Final Project
Lumpy Diseaseclassification
(IBM-Data Science)

Team Members:

1- Rana Hamed
2- Heba Ibrahim
3- Beshoy Ageeb
4- Eman Mansour

Project Main points

EDA & Pre-processing

- Collect data
- Data Cleaning
- Explore the Data (Exploratory Data Analysis - EDA)
- Handle imbalance dataset
- Scaling dataset

Model building and evaluation

- Splitting the Dataset
- Training the Model
- - Model Evaluation
- - Model Optimization
- - Testing & Validation
- - Feature Importance
- - Model Selection (from many models)
- - Deployment → Streamlit

Lumpy skin disease- مرض الجلد العقدي

Lumpy Skin Disease (LSD) is a viral disease that affects cattle, causing significant economic losses in the livestock industry. The disease is caused by the *Lumpy Skin Disease Virus* (LSDV), which belongs to the genus *Capripoxvirus* in the family *Poxviridae*. LSD primarily affects cattle but can also infect other bovine species.

Data Collection

kaggle

https://www.kaggle.com/datasets/saurabhshahane/lumpy-skin-disease-dataset

kaggle

Search

Create

- Home
- Competitions
- Datasets
- Models
- Code
- Discussions
- Learn
- More

Your Work

VIEWED

Lumpy Disease - E...

SAURABH SHAHANE · UPDATED 2 YEARS AGO

60 | New Notebook | Download (697 kB)

Lumpy Skin Disease Dataset

Skin Disease Detection using Machine Learning

Data Card | Code (12) | Discussion (3) | Suggestions (0)

About Dataset

Context

Assessing machine learning techniques in forecasting Lumpy Skin Disease occurrence based on meteorological and geospatial features - dataset

Acknowledgements

Afshari Safavi, Ehsanallah (2021), “Lumpy Skin disease dataset”, Mendeley Data, V1, doi: 10.17632/7pyhbzb2n9.1

Usability
8.24

License
Attribution 4.0 International (CC ...

Update frequency
Daily

Tags

Dataset Review

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	x	y	region	country	reporting	cld	dtr	frs	pet	pre	tmn	tmp	tmx	vap	wet	elevation	dominant_land_cover	X5_Ct_2010_Da	X5_Bf_2010_Da	lumpy
2	90.38093	22.43718	Asia	Bangladesh	10/9/2020	41.6	13	0	2.3	1.7	12.7	19.1	25.5	15.7	0	147	2	27970.9831	3691.74695	1
3	87.85498	22.98676	Asia	India	20/12/2019	40.5	13	0	2.4	0	13.2	19.8	26.5	16.3	0	145	2	25063.64669	671.3267014	1
4	85.27994	23.61018	Asia	India	20/12/2019	27.3	14	0.1	2.3	0.6	9.4	16.2	23	13	0.98	158	2	6038.477155	1426.839831	1
5	81.56451	43.88222	Asia	China	25/10/2019	45.3	13	31	0.4	8.8	-22.5	-16.1	-9.7	0.9	4.64	178	2	760.7033397	0	1
6	81.16106	43.83498	Asia	China	25/10/2019	38.8	13	31	0.4	10.5	-20.4	-13.8	-7.2	1.2	1.69	185	3	270.3674263	0	1
7	81.24834	43.96601	Asia	China	25/10/2019	38.8	13	31	0.4	10.5	-20.4	-13.8	-7.2	1.2	1.69	185	3	270.3674263	0	1
8	81.07417	43.83602	Asia	China	25/10/2019	38.8	13	31	0.4	10.5	-20.4	-13.8	-7.2	1.2	1.69	185	3	270.3674263	0	1
9	81.54713	43.68831	Asia	China	25/10/2019	45.3	13	31	0.4	8.8	-22.5	-16.1	-9.7	0.9	4.64	178	2	760.7033397	0	1
10	81.23957	43.59139	Asia	China	25/10/2019	38.8	13	31	0.4	10.5	-20.4	-13.8	-7.2	1.2	1.69	185	3	270.3674263	0	1
11	81.32415	43.97809	Asia	China	25/10/2019	38.8	13	31	0.4	10.5	-20.4	-13.8	-7.2	1.2	1.69	185	3	270.3674263	0	1
12	81.41911	43.76128	Asia	China	25/10/2019	38.8	13	31	0.4	10.5	-20.4	-13.8	-7.2	1.2	1.69	185	3	270.3674263	0	1
13	80.98576	43.8427	Asia	China	25/10/2019	43.6	13	31	0.4	11.1	-19	-12.3	-5.7	1.6	1.52	183	3	165.8319872	0	1
14	80.99889	43.54693	Asia	China	25/10/2019	43.6	13	31	0.4	11.1	-19	-12.3	-5.7	1.6	1.52	183	3	165.8319872	0	1
15	80.80078	43.80137	Asia	China	25/10/2019	43.6	13	31	0.4	11.1	-19	-12.3	-5.7	1.6	1.52	183	3	165.8319872	0	1
16	81.25112	43.80541	Asia	China	25/10/2019	38.8	13	31	0.4	10.5	-20.4	-13.8	-7.2	1.2	1.69	185	3	270.3674263	0	1
17	80.86655	43.83352	Asia	China	25/10/2019	43.6	13	31	0.4	11.1	-19	-12.3	-5.7	1.6	1.52	183	3	165.8319872	0	1
18	81.31971	43.7278	Asia	China	25/10/2019	38.8	13	31	0.4	10.5	-20.4	-13.8	-7.2	1.2	1.69	185	3	270.3674263	0	1
19	81.20617	43.8284	Asia	China	25/10/2019	38.8	13	31	0.4	10.5	-20.4	-13.8	-7.2	1.2	1.69	185	3	270.3674263	0	1
20	80.79963	44.04501	Asia	China	25/10/2019	43.8	13	31	0.5	11.9	-21.8	-15.2	-8.6	0.8	4.87	167	2	1044.908233	0	1
21	80.90709	43.99365	Asia	China	25/10/2019	43.6	13	31	0.4	11.1	-19	-12.3	-5.7	1.6	1.52	183	3	165.8319872	0	1
22	81.12118	43.15416	Asia	China	25/10/2019	45.1	14	31	0.5	7.1	-25.3	-18.4	-11.6	0	2.08	203	3	252.1598838	0	1
23	35.33016	32.79198	Asia	Israel	17/09/2019	57.2	8.1	0.2	1.7	114	9.2	13.2	17.3	11	15.38	167	2	1945.913549	0	1
24	35.30666	32.77668	Asia	Israel	17/09/2019	57.2	8.1	0.2	1.7	114	9.2	13.2	17.3	11	15.38	167	2	1945.913549	0	1
25	35.02872	32.4565	Asia	Israel	17/09/2019	54	8.3	0.7	1.7	103	8.2	12.3	16.5	10.4	14.83	191	3	523.3880539	0	1
26	35.43559	32.69445	Asia	Israel	17/09/2019	57.2	8.1	0.2	1.7	114	9.2	13.2	17.3	11	15.38	167	2	1945.913549	0	1
27	35.46579	32.81161	Asia	Israel	17/09/2019	57.2	8.1	0.2	1.7	114	9.2	13.2	17.3	11	15.38	167	2	1945.913549	0	1

Read data_lumpy skin

```python
df=pd.read_csv("C:\\Users\\heba\\Downloads\\Lumpy skin disease data.csv\\Lumpy skin disease data.csv")

df.head()
```

Python

	x	y	region	country	reportingDate	cld	dtr	frs	pet	pre	tmn	tmp	tmx	vap	wet	elevation	dominant_land_cover	X5_Ct_2010_Da	X5_Bf_2010_Da	lumpy
0	90.380931	22.437184	Asia	Bangladesh	10/9/2020	41.6	12.8	0.00	2.3	1.7	12.7	19.1	25.5	15.7	0.00	147	2	27970.983100	3691.746950	1
1	87.854975	22.986757	Asia	India	20/12/2019	40.5	13.3	0.00	2.4	0.0	13.2	19.8	26.5	16.3	0.00	145	2	25063.646690	671.326701	1
2	85.279935	23.610181	Asia	India	20/12/2019	27.3	13.6	0.08	2.3	0.6	9.4	16.2	23.0	13.0	0.98	158	2	6038.477155	1426.839831	1
3	81.564510	43.882221	Asia	China	25/10/2019	45.3	12.8	31.00	0.4	8.8	-22.5	-16.1	-9.7	0.9	4.64	178	2	760.703340	0.000000	1
4	81.161057	43.834976	Asia	China	25/10/2019	38.8	13.2	31.00	0.4	10.5	-20.4	-13.8	-7.2	1.2	1.69	185	3	270.367426	0.000000	1

