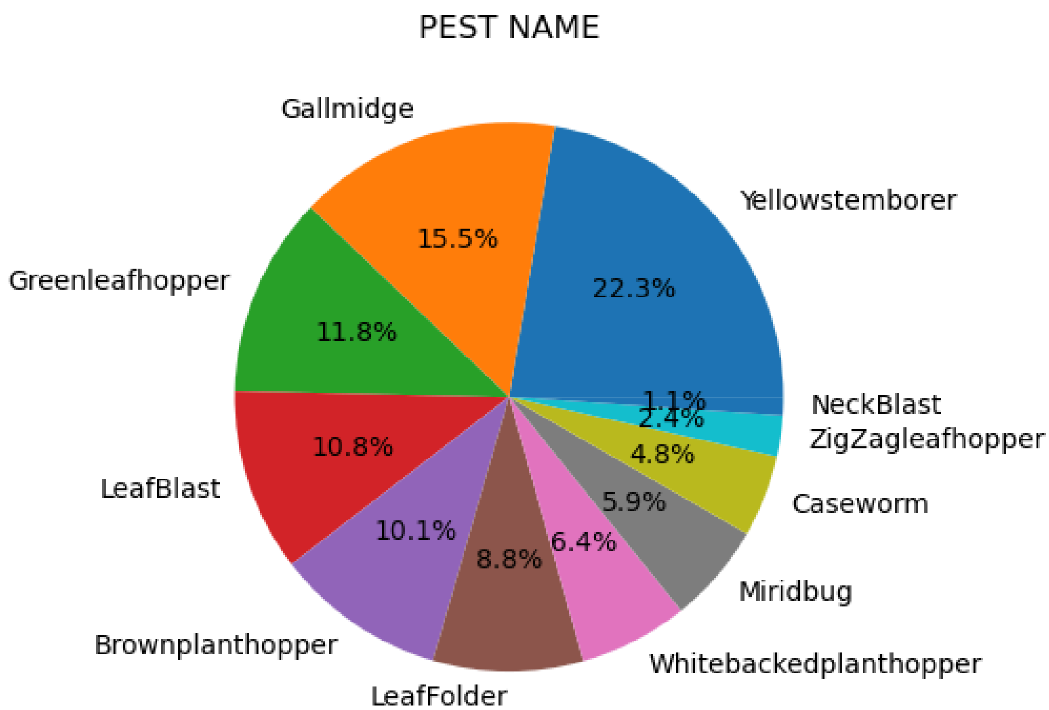


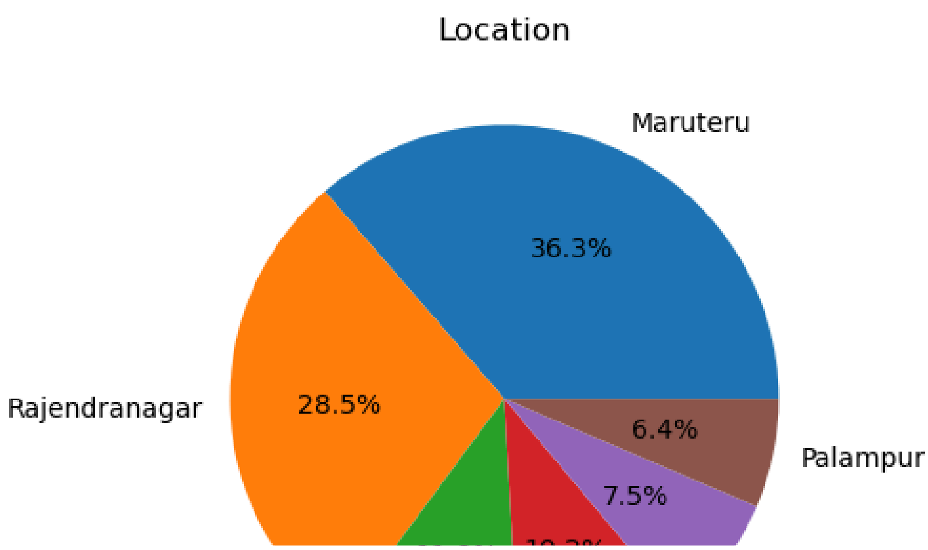
Pie Charts

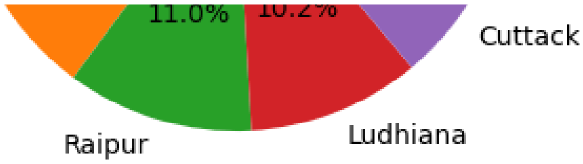
In [19]:

```python
# Select the columns for pie charts
columns = ['Collection Type', 'PEST NAME', 'Location']

# Create subplots for pie charts (stacked vertically)
fig, axs = plt.subplots(len(columns), 1, figsize=(8, 4*len(columns)))

# Ensure axs is iterable for single subplot case
if len(columns) == 1:
    axs = [axs]

# Create pie charts for each column
for i, column in enumerate(columns):
    # Count the frequency of each unique value in the column
    value_counts = data[column].value_counts()

    # Plotting the pie chart
    axs[i].pie(value_counts, labels=value_counts.index, autopct='%1.1f%%')
    axs[i].set_title(column)

# Adjust the spacing between subplots
plt.tight_layout()

# Display the plots
plt.show()
```


Collection Type
Number/Light trap
84.7%
11.8%
Percentage
2.7%
Number/Pheromone trap
0.3%
Number/hill
Percent Damage


PEST NAME
Gallmidge
15.5%
Yellowstemborer
22.3%
Greenleafhopper
11.8%
LeafBlast
10.8%
Brownplanthopper
10.1%
LeafFolder
8.8%
Whitebackedplanthopper
6.4%
Miridbug
5.9%
Caseworm
4.8%
ZigZagleafhopper
2.4%
NeckBlast
1.1%


Location
Maruteru
36.3%
Rajendranagar
28.5%
Palampur
6.4%
7.5%



4. Data Preprocessing

4.1 Creating The Target Attribute

The "Outbreak" attribute is used as the target attribute for the Machine Learning Models for the "pest outbreak prediction", which is a typical Decision Support System (DSS). It is created with the following logic:

"If the 'Pest Value' is anything above '0', it is seen as an outbreak."

In [20]:
```
