In [1]:
```
# importing necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.multioutput import MultiOutputClassifier
from sklearn.metrics import classification_report

from sklearn.preprocessing import MinMaxScaler
import joblib
```

In [2]:
```
# --------------------------------
# STEP 1: LOAD AND PREPROCESS DATA
# --------------------------------

# Load the dataset (update the filename accordingly)
df = pd.read_csv("irrigation_machine.csv")
```

In [3]:
```
# first 5 rows to be printed, df.tail()
df.head()
```

Out[3]:

	Unnamed: 0	sensor_0	sensor_1	sensor_2	sensor_3	sensor_4	sensor_5	sensor_6	s
0	0	1.0	2.0	1.0	7.0	0.0	1.0	1.0	
1	1	5.0	1.0	3.0	5.0	2.0	2.0	1.0	
2	2	3.0	1.0	4.0	3.0	4.0	0.0	1.0	
3	3	2.0	2.0	4.0	3.0	5.0	0.0	3.0	
4	4	4.0	3.0	3.0	2.0	5.0	1.0	3.0	

5 rows × 24 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 24 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  2000 non-null   int64  
 1   sensor_0    2000 non-null   float64
 2   sensor_1    2000 non-null   float64
 3   sensor_2    2000 non-null   float64
 4   sensor_3    2000 non-null   float64
 5   sensor_4    2000 non-null   float64
 6   sensor_5    2000 non-null   float64
 7   sensor_6    2000 non-null   float64
 8   sensor_7    2000 non-null   float64
 9   sensor_8    2000 non-null   float64
 10  sensor_9    2000 non-null   float64
 11  sensor_10   2000 non-null   float64
 12  sensor_11   2000 non-null   float64
 13  sensor_12   2000 non-null   float64
 14  sensor_13   2000 non-null   float64
 15  sensor_14   2000 non-null   float64
 16  sensor_15   2000 non-null   float64
 17  sensor_16   2000 non-null   float64
 18  sensor_17   2000 non-null   float64
 19  sensor_18   2000 non-null   float64
 20  sensor_19   2000 non-null   float64
 21  parcel_0    2000 non-null   int64  
 22  parcel_1    2000 non-null   int64  
 23  parcel_2    2000 non-null   int64  
dtypes: float64(20), int64(4)
memory usage: 375.1 KB
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Unnamed: 0', 'sensor_0', 'sensor_1', 'sensor_2', 'sensor_3',
       'sensor_4', 'sensor_5', 'sensor_6', 'sensor_7', 'sensor_8', 'sensor_9
       'sensor_10', 'sensor_11', 'sensor_12', 'sensor_13', 'sensor_14',
       'sensor_15', 'sensor_16', 'sensor_17', 'sensor_18', 'sensor_19',
       'parcel_0', 'parcel_1', 'parcel_2'],
      dtype='object')
```

In [6]:
```
df = df.drop('Unnamed: 0', axis=1)
df.head()
```

Out[6]:

	sensor_0	sensor_1	sensor_2	sensor_3	sensor_4	sensor_5	sensor_6	sensor_7	se
0	1.0	2.0	1.0	7.0	0.0	1.0	1.0	4.0	
1	5.0	1.0	3.0	5.0	2.0	2.0	1.0	2.0	
2	3.0	1.0	4.0	3.0	4.0	0.0	1.0	6.0	
3	2.0	2.0	4.0	3.0	5.0	0.0	3.0	2.0	
4	4.0	3.0	3.0	2.0	5.0	1.0	3.0	1.0	

5 rows × 23 columns

In [7]:
```
df.describe() # Statistics of the dataset
```

Out[7]:

	sensor_0	sensor_1	sensor_2	sensor_3	sensor_4	sensor_5	
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	20
mean	1.437000	1.659000	2.654500	2.674500	2.887500	1.411000	
std	1.321327	1.338512	1.699286	1.855875	1.816451	1.339394	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	1.000000	1.000000	1.000000	2.000000	0.000000	
50%	1.000000	1.000000	2.000000	2.000000	3.000000	1.000000	
75%	2.000000	2.000000	4.000000	4.000000	4.000000	2.000000	
max	8.000000	9.000000	10.000000	11.000000	12.000000	7.000000	

8 rows × 23 columns

In [8]:
```
# --------------------------------
# STEP 2: DEFINE FEATURES AND LABELS
# --------------------------------