Renaming the columns

	Longitude	Latitude	region	country	reportingDate	Cloud_Cover_Percentage	Diurnal_Temp_Range	Frost_Days	Evapotranspiration	Precipitation_Amount	Min_Temp	Mean_Temp	Max_Temp
0	90.380931	22.437184	Asia	Bangladesh	10/9/2020	41.6	12.8	0.00	2.3	1.7	12.7	19.1	25.5
1	87.854975	22.986757	Asia	India	20/12/2019	40.5	13.3	0.00	2.4	0.0	13.2	19.8	26.5
2	85.279935	23.610181	Asia	India	20/12/2019	27.3	13.6	0.08	2.3	0.6	9.4	16.2	23.0
3	81.564510	43.882221	Asia	China	25/10/2019	45.3	12.8	31.00	0.4	8.8	-22.5	-16.1	-9.7
4	81.161057	43.834976	Asia	China	25/10/2019	38.8	13.2	31.00	0.4	10.5	-20.4	-13.8	-7.2

Min_Temp	Mean_Temp	Max_Temp	Vapor_Pressure	Wet_Days_Count	elevation	Land_Cover	Buffalo_Population	Cattle_Population	lumpy
12.7	19.1	25.5	15.7	0.00	147	2	27970.983100	3691.746950	1
13.2	19.8	26.5	16.3	0.00	145	2	25063.646690	671.326701	1
9.4	16.2	23.0	13.0	0.98	158	2	6038.477155	1426.839831	1
-22.5	-16.1	-9.7	0.9	4.64	178	2	760.703340	0.000000	1
-20.4	-13.8	-7.2	1.2	1.69	185	3	270.367426	0.000000	1

Df.describe()

	Longitude	Latitude	Cloud_Cover_Percentage	Diurnal_Temp_Range	Frost_Days	Evapotranspiration	Precipitation_Amount	Min_Temp	Mean_Temp	Max_Temp	Vapor_Pressure
count	24803.000000	24803.000000	24803.000000	24803.000000	24803.000000	24803.000000	24803.000000	24803.000000	24803.000000	24803.000000	24803.000000
mean	79.221374	46.370056	59.452159	9.107777	23.978048	0.803487	26.271137	-15.794755	-11.227807	-6.681212	3.728230
std	43.338530	19.220555	19.423029	2.988448	11.518315	1.172915	33.630747	17.587685	17.989715	18.540915	4.952353
min	-179.750000	-28.750000	0.000000	2.000000	0.000000	0.000000	0.000000	-52.100000	-48.100000	-44.200000	0.000000
25%	45.083150	34.750000	43.800000	6.800000	23.210000	0.000000	5.900000	-30.100000	-25.500000	-20.900000	0.400000
50%	80.750000	48.250000	62.300000	8.300000	31.000000	0.200000	14.700000	-19.100000	-14.200000	-9.700000	1.500000
75%	109.750000	61.750000	75.300000	11.100000	31.000000	1.100000	33.400000	-2.200000	1.400000	4.900000	4.800000
max	179.750000	81.750000	98.700000	20.600000	31.000000	7.500000	341.900000	23.900000	28.500000	36.400000	28.600000

Min_Temp	Mean_Temp	Max_Temp	Vapor_Pressure	Wet_Days_Count	elevation	Land_Cover	Buffalo_Population	Cattle_Population	lumpy
24803.000000	24803.000000	24803.000000	24803.000000	24803.000000	24803.000000	24803.000000	24803.000000	24803.000000	24803.000000
-15.794755	-11.227807	-6.681212	3.728230	8.542482	164.769302	4.416119	629.129412	170.306057	0.122526
17.587685	17.989715	18.540915	4.952353	6.205199	19.679197	2.406231	2279.198775	1127.977653	0.327898
-52.100000	-48.100000	-44.200000	0.000000	0.000000	66.000000	0.000000	0.000000	0.000000	0.000000
-30.100000	-25.500000	-20.900000	0.400000	3.000000	152.000000	3.000000	2.513366	0.000000	0.000000
-19.100000	-14.200000	-9.700000	1.500000	8.020000	161.000000	4.000000	43.383823	0.000197	0.000000
-2.200000	1.400000	4.900000	4.800000	12.710000	176.000000	4.000000	386.124908	0.002094	0.000000
23.900000	28.500000	36.400000	28.600000	30.920000	249.000000	11.000000	167388.672700	56654.780150	1.000000

Click to add a breakpoint

```python
# Drop rows where Latitude is not between -90 and 90
df = df.drop(df[(df['Latitude'] < -90) | (df['Latitude'] > 90)].index)

# Drop rows where Longitude is not between -180 and 180
df = df.drop(df[(df['Longitude'] < -180) | (df['Longitude'] > 180)].index)

# Now, df will only contain rows where Latitude and Longitude are within valid ranges
```

1-df.shape =(24803, 20)
2-df.duplicated().sum()=608 do drop for these rows
3-df.isna().sum()

- region 90.0% from values in columns is missing
- country 90.0 % from values in columns is missing
- reportingDate 90.0%from values in columns is missing

Do drop for these columns
drop(Land_Cover column)

```
df["Land_Cover"].unique()
```

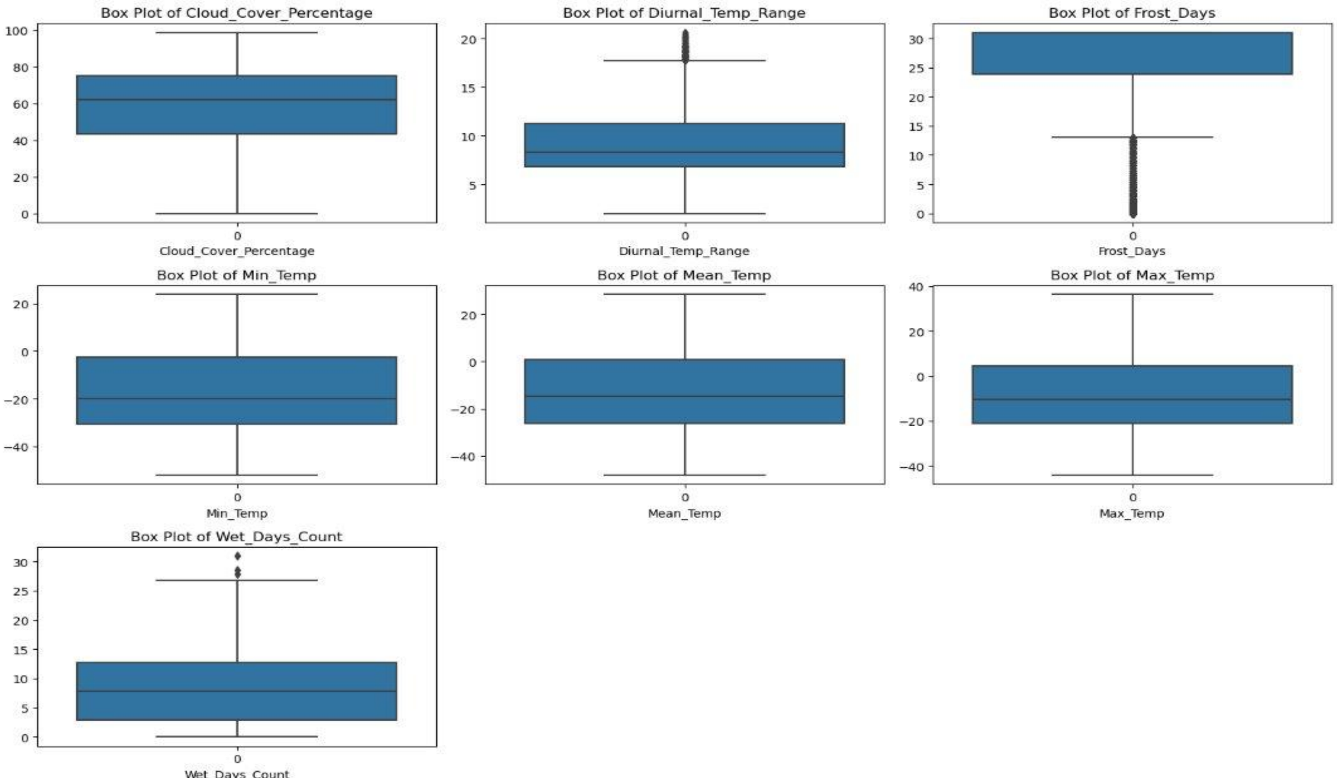
```
array([ 2,  3,  4,  9,  1,  5, 11,  8, 10,  6,  0,  7], dtype=int64)
```