data['Outbreak'] = (data['Pest Value'] > 0).astype(int)
data.head()
```

Out[20]:

	Observation Year	Standard Week	Pest Value	Collection Type	MaxT	MinT	RH1(%)	RH2(%)	RF(mm)
0	2003	1	0.0	Number/hill	27.9	14.8	94.7	51.3	0.0
1	2003	2	0.0	Number/hill	27.2	15.0	93.9	53.1	0.0
2	2003	3	0.0	Number/hill	28.7	18.3	94.1	56.7	0.6
3	2003	4	0.0	Number/hill	25.3	16.4	90.9	57.4	0.3
4	2003	5	0.0	Number/hill	28.8	18.7	95.7	55.0	0.0

Total Outbreak Percentage

In [21]:
```
print(data['Outbreak'].value_counts())
print('')
count_ones = data['Outbreak'].sum()
total = data['Outbreak'].count()
pct = count_ones / total * 100
print(f"Outbreak == 1: {count_ones}/{total} ({pct:.2f}%)")
```

```
Outbreak
1    10734
0     8670
Name: count, dtype: int64

Outbreak == 1: 10734/19404 (55.32%)
```

This represents a balanced dataset. Allowing the model to be trained equally with records that contain an outbreak and records that don't.

Removing The 'Pest Value' Attribute

The 'Pest Value' attribute was used to create the target attribute called 'Outbreak', therefore it is no longer needed. Now it will be removed from the dataset.

In [22]:
```
data.drop(columns=['Pest Value'], inplace=True)
data.head()
```

Out[22]:

	Observation Year	Standard Week	Collection Type	MaxT	MinT	RH1(%)	RH2(%)	RF(mm)	WS(kn
0	2003	1	Number/hill	27.9	14.8	94.7	51.3	0.0	
1	2003	2	Number/hill	27.2	15.0	93.9	53.1	0.0	
2	2003	3	Number/hill	28.7	18.3	94.1	56.7	0.6	
3	2003	4	Number/hill	25.3	16.4	90.9	57.4	0.3	
4	2003	5	Number/hill	28.8	18.7	95.7	55.0	0.0	

Empty Values

This is to check if there are any empty values in the dataset. This shows that all of the columns and records are populated.