X = df.iloc[:, 0:20]   # This gives you columns 0 to 19 (sensor_0 to sens

y = df.iloc[:, 20:]
```

In [9]:
```
X.sample(10)
```

Out[9]:

	sensor_0	sensor_1	sensor_2	sensor_3	sensor_4	sensor_5	sensor_6	sensor_7
1875	3.0	1.0	5.0	0.0	7.0	0.0	5.0	9.0
1914	2.0	3.0	2.0	1.0	1.0	5.0	1.0	2.0
1460	2.0	1.0	2.0	2.0	2.0	4.0	6.0	3.0
92	0.0	0.0	3.0	3.0	2.0	2.0	5.0	7.0
634	1.0	3.0	4.0	4.0	3.0	2.0	2.0	9.0
1224	3.0	0.0	2.0	2.0	4.0	0.0	1.0	7.0
490	0.0	3.0	3.0	3.0	2.0	1.0	1.0	2.0
880	3.0	2.0	3.0	1.0	1.0	4.0	6.0	2.0
1912	1.0	3.0	5.0	5.0	4.0	0.0	1.0	9.0
1657	0.0	1.0	2.0	1.0	4.0	0.0	4.0	6.0

In [10]:
```
y.sample(10)
```

Out[10]:

	parcel_0	parcel_1	parcel_2
1451	1	1	0
1510	1	1	0
1799	1	1	0
1149	1	1	0
1588	1	0	0
269	0	0	0
357	1	1	0
861	0	0	0
1524	0	1	0
187	1	1	1

In [11]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 20 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   sensor_0   2000 non-null   float64
 1   sensor_1   2000 non-null   float64
 2   sensor_2   2000 non-null   float64
 3   sensor_3   2000 non-null   float64
 4   sensor_4   2000 non-null   float64
 5   sensor_5   2000 non-null   float64
 6   sensor_6   2000 non-null   float64
 7   sensor_7   2000 non-null   float64
 8   sensor_8   2000 non-null   float64
 9   sensor_9   2000 non-null   float64
 10  sensor_10  2000 non-null   float64
 11  sensor_11  2000 non-null   float64
 12  sensor_12  2000 non-null   float64
 13  sensor_13  2000 non-null   float64
 14  sensor_14  2000 non-null   float64
 15  sensor_15  2000 non-null   float64
 16  sensor_16  2000 non-null   float64
 17  sensor_17  2000 non-null   float64
 18  sensor_18  2000 non-null   float64
 19  sensor_19  2000 non-null   float64
dtypes: float64(20)
memory usage: 312.6 KB
```

In [12]:
```
y.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   parcel_0  2000 non-null   int64
 1   parcel_1  2000 non-null   int64
 2   parcel_2  2000 non-null   int64
dtypes: int64(3)
memory usage: 47.0 KB
```

In [13]: X

Out[13]:

	sensor_0	sensor_1	sensor_2	sensor_3	sensor_4	sensor_5	sensor_6	sensor_7
0	1.0	2.0	1.0	7.0	0.0	1.0	1.0	4.0
1	5.0	1.0	3.0	5.0	2.0	2.0	1.0	2.0
2	3.0	1.0	4.0	3.0	4.0	0.0	1.0	6.0
3	2.0	2.0	4.0	3.0	5.0	0.0	3.0	2.0
4	4.0	3.0	3.0	2.0	5.0	1.0	3.0	1.0
...	...	...	...	...	...	...	...	...
1995	4.0	1.0	2.0	2.0	1.0	1.0	1.0	2.0
1996	1.0	3.0	3.0	3.0	2.0	2.0	3.0	3.0
1997	1.0	3.0	3.0	1.0	1.0	4.0	8.0	1.0
1998	2.0	1.0	0.0	2.0	2.0	0.0	1.0	3.0
1999	0.0	1.0	4.0	1.0	2.0	2.0	6.0	8.0