I do not have any information that expresses these values

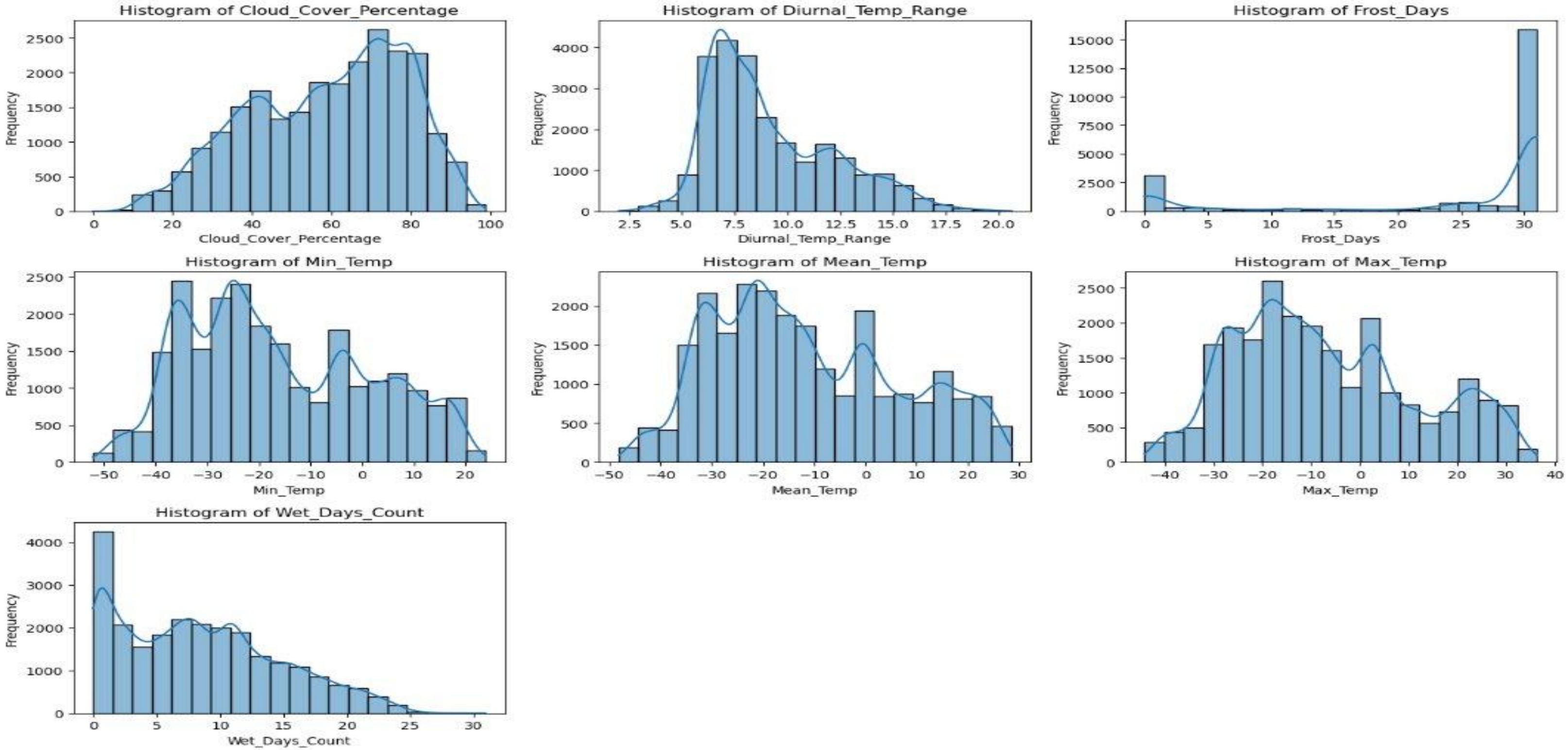

Plotting Points from Latitude and Longitude on the World Map

Handle Outliers

BOX_PLOT


Box Plot of Cloud_Cover_Percentage
Cloud_Cover_Percentage
Box Plot of Diurnal_Temp_Range
Diurnal_Temp_Range
Box Plot of Frost_Days
Frost_Days
Box Plot of Min_Temp
Min_Temp
Box Plot of Mean_Temp
Mean_Temp
Box Plot of Max_Temp
Max_Temp
Box Plot of Wet_Days_Count
Wet_Days_Count

HISTOGRAM


Histogram of Cloud_Cover_Percentage
Cloud_Cover_Percentage
Frequency
Histogram of Diurnal_Temp_Range
Diurnal_Temp_Range
Frequency
Histogram of Frost_Days
Frost_Days
Frequency
Histogram of Min_Temp
Min_Temp
Frequency
Histogram of Mean_Temp
Mean_Temp
Frequency
Histogram of Max_Temp
Max_Temp
Frequency
Histogram of Wet_Days_Count
Wet_Days_Count
Frequency

Handle outliers

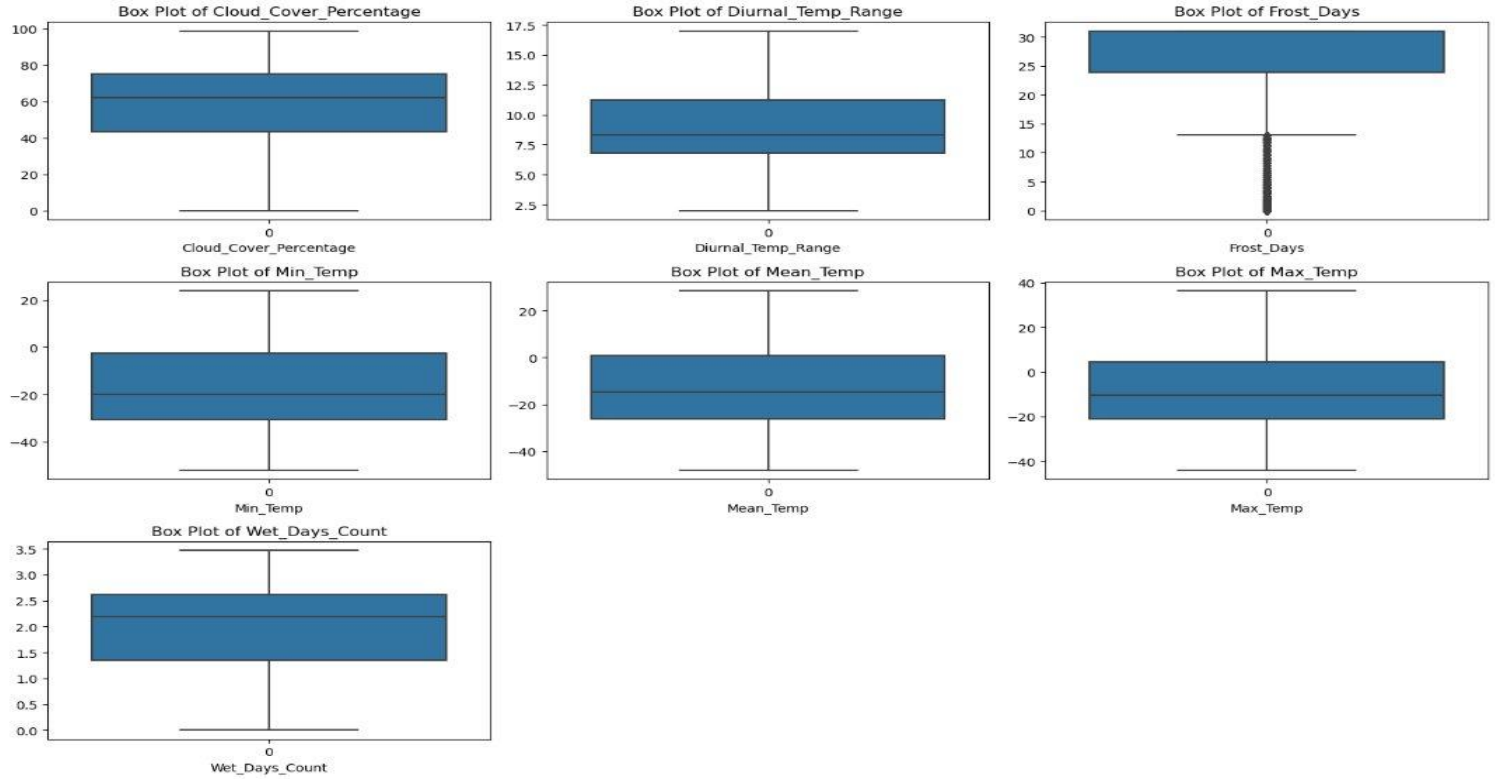
```
from scipy.stats import mstats

df["Diurnal_Temp_Range"]=df["Diurnal_Temp_Range"].apply(lambda x: 17 if x>17 else x)
df['Wet_Days_Count'] = np.log1p(df['Wet_Days_Count'])
df['Frost_Days'] = mstats.winsorize(df['Frost_Days'], limits=[0.01, 0.01])  # Adjust limits as needed
```