In [23]:
```
data.isnull().sum()
```

Out[23]:
```
Observation Year    0
Standard Week       0
Collection Type     0
MaxT                0
MinT                0
RH1(%)              0
RH2(%)              0
RF(mm)              0
WS(kmph)            0
SSH(hrs)            0
EVP(mm)             0
PEST NAME           0
Location            0
Outbreak            0
dtype: int64
```

4.2 Duplicate Records

This is to check if there are any duplicate records found in the dataset. They are then removed. This dataset doesn't contain any duplicates.

In [24]:
```
print(data.duplicated().value_counts())
data = data.drop_duplicates()
```

```
False    19404
Name: count, dtype: int64
```

4.3 Handling Outliers

In [25]:
```
# Plot / Visualize the outliers of the numerical features
for col in data[['MaxT', 'MinT', 'RH1(%)', 'RH2(%)', 'RF(mm)', 'WS(kmph)', 'SSH(
    fig = px.box(
        data_frame=data,
        x=col,
        orientation='h',
        title=f'Boxplot of the Target ({col}) - With Outliers'
    )
    fig.show()
```

In [26]:
```
# Create a mask to filter out the outliers for 'LoanAmount'
# mask_MaxT = (data['MaxT'] >= 23.3) & (data['MaxT'] <= 38.7)

# # print(data[mask_MaxT].head())
# # print(data[mask_MaxT].info())

# fig = px.box(
#     data_frame=data[mask_MaxT],
#     x='MaxT',
#     orientation='h',
#     title='Boxplot of the Target (MaxT) - Without Outliers')

# fig.update_layout(xaxis_title='Target')
# fig.show()
```

4.4 Encode Categorical Variables

Use One-Hot encoding for 'PEST NAME', 'Location', and 'Collection Type'.

In [27]:
```
# 1. Define categorical and numeric columns
cat_cols = ['Collection Type', 'PEST NAME', 'Location']
num_cols = [c for c in data.columns if c not in cat_cols + ['Outbreak']]

# 2. One-hot encode categorical features
data = pd.get_dummies(data, columns=cat_cols, drop_first=False)
```

5. Modelling

5.1 Splitting The Data

The data needs to be split into the training set and the testing set. 80% of the dataset will be assigned to the training set and 20% will be assigned to the training set. The sets are assigned randomly to ensure integrity.

In [28]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split

X = data.drop(columns=['Outbreak'], inplace=False)
y = data['Outbreak']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

5.2 Random Forest Model

A Random Forest Model is trained using the training set from above.

In [29]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, roc_auc_score

# Random Forest model
rf_clf = RandomForestClassifier(
    n_estimators=300,
    max_depth=None,
    random_state=42,
    n_jobs=-1,
    class_weight=None  # or 'balanced' if you use a stricter Outbreak threshold
)

rf_clf.fit(X_train, y_train)

y_pred = rf_clf.predict(X_test)
y_prob = rf_clf.predict_proba(X_test)[:, 1]

print("Random Forest classification report:")
print(classification_report(y_test, y_pred))
print("Random Forest ROC AUC:", roc_auc_score(y_test, y_prob))
```

```
Random Forest classification report:
              precision    recall  f1-score   support

           0       0.88      0.81      0.84      1734
           1       0.85      0.91      0.88      2147

    accuracy                           0.87      3881
   macro avg       0.87      0.86      0.86      3881
weighted avg       0.87      0.87      0.86      3881

Random Forest ROC AUC: 0.9414335283964266
```

5.3 Gradient Boosting Model

A Gradient Boosting Model is trained using the training set from above.

In [30]:
```python
from xgboost import XGBClassifier

xgb_clf = XGBClassifier(
    n_estimators=400,
    learning_rate=0.05,
    max_depth=5,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42,
    n_jobs=-1,
    eval_metric='logloss'  # suppresses warning
    # scale_pos_weight can be tuned if classes become imbalanced
)

xgb_clf.fit(X_train, y_train)
```

```python
y_pred_xgb = xgb_clf.predict(X_test)
y_prob_xgb = xgb_clf.predict_proba(X_test)[:, 1]

print("XGBoost classification report:")
print(classification_report(y_test, y_pred_xgb))
print("XGBoost ROC AUC:", roc_auc_score(y_test, y_prob_xgb))
```