2000 rows × 20 columns

In [14]:
```
X.shape, y.shape
```

Out[14]:
```
((2000, 20), (2000, 3))
```

In [15]:
```
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
X_scaled
```

Out[15]:
```
array([[0.125     , 0.22222222, 0.1       , ..., 0.09090909, 0.9       ,
        0.28571429],
       [0.625     , 0.11111111, 0.3       , ..., 0.18181818, 0.2       ,
        1.        ],
       [0.375     , 0.11111111, 0.4       , ..., 0.27272727, 0.1       ,
        0.        ],
       ...,
       [0.125     , 0.33333333, 0.3       , ..., 0.36363636, 0.1       ,
        0.        ],
       [0.25      , 0.11111111, 0.        , ..., 0.        , 0.3       ,
        0.        ],
       [0.        , 0.11111111, 0.4       , ..., 0.45454545, 0.2       ,
        0.14285714]])
```

In [16]:
```
# ------------------------------
# STEP 3: TRAIN-TEST SPLIT
# ------------------------------

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_siz
```

In [17]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[17]:
```
((1600, 20), (400, 20), (1600, 3), (400, 3))
```

In [18]:
```
# ------------------------------
# STEP 4: TRAIN CLASSIFIER
# ------------------------------

# Use MultiOutputClassifier to handle multi-label classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.multioutput import MultiOutputClassifier

# Custom hyperparameters for RandomForest
rf = RandomForestClassifier(
    n_estimators=200,         # Number of trees
    max_depth=10,             # Maximum depth of each tree
    min_samples_split=4,      # Minimum samples to split a node
    min_samples_leaf=2,       # Minimum samples per leaf
    max_features='sqrt',      # Number of features to consider at each sp
    random_state=42
)

# Wrap it with MultiOutputClassifier
model = MultiOutputClassifier(rf)

# Train the model
model.fit(X_train, y_train)
```

Out[18]:

MultiOutputClassifier

estimator: RandomForestClassifier

RandomForestClassifier

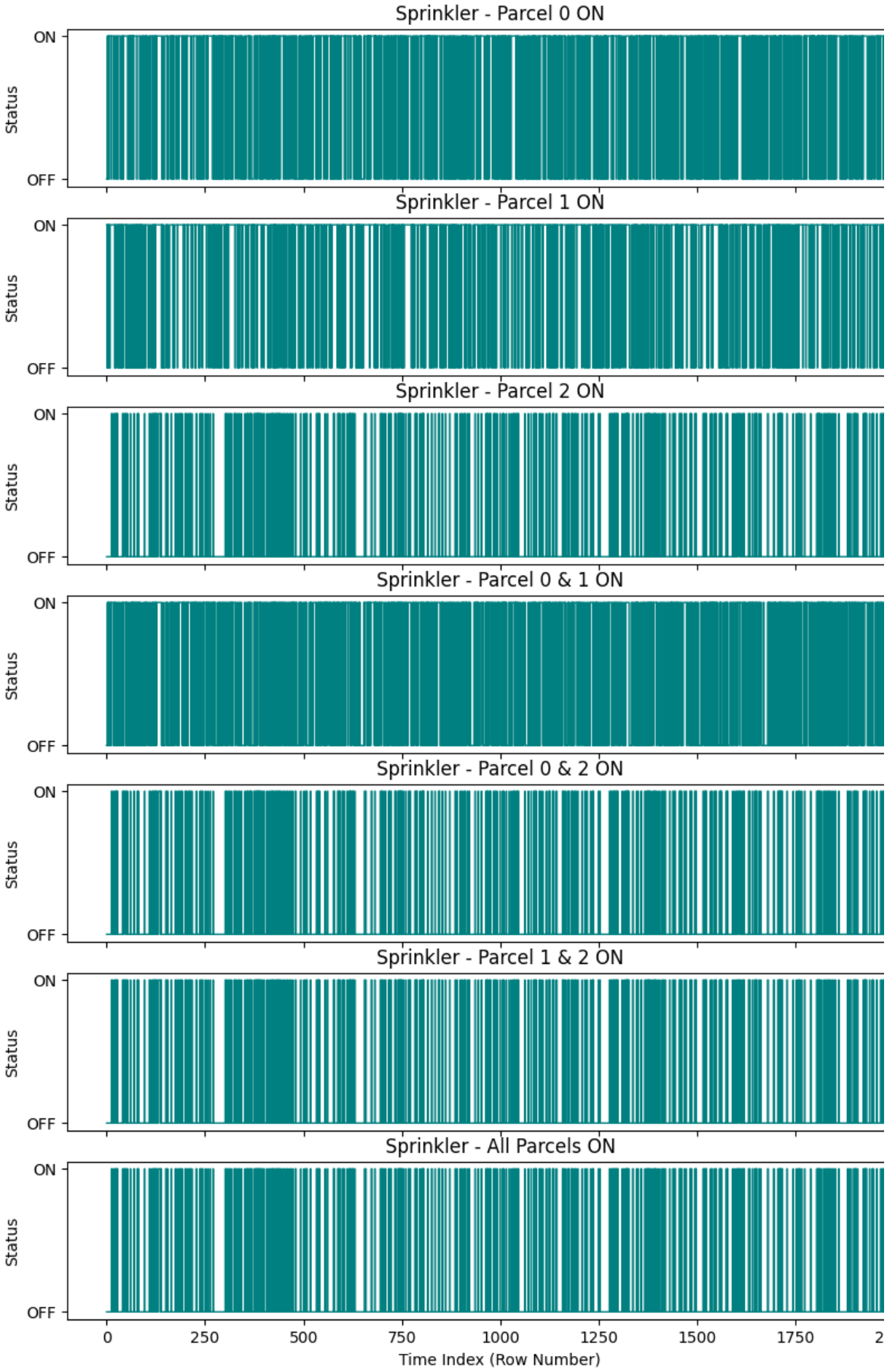
In [19]:
```
# ------------------------------
# STEP 5: EVALUATE MODEL
# ------------------------------

y_pred = model.predict(X_test)
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=y.columns))
```

```
Classification Report:
              precision    recall  f1-score   support

    parcel_0       0.87      0.93      0.90       256
    parcel_1       0.91      0.97      0.94       304
    parcel_2       0.93      0.48      0.64        87

   micro avg       0.90      0.89      0.89       647
   macro avg       0.91      0.80      0.83       647
weighted avg       0.90      0.89      0.88       647
 samples avg       0.82      0.79      0.79       647


/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packag
sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is
defined and being set to 0.0 in samples with no predicted labels. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packag
sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is il
defined and being set to 0.0 in samples with no true labels. Use `zero_divisi
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packag
sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: F-score is i
defined and being set to 0.0 in samples with no true nor predicted labels. Us
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

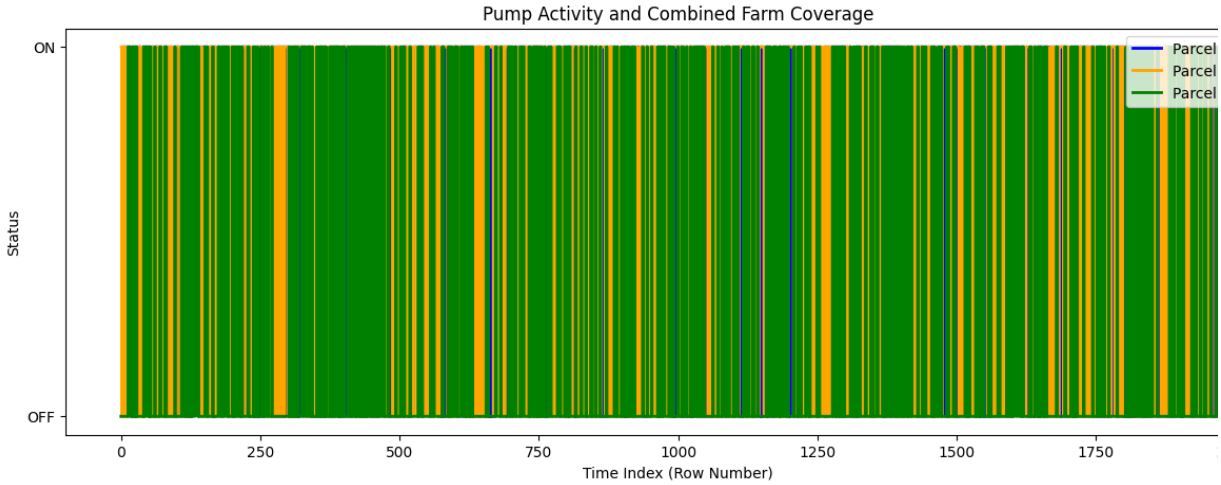
In [20]:
```
print(df[['parcel_0', 'parcel_1', 'parcel_2']].sum())
```