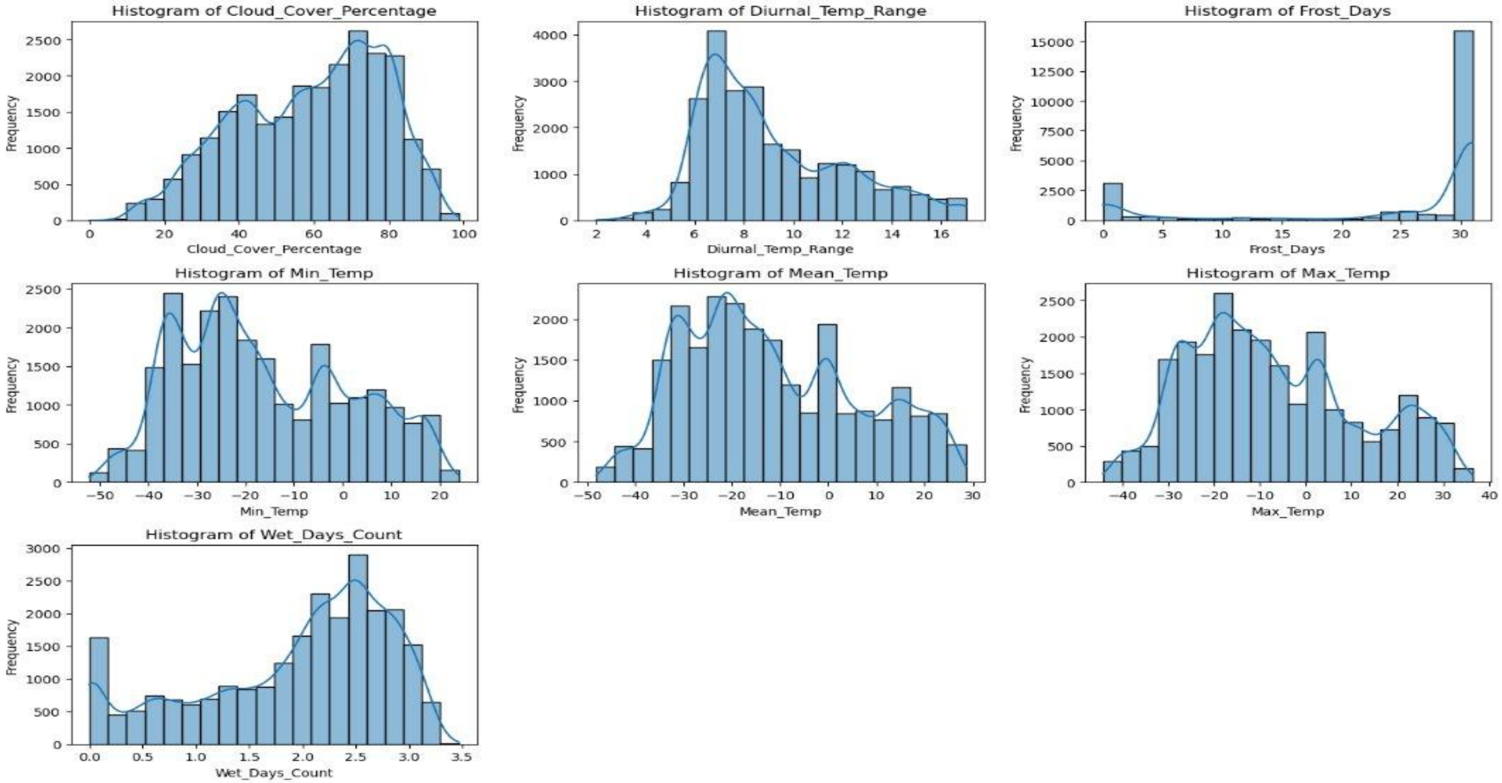
(parameter) x: Any

✓ 0.0s

BOX_PLOT

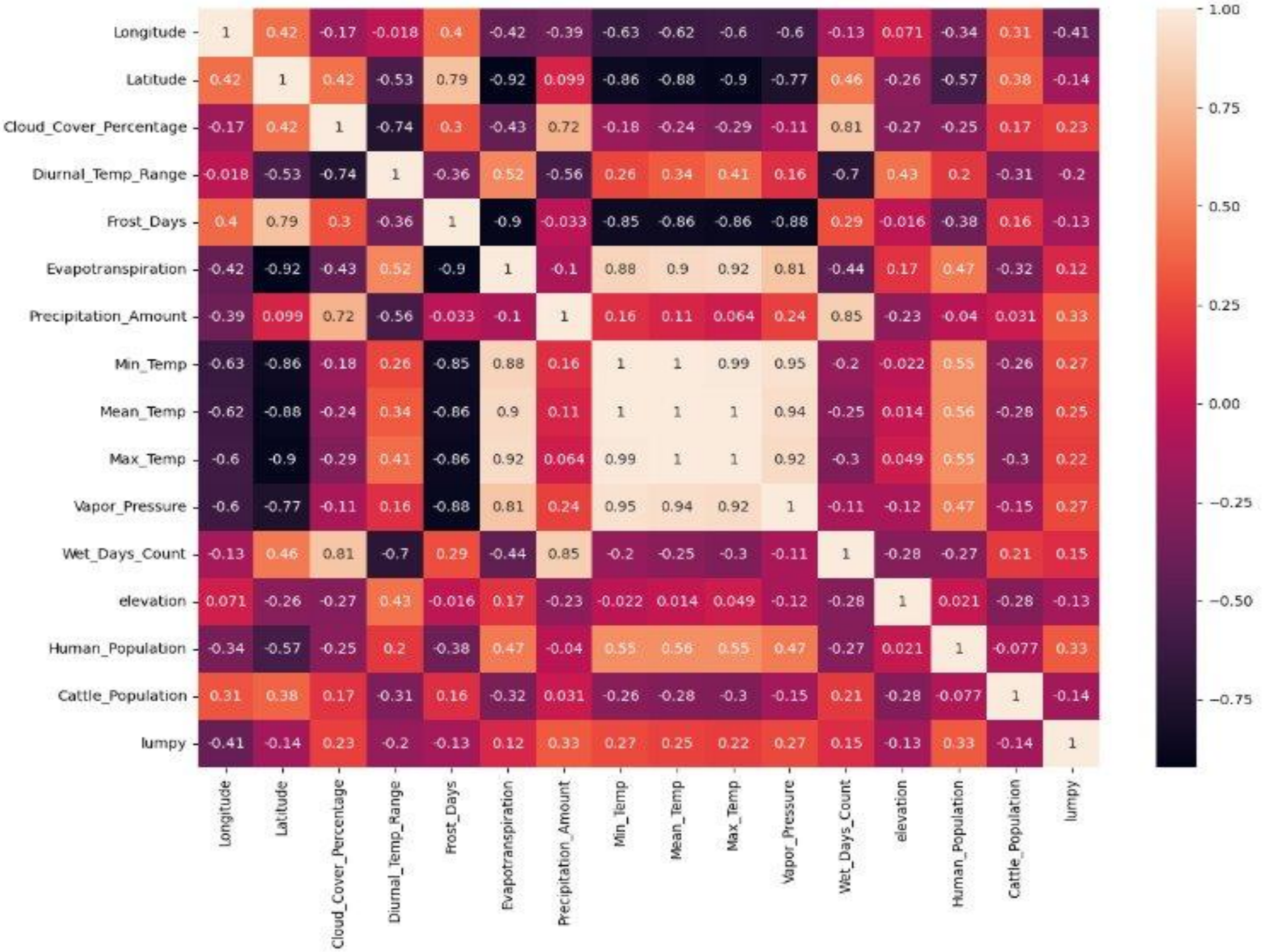

Box Plot of Cloud_Cover_Percentage
Cloud_Cover_Percentage
Box Plot of Diurnal_Temp_Range
Diurnal_Temp_Range
Box Plot of Frost_Days
Frost_Days
Box Plot of Min_Temp
Min_Temp
Box Plot of Mean_Temp
Mean_Temp
Box Plot of Max_Temp
Max_Temp
Box Plot of Wet_Days_Count
Wet_Days_Count