```
XGBoost classification report:
              precision    recall  f1-score   support

           0       0.89      0.81      0.85      1734
           1       0.86      0.92      0.89      2147

    accuracy                           0.87      3881
   macro avg       0.87      0.87      0.87      3881
weighted avg       0.87      0.87      0.87      3881

XGBoost ROC AUC: 0.9433640137333874
```

5.4 Making Predictions With The Models

This is to use the trained models on new raw records. When you want to predict for new raw rows (with original columns).

In [31]:
```python
# 1. Example new raw records (same columns as original data before encoding)
df_new_raw = pd.DataFrame([
    {
        "Observation Year": 2010,
        "Standard Week": 30,
        "Pest Value": 50.0,                 # this will be dropped, same as whe
        "Collection Type": "Number/Light trap",
        "MaxT": 32.5,
        "MinT": 24.0,
        "RH1(%)": 90.0,
        "RH2(%)": 60.0,
        "RF(mm)": 25.0,
        "WS(kmph)": 4.0,
        "SSH(hrs)": 7.0,
        "EVP(mm)": 4.5,
        "PEST NAME": "Brownplanthopper",
        "Location": "Cuttack"
    },
    {
        "Observation Year": 2005,
        "Standard Week": 10,
        "Pest Value": 0.0,
        "Collection Type": "Number/Light trap",
        "MaxT": 28.0,
        "MinT": 18.0,
        "RH1(%)": 80.0,
        "RH2(%)": 45.0,
        "RF(mm)": 0.0,
        "WS(kmph)": 3.0,
        "SSH(hrs)": 8.5,
        "EVP(mm)": 3.0,
        "PEST NAME": "Yellowstemborer",
        "Location": "Maruteru"
```

```
    },
    {
        "Observation Year": 2008,
        "Standard Week": 38,
        "Pest Value": 200.0,
        "Collection Type": "Number/Light trap",
        "MaxT": 33.0,
        "MinT": 23.0,
        "RH1(%)": 92.0,
        "RH2(%)": 65.0,
        "RF(mm)": 40.0,
        "WS(kmph)": 6.0,
        "SSH(hrs)": 5.5,
        "EVP(mm)": 5.0,
        "PEST NAME": "LeafBlast",
        "Location": "Rajendranagar"
    }
])

# 2. Drop the "Pest Value" attribute
df_new_raw = df_new_raw.drop(columns=["Pest Value"])

# 3. Function from previous answer, adjusted to use global cat_cols
cat_cols = ['Collection Type', 'PEST NAME', 'Location']  # must match training
trained_feature_cols = X_train.columns                    # from the training ste

def prepare_new_records(df_raw, trained_columns, cat_cols):
    # One-hot encode categorical as in training
    df_encoded = pd.get_dummies(df_raw, columns=cat_cols, drop_first=False)

    # Add missing dummy columns
    for col in trained_columns:
        if col not in df_encoded.columns:
            df_encoded[col] = 0

    # Drop extra columns not used in training
    df_encoded = df_encoded[trained_columns]

    return df_encoded

# 4. Prepare new data
X_new = prepare_new_records(df_new_raw, trained_feature_cols, cat_cols)

# 5. Predict with trained models
# Random Forest predictions
rf_preds_new = rf_clf.predict(X_new)
rf_probs_new = rf_clf.predict_proba(X_new)[:, 1]

print("New raw records:")
print(df_new_raw)
print("\nRandom Forest predicted Outbreak (0/1):", rf_preds_new)
print("Random Forest predicted probability of Outbreak:", rf_probs_new)

print("\n")

# XGBoost predictions
xgb_preds_new = xgb_clf.predict(X_new)
xgb_probs_new = xgb_clf.predict_proba(X_new)[:, 1]
```

```
print("XGBoost predicted Outbreak (0/1):", xgb_preds_new)
print("XGBoost predicted probability of Outbreak:", xgb_probs_new)
```

```
New raw records:
   Observation Year  Standard Week    Collection Type  MaxT  MinT  RH1(%)  \
0              2010             30  Number/Light trap  32.5  24.0    90.0
1              2005             10  Number/Light trap  28.0  18.0    80.0
2              2008             38  Number/Light trap  33.0  23.0    92.0