```
parcel_0    1271
parcel_1    1461
parcel_2     424
dtype: int64
```

In [21]:
```
import matplotlib.pyplot as plt

# Define parcel activation conditions with descriptive labels
conditions = {
    "Parcel 0 ON": df['parcel_0'],
    "Parcel 1 ON": df['parcel_1'],
    "Parcel 2 ON": df['parcel_2'],
    "Parcel 0 & 1 ON": df['parcel_0'] & df['parcel_1'],
    "Parcel 0 & 2 ON": df['parcel_0'] & df['parcel_2'],
    "Parcel 1 & 2 ON": df['parcel_1'] & df['parcel_2'],
    "All Parcels ON": df['parcel_0'] & df['parcel_1'] & df['parcel_2'],
}

# Create vertically stacked subplots (one for each condition)
fig, axs = plt.subplots(nrows=len(conditions), figsize=(10,15), sharex=Tru

# Loop through each condition to plot corresponding square wave
for ax, (title, condition) in zip(axs, conditions.items()):
    ax.step(df.index, condition.astype(int), where='post', linewidth=1, c
    ax.set_title(f"Sprinkler - {title}")
    ax.set_ylabel("Status")
    ax.set_yticks([0, 1])
    ax.set_yticklabels(['OFF', 'ON'])


# Label x-axis on the last subplot
axs[-1].set_xlabel("Time Index (Row Number)")

# Plot
plt.show()
```


Sprinkler - Parcel 0 ON
Sprinkler - Parcel 1 ON
Sprinkler - Parcel 2 ON
Sprinkler - Parcel 0 & 1 ON
Sprinkler - Parcel 0 & 2 ON
Sprinkler - Parcel 1 & 2 ON
Sprinkler - All Parcels ON
Status
ON
OFF
0
250
500
750
1000
1250
1500
1750
Time Index (Row Number)

In [22]:
```
# Calculate combined activity of all pumps (overlap)
any_pump_on = (df['parcel_0'] == 1) | (df['parcel_1'] == 1) | (df['parcel_

plt.figure(figsize=(15, 5))

# Plot individual pump statuses
plt.step(df.index, df['parcel_0'], where='post', linewidth=2, label='Parce
plt.step(df.index, df['parcel_1'], where='post', linewidth=2, label='Parce
plt.step(df.index, df['parcel_2'], where='post', linewidth=2, label='Parce

plt.title("Pump Activity and Combined Farm Coverage")
plt.xlabel("Time Index (Row Number)")
plt.ylabel("Status")
plt.yticks([0, 1], ['OFF', 'ON'])
plt.legend(loc='upper right')
plt.show()
```



In [23]:
```
import joblib
from sklearn.pipeline import Pipeline

joblib.dump(model, "Farm_Irrigation_System.pkl")
```

Out[23]:
```
['Farm_Irrigation_System.pkl']
```