HISTOGRAM



Histogram of Cloud_Cover_Percentage
Frequency
Cloud_Cover_Percentage
Histogram of Diurnal_Temp_Range
Frequency
Diurnal_Temp_Range
Histogram of Frost_Days
Frequency
Frost_Days
Histogram of Min_Temp
Frequency
Min_Temp
Histogram of Mean_Temp
Frequency
Mean_Temp
Histogram of Max_Temp
Frequency
Max_Temp
Histogram of Wet_Days_Count
Frequency
Wet_Days_Count

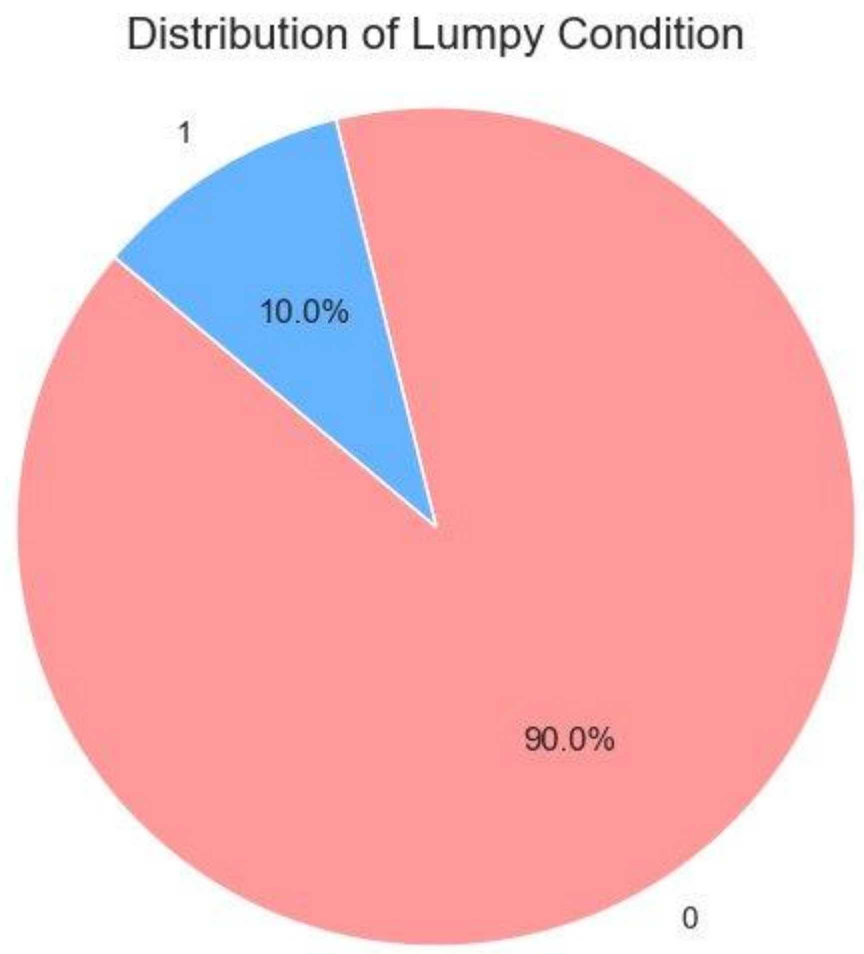
CORRELATION

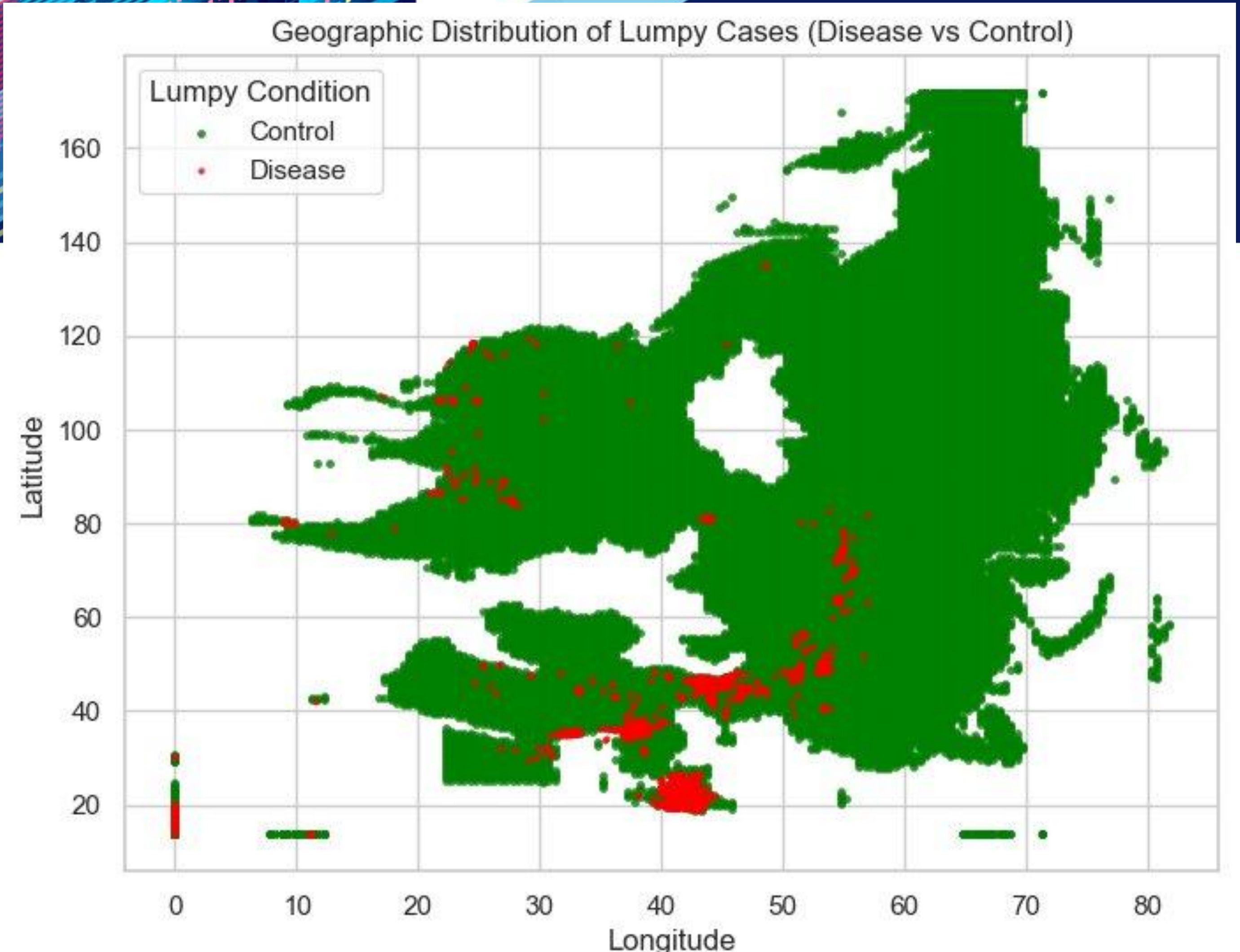


```
df["lumpy"].value_counts()
```

✓ 0.0s

```
lumpy
0    21764
1     2431
Name: count, dtype: int64
```

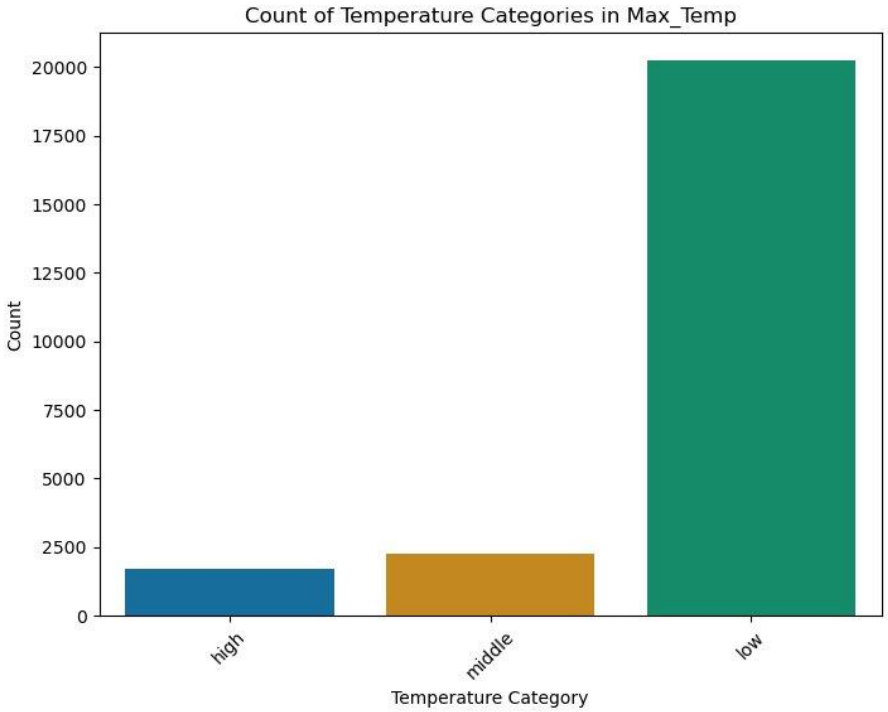

Distribution of Lumpy Condition
1
10.0%
90.0%
0

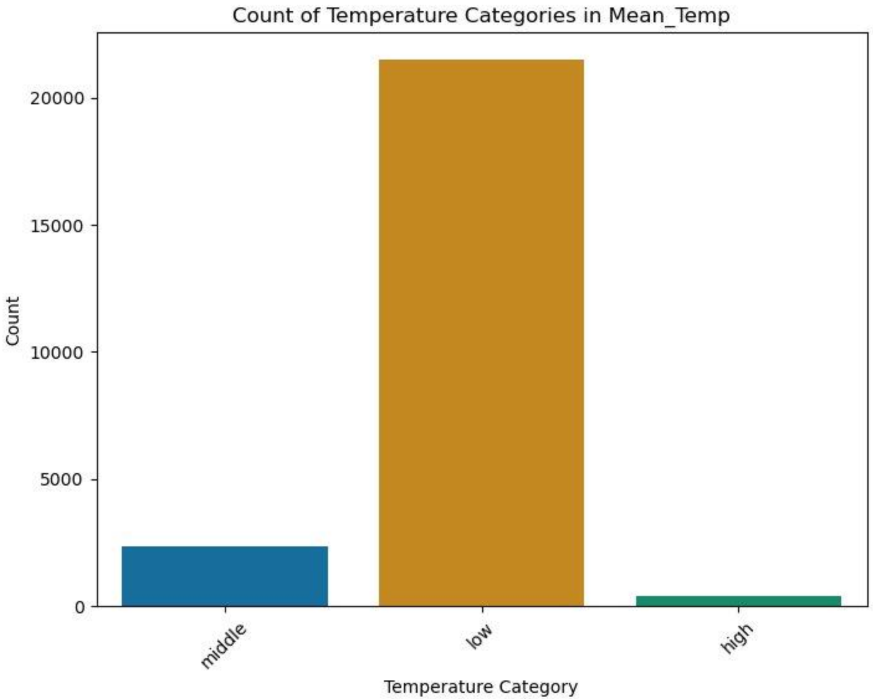


Geographic Distribution of Lumpy Cases (Disease vs Control)
Lumpy Condition
Control
Disease
Latitude
160
140
120
100
80
60
40
20
0
10
20
30
40
50
60
70
80
Longitude