   RH2(%)  RF(mm)  WS(kmph)  SSH(hrs)  EVP(mm)         PEST NAME  \
0    60.0    25.0       4.0       7.0      4.5  Brownplanthopper
1    45.0     0.0       3.0       8.5      3.0   Yellowstemborer
2    65.0    40.0       6.0       5.5      5.0         LeafBlast

        Location
0        Cuttack
1       Maruteru
2  Rajendranagar

Random Forest predicted Outbreak (0/1): [0 1 1]
Random Forest predicted probability of Outbreak: [0.48       0.91333333 0.79
]


XGBoost predicted Outbreak (0/1): [0 1 1]
XGBoost predicted probability of Outbreak: [0.258903   0.96805286 0.91739213]
XGBoost predicted Outbreak (0/1): [0 1 1]
XGBoost predicted probability of Outbreak: [0.258903   0.96805286 0.91739213]
```

5.5 Long Short-Term Memory Model - Specific

The LSTM Model will be trained on one pest and one location. The following code focuses on Yellowstemborer in Rajendranagar. The pest (Yellowstemborer) was chosen, seeing as it is the pest with the most amount of records. The location (Rajendranagar) was chosen, because it was the location with the most amount of records for this pest.

1. Choose Features And Sort

In [39]:
```
data = pd.read_csv('RICE.csv')
data['Outbreak'] = (data['Pest Value'] > 0).astype(int)

# Start from your cleaned 'data' where Pest Value is dropped and Outbreak is def
# data columns: Observation Year, Standard Week, Collection Type, MaxT, ..., PES

# 1. Filter to one pest and one location (simplest case)
pest = "Yellowstemborer"
loc = "Rajendranagar"

df = data[(data["PEST NAME"] == pest) & (data["Location"] == loc)].copy()
print(df.head())
print(df.info())

# 2. Sort by time
df = df.sort_values(by=["Observation Year", "Standard Week"]).reset_index(drop=T

# 3. (Optional) Drop columns you don't want as inputs
# For a first LSTM, you might use only numeric weather + week + year as inputs
```

```python
input_cols = [
    "Observation Year",
    "Standard Week",
    "MaxT",
    "MinT",
    "RH1(%)",
    "RH2(%)",
    "RF(mm)",
    "WS(kmph)",
    "SSH(hrs)",
    "EVP(mm)",
]
target_col = "Outbreak"

df_model = df[input_cols + [target_col]].copy()
```

```
       Observation Year  Standard Week  Pest Value    Collection Type  MaxT  \
17775              1975              1        11.0  Number/Light trap  31.4
17776              1975              2        27.0  Number/Light trap  31.3
17777              1975              3        52.0  Number/Light trap  30.4
17778              1975              4        63.0  Number/Light trap  31.0
17779              1975              5         0.0  Number/Light trap  30.2

       MinT  RH1(%)  RH2(%)  RF(mm)  WS(kmph)  SSH(hrs)  EVP(mm)  \
17775  17.5    79.6    43.7     0.8       1.8       8.0      3.4
17776  16.2    83.7    38.0     0.0       2.1       9.4      4.1
17777  13.7    79.0    37.3     0.0       2.2      10.0      3.8
17778  14.4    80.4    34.9     0.0       1.6       9.1      3.5
17779  16.3    81.1    38.2     1.2       2.7       7.2      3.5

             PEST NAME       Location  Outbreak
17775  Yellowstemborer  Rajendranagar         1
17776  Yellowstemborer  Rajendranagar         1
17777  Yellowstemborer  Rajendranagar         1
17778  Yellowstemborer  Rajendranagar         1
17779  Yellowstemborer  Rajendranagar         0
<class 'pandas.core.frame.DataFrame'>
Index: 1629 entries, 17775 to 19403
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Observation Year  1629 non-null   int64
 1   Standard Week     1629 non-null   int64
 2   Pest Value        1629 non-null   float64
 3   Collection Type   1629 non-null   object
 4   MaxT              1629 non-null   float64
 5   MinT              1629 non-null   float64
 6   RH1(%)            1629 non-null   float64
 7   RH2(%)            1629 non-null   float64
 8   RF(mm)            1629 non-null   float64
 9   WS(kmph)          1629 non-null   float64
 10  SSH(hrs)          1629 non-null   float64
 11  EVP(mm)           1629 non-null   float64
 12  PEST NAME         1629 non-null   object
 13  Location          1629 non-null   object
 14  Outbreak          1629 non-null   int32
dtypes: float64(9), int32(1), int64(2), object(3)
memory usage: 197.3+ KB
None
```

2. Scale numeric features

Use MinMax or standard scaling. Here is MinMax to using scikit-learn. Save "scaler" for later to transform any future data identically.

In [40]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df_model_scaled = df_model.copy()
df_model_scaled[input_cols] = scaler.fit_transform(df_model[input_cols])
df_model_scaled.head()
```