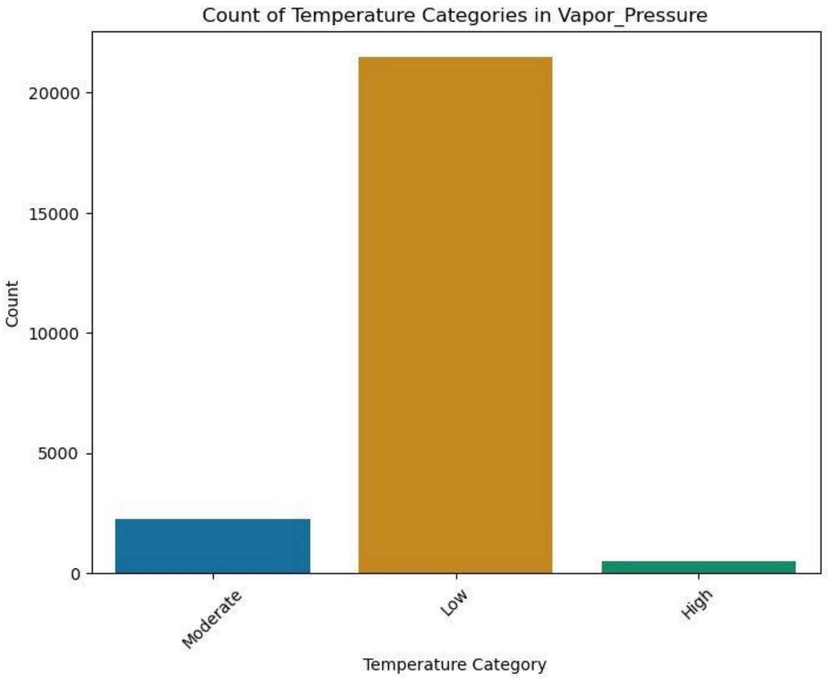
Data visualization

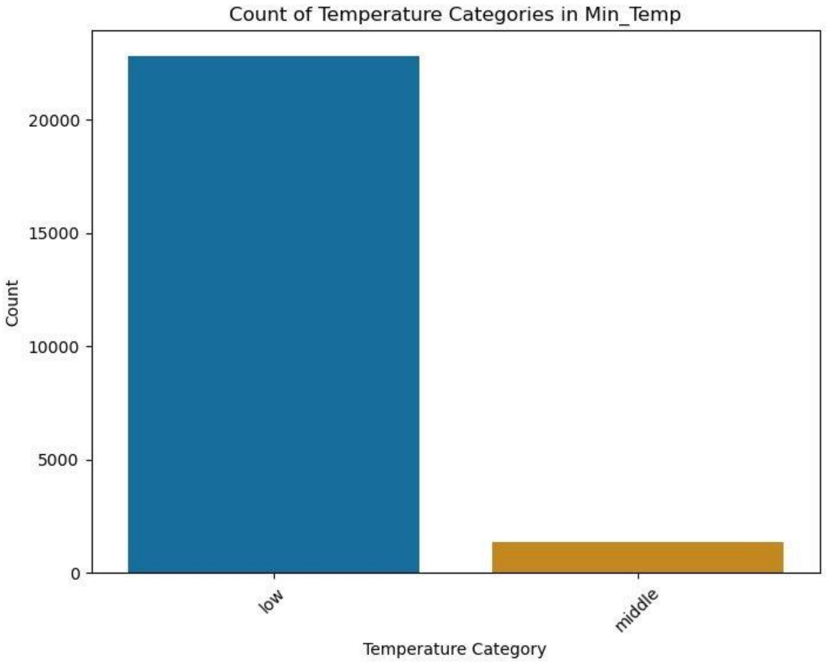
Univariate Analysis



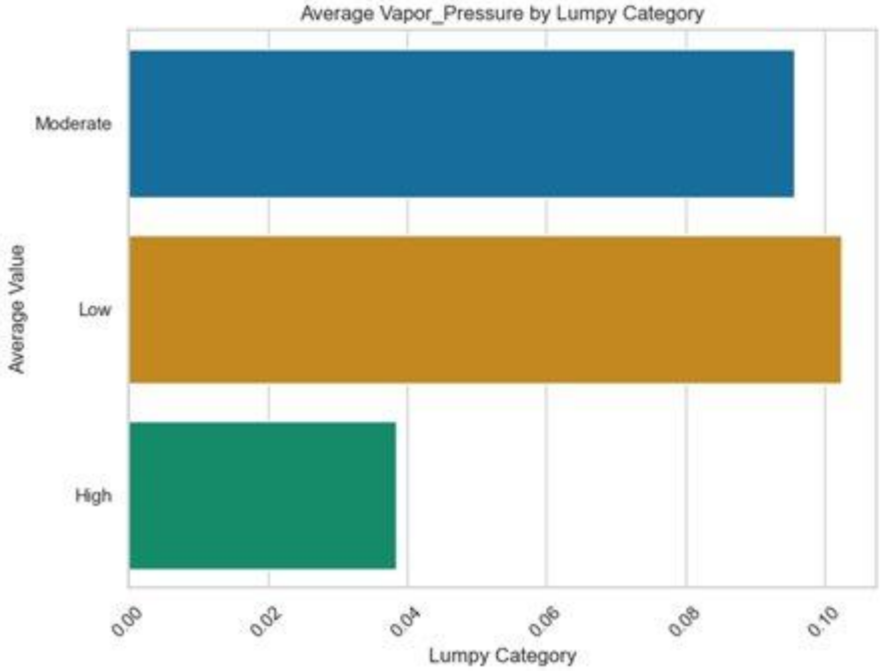


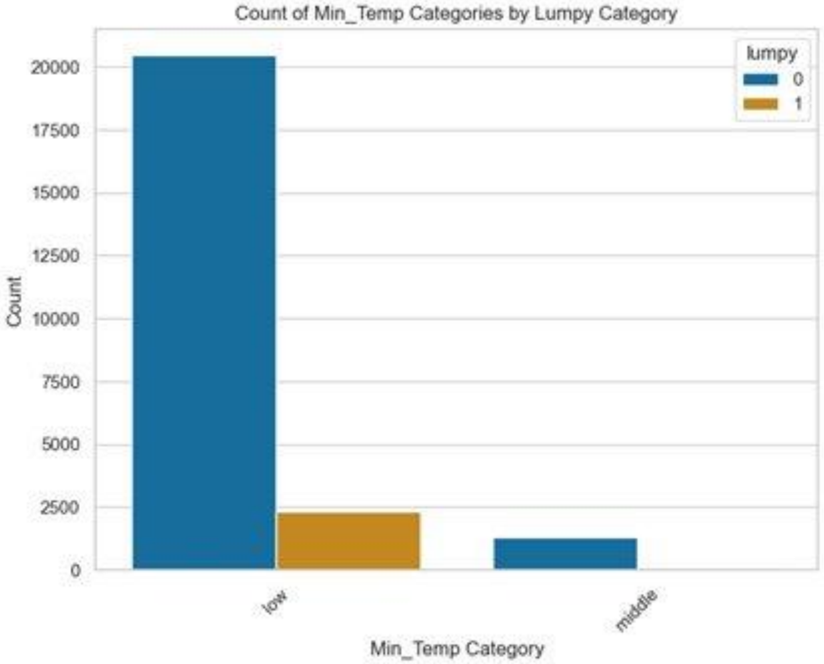
Univariate Analysis



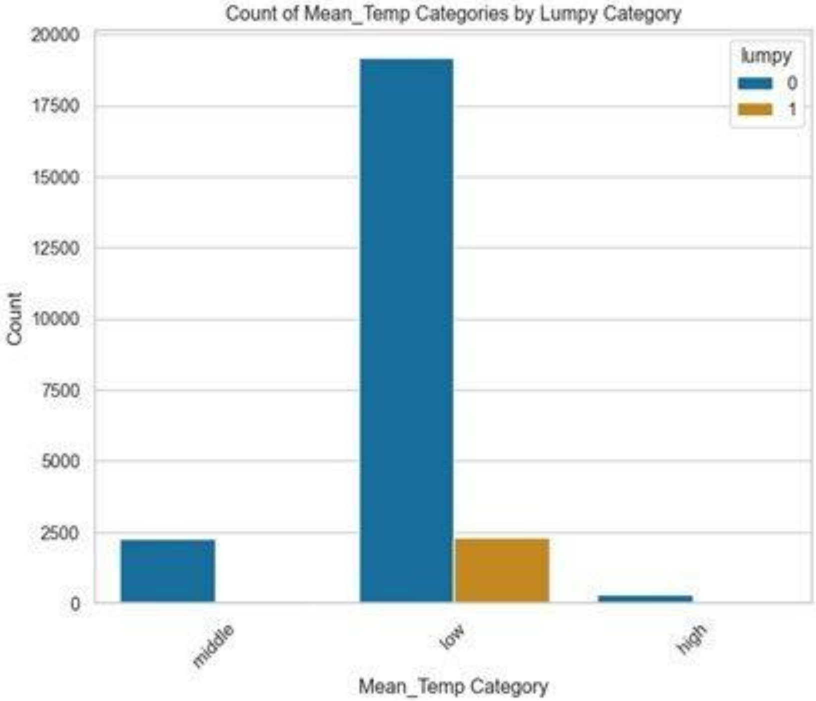


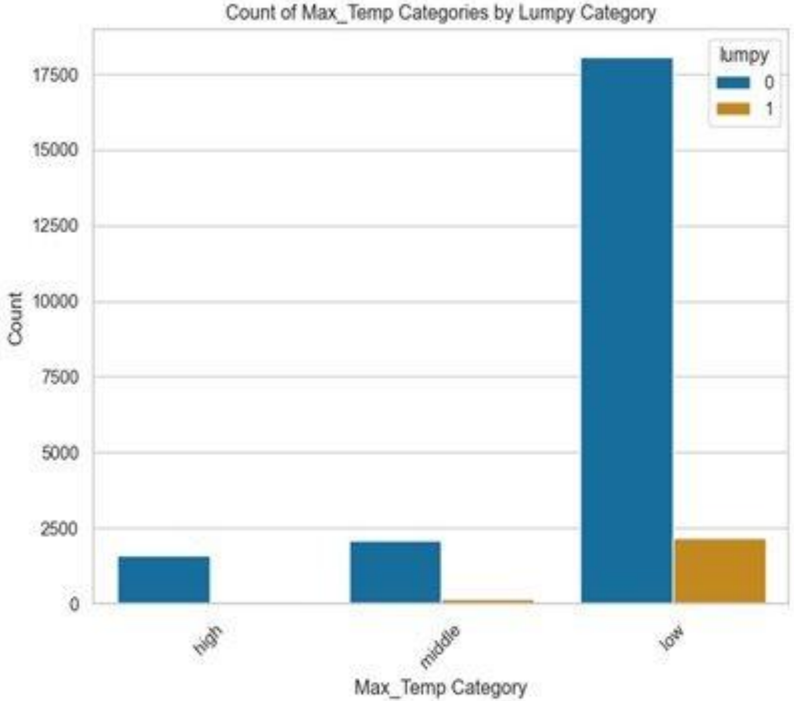
Bivariate Analysis

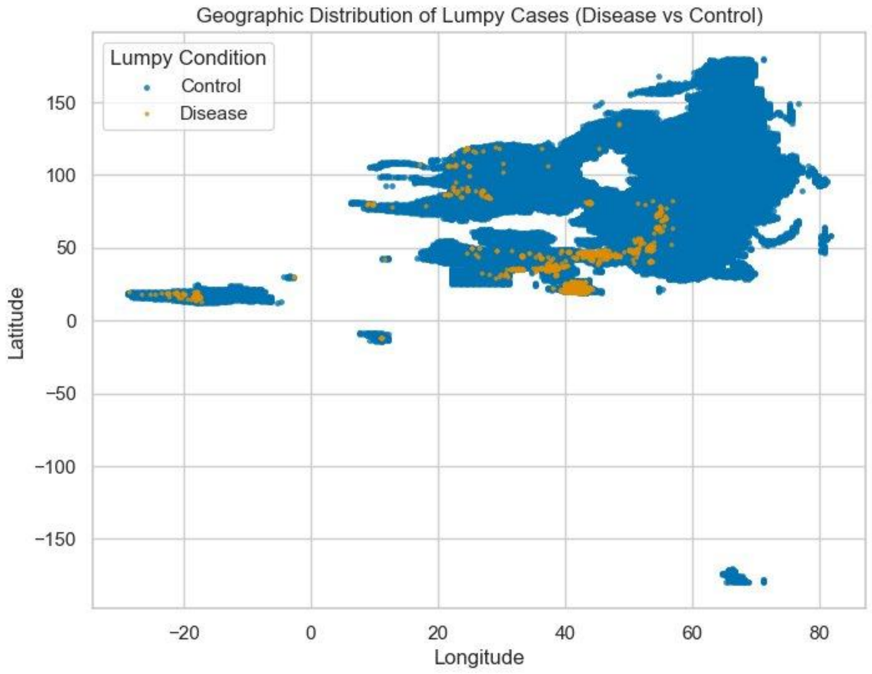




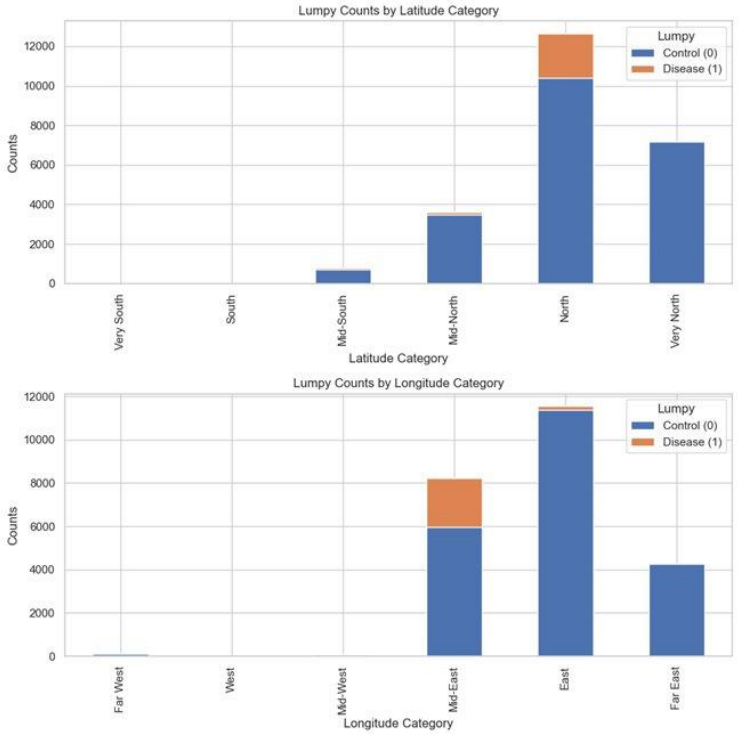
Bivariate Analysis







Geographic Distribution of Lumpy Cases (Disease vs Control)
Lumpy Condition
Control
Disease
Latitude
Longitude