Out[40]:

	Observation Year	Standard Week	MaxT	MinT	RH1(%)	RH2(%)	RF(mm)	WS(kmph)
0	0.0	0.000000	0.396739	0.492958	0.744648	0.405759	0.003810	0.065217
1	0.0	0.019608	0.391304	0.431925	0.807339	0.331152	0.000000	0.076087
2	0.0	0.039216	0.342391	0.314554	0.735474	0.321990	0.000000	0.079710
3	0.0	0.058824	0.375000	0.347418	0.756881	0.290576	0.000000	0.057971
4	0.0	0.078431	0.331522	0.436620	0.767584	0.333770	0.005714	0.097826

3. Build supervised sequences (sliding windows)

Define a helper to turn a univariate time series with features into sequences of length T predicting the next week's Outbreak.

In [41]:
```
def make_sequences(df_scaled, input_cols, target_col, window_size=4):
    """
    df_scaled: DataFrame with scaled inputs and target
    Returns X (num_samples, window_size, num_features), y (num_samples,)
    """
    X_list, y_list = [], []
    values = df_scaled[input_cols + [target_col]].values
    n_total = len(values)

    for i in range(n_total - window_size):
        window = values[i : i + window_size]
        target = values[i + window_size, -1]  # Outbreak after the window
        X_list.append(window[:, :-1])        # all input features over the windo
        y_list.append(target)

    X = np.array(X_list)
    y = np.array(y_list).astype(int)
    return X, y

window_size = 4  # e.g., use 4 weeks history
X, y = make_sequences(df_model_scaled, input_cols, target_col, window_size=windc

print("X shape:", X.shape)  # (samples, time_steps=window_size, features=len(inp
print("y shape:", y.shape)
```

```
X shape: (1625, 4, 10)
y shape: (1625,)
```

4. Train/test split for sequences

Use a chronological split to respect time ordering (no shuffling).

In [42]:
```
# Simple temporal split: first 80% for train, last 20% for test
n_samples = X.shape[0]
split_index = int(n_samples * 0.8)

X_train, X_test = X[:split_index], X[split_index:]
y_train, y_test = y[:split_index], y[split_index:]

print("Train samples:", X_train.shape[0], "Test samples:", X_test.shape[0])
```

```
Train samples: 1300 Test samples: 325
```

5. Define and train an LSTM model

Example with Keras (TensorFlow backend):

In [43]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

n_timesteps = X_train.shape[1]
n_features = X_train.shape[2]

model = Sequential()
model.add(LSTM(64, input_shape=(n_timesteps, n_features), return_sequences=False
model.add(Dropout(0.3))
model.add(Dense(32, activation="relu"))
model.add(Dense(1, activation="sigmoid"))  # binary classification

model.compile(
    loss="binary_crossentropy",
    optimizer="adam",
    metrics=["accuracy"]
)

history = model.fit(
    X_train, y_train,
    epochs=30,
    batch_size=32,
    validation_split=0.2,
    shuffle=False  # keep temporal order
)
```

```
Epoch 1/30
C:\Users\hroux\AppData\Roaming\Python\Python312\site-packages\keras\src\layers\rn
n\rnn.py:199: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequenti
al models, prefer using an `Input(shape)` object as the first layer in the model
instead.
```

```
33/33 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5144 - loss: 0.6918 - val_ac
curacy: 0.7192 - val_loss: 0.6615
Epoch 2/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5683 - loss: 0.6829 - val_ac
curacy: 0.7192 - val_loss: 0.6380
Epoch 3/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5673 - loss: 0.6812 - val_ac
curacy: 0.7192 - val_loss: 0.6248
Epoch 4/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6087 - loss: 0.6711 - val_ac
curacy: 0.7192 - val_loss: 0.6035
Epoch 5/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6231 - loss: 0.6622 - val_ac
curacy: 0.7231 - val_loss: 0.5798
Epoch 6/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6442 - loss: 0.6459 - val_ac
curacy: 0.7308 - val_loss: 0.5549
Epoch 7/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6500 - loss: 0.6331 - val_ac
curacy: 0.7423 - val_loss: 0.5268
Epoch 8/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6740 - loss: 0.6132 - val_ac
curacy: 0.7692 - val_loss: 0.4993
Epoch 9/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6981 - loss: 0.6025 - val_ac
curacy: 0.7769 - val_loss: 0.4741
Epoch 10/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6827 - loss: 0.5999 - val_ac
curacy: 0.7769 - val_loss: 0.4728
Epoch 11/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6981 - loss: 0.5886 - val_ac
curacy: 0.7885 - val_loss: 0.4595
Epoch 12/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7010 - loss: 0.5847 - val_ac
curacy: 0.7923 - val_loss: 0.4581
Epoch 13/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7067 - loss: 0.5777 - val_ac
curacy: 0.7923 - val_loss: 0.4526
Epoch 14/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7106 - loss: 0.5749 - val_ac
curacy: 0.7923 - val_loss: 0.4530
Epoch 15/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7125 - loss: 0.5654 - val_ac
curacy: 0.7923 - val_loss: 0.4570
Epoch 16/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7106 - loss: 0.5642 - val_ac
curacy: 0.7923 - val_loss: 0.4532
Epoch 17/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7154 - loss: 0.5617 - val_ac
curacy: 0.7923 - val_loss: 0.4539
Epoch 18/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7115 - loss: 0.5605 - val_ac
curacy: 0.7885 - val_loss: 0.4563
Epoch 19/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7154 - loss: 0.5570 - val_ac
curacy: 0.7962 - val_loss: 0.4614
Epoch 20/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7260 - loss: 0.5518 - val_ac
curacy: 0.7885 - val_loss: 0.4549
Epoch 21/30
```

```
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7240 - loss: 0.5548 - val_ac
curacy: 0.7885 - val_loss: 0.4634
Epoch 22/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7202 - loss: 0.5520 - val_ac
curacy: 0.7885 - val_loss: 0.4587
Epoch 23/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7308 - loss: 0.5499 - val_ac
curacy: 0.7962 - val_loss: 0.4534
Epoch 24/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7288 - loss: 0.5390 - val_ac
curacy: 0.7885 - val_loss: 0.4627
Epoch 25/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7231 - loss: 0.5499 - val_ac
curacy: 0.7885 - val_loss: 0.4554
Epoch 26/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7356 - loss: 0.5399 - val_ac
curacy: 0.8000 - val_loss: 0.4507
Epoch 27/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7413 - loss: 0.5400 - val_ac
curacy: 0.8000 - val_loss: 0.4423
Epoch 28/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7327 - loss: 0.5414 - val_ac
curacy: 0.8000 - val_loss: 0.4487
Epoch 29/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7413 - loss: 0.5387 - val_ac
curacy: 0.8038 - val_loss: 0.4503
Epoch 30/30
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7442 - loss: 0.5283 - val_ac
curacy: 0.8038 - val_loss: 0.4489
```

6. Evaluate the LSTM

In [44]:

```python
from sklearn.metrics import classification_report, roc_auc_score

y_prob = model.predict(X_test).ravel()
y_pred = (y_prob >= 0.5).astype(int)

print("LSTM classification report:")
print(classification_report(y_test, y_pred))
print("LSTM ROC AUC:", roc_auc_score(y_test, y_prob))
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
LSTM classification report:
              precision    recall  f1-score   support

           0       0.48      0.29      0.36        45
           1       0.89      0.95      0.92       280

    accuracy                           0.86       325
   macro avg       0.69      0.62      0.64       325
weighted avg       0.84      0.86      0.84       325

LSTM ROC AUC: 0.6388888888888888
```