Lumpy Counts by Latitude Category
Lumpy
Control (0)
Disease (1)
Counts
Very South
South
Mid-South
Mid-North
North
Very North
Latitude Category
Lumpy Counts by Longitude Category
Lumpy
Control (0)
Disease (1)
Counts
Far West
West
Mid-West
Mid-East
East
Far East
Longitude Category

Preprocessing

Split the data

```
# Define features and target variable

X = df.drop(["lumpy"], axis=1)
y = df["lumpy"]

# Split the data
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True, stratify=y, random_state=42)
```

✓ 0.0s

Handle imbalance_dataset

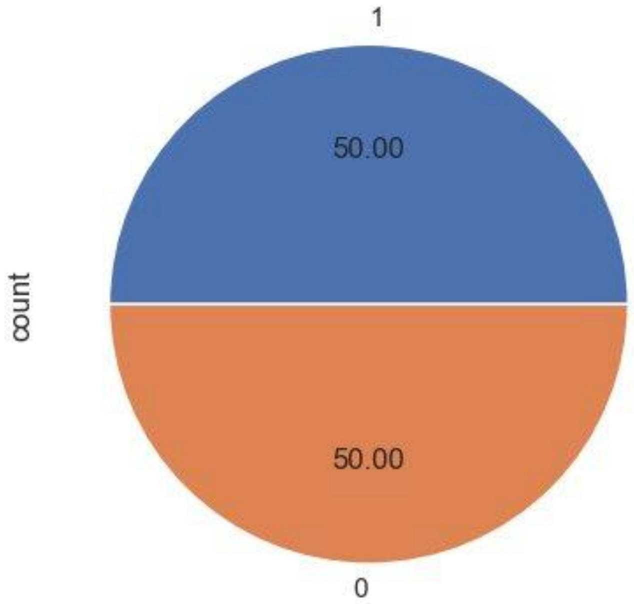
handle imbalance_dataset

```
from imblearn.over_sampling import RandomOverSampler
rus= RandomOverSampler(sampling_strategy=1)
x_res,y_res=rus.fit_resample(X,y)
print("------\n")
print(y_res.value_counts())
ax=y_res.value_counts().plot.pie(autopct='%.2f')
```

✓ 0.0s

```
------

lumpy
1    21764
0    21764
Name: count, dtype: int64
```



Scaling data

scaling

```
# Feature scaling
scaler = StandardScaler()
x_res_scaled = scaler.fit_transform(x_res)
x_test_scaled = scaler.transform(x_test)  # Scale test set using the same scaler
```

✓ 0.0s

Feature selection

```
# Feature selection
selector = SelectKBest(score_func=f_classif, k=10)  # You can change k to select fewer features
x_res_selected = selector.fit_transform(x_res_scaled, y_res)
x_test_selected = selector.transform(x_test_scaled)
```

✓ 0.0s

```
# Check the selected features' scores
feature_scores = selector.scores_
feature_names = X.columns
feature_importance = pd.DataFrame({'Feature': feature_names, 'Score': feature_scores})
```

✓ 0.0s

Display the scores

```
# Display the scores
print(feature_importance.sort_values(by='Score', ascending=False))
```

✓ 0.0s

```
                 Feature         Score
0              Longitude  29448.143626
6   Precipitation_Amount  14046.612499
7              Mean_Temp  12192.451461
2 Cloud_Cover_Percentage   9123.648789
3     Diurnal_Temp_Range   7060.960604
9         Wet_Days_Count   4158.718909
8         Vapor_Pressure   3102.139853
1               Latitude   2839.175684
4            Frost_Days   2354.807620
10             elevation   1548.932197
11     Buffalo_Population    480.392742
5     Evapotranspiration    279.897499
12      Cattle_Population    208.434680
```

Model building

Data splitting

```python
# Drop the specified columns from the DataFrame
df = df.drop(columns=['Longitude', 'Latitude'])
```

```python
# step 1: Define features and target variable

X = df.drop(["lumpy"], axis=1)
y = df["lumpy"]
# Step 2: Splitting the resampled data into train (60%), validation (20%), and test (20%) sets

X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4,random_state=42, stratify=y) # 60% training

X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,random_state=42, stratify=y_temp) # 20% validation, 20% testing
```

Solve imbalance in data

```
# Step 3: Handling imbalanced data with SMOTE (oversampling the minority class)
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

feature_selection

```python
# Step 4: Feature Selection using SelectKBest (ANOVA F-test)
# Select the top 3 features (you can adjust k to any number)
feature_selector = SelectKBest(score_func=f_classif, k=3)
X_train_selected = feature_selector.fit_transform(X_train, y_train)
X_val_selected = feature_selector.transform(X_val)
X_test_selected = feature_selector.transform(X_test)
```

```python
# Get the names of the selected features
selected_features = X.columns[feature_selector.get_support()]
print(selected_features)
```

```
Index(['Cloud_Cover_Percentage', 'Precipitation_Amount', 'Mean_Temp'], dtype='object')
```

Scaling data

```
# Step 5: Scaling Features
scaler = StandardScaler()
X_train_selected = scaler.fit_transform(X_train_selected)
X_val_selected = scaler.transform(X_val_selected)
X_test_selected = scaler.transform(X_test_selected)
```

1- Random Forest Model

```
# Step 6: Hyperparameter Tuning with GridSearchCV
# Random Forest Hyperparameter Tuning
rf_params = {
    'n_estimators': [100, 200, 300],
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
rf_model = RandomForestClassifier(random_state=42)
rf_grid = GridSearchCV(estimator=rf_model, param_grid=rf_params, cv=3, n_jobs=-1, verbose=2)
rf_grid.fit(X_train_selected, y_train)
best_rf_model = rf_grid.best_estimator_
```

```
Fitting 3 folds for each of 81 candidates, totalling 243 fits
```

```python
# Step 7: Validate the models
# Random Forest Validation
y_val_pred_rf = best_rf_model.predict(X_val_selected)
rf_accuracy = accuracy_score(y_val, y_val_pred_rf)
print(f"Random Forest Validation Accuracy: {rf_accuracy}")
print("Random Forest Validation Report:")
print(classification_report(y_val, y_val_pred_rf))
```

```
Random Forest Validation Accuracy: 0.9663153544120686
Random Forest Validation Report:
```

	precision	recall	f1-score	support
0	0.98	0.99	0.98	4353
1	0.87	0.79	0.82	486
accuracy			0.97	4839
macro avg	0.92	0.89	0.90	4839
weighted avg	0.97	0.97	0.97	4839

Random Forest Testing

```python
# Step 8: Testing the models
# Random Forest Testing
y_test_pred_rf = best_rf_model.predict(X_test_selected)
rf_test_accuracy = accuracy_score(y_test, y_test_pred_rf)
print(f"Random Forest Test Accuracy: {rf_test_accuracy}")
print("Random Forest Test Report:")
print(classification_report(y_test, y_test_pred_rf))
print("Random Forest Confusion Matrix:")
print(confusion_matrix(y_test, y_test_pred_rf))
```

```
Random Forest Test Accuracy: 0.9687952056209961
Random Forest Test Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      4353
           1       0.87      0.81      0.84       486

    accuracy                           0.97      4839
   macro avg       0.93      0.90      0.91      4839
weighted avg       0.97      0.97      0.97      4839

Random Forest Confusion Matrix:
[[4296   57]
 [  94  392]]
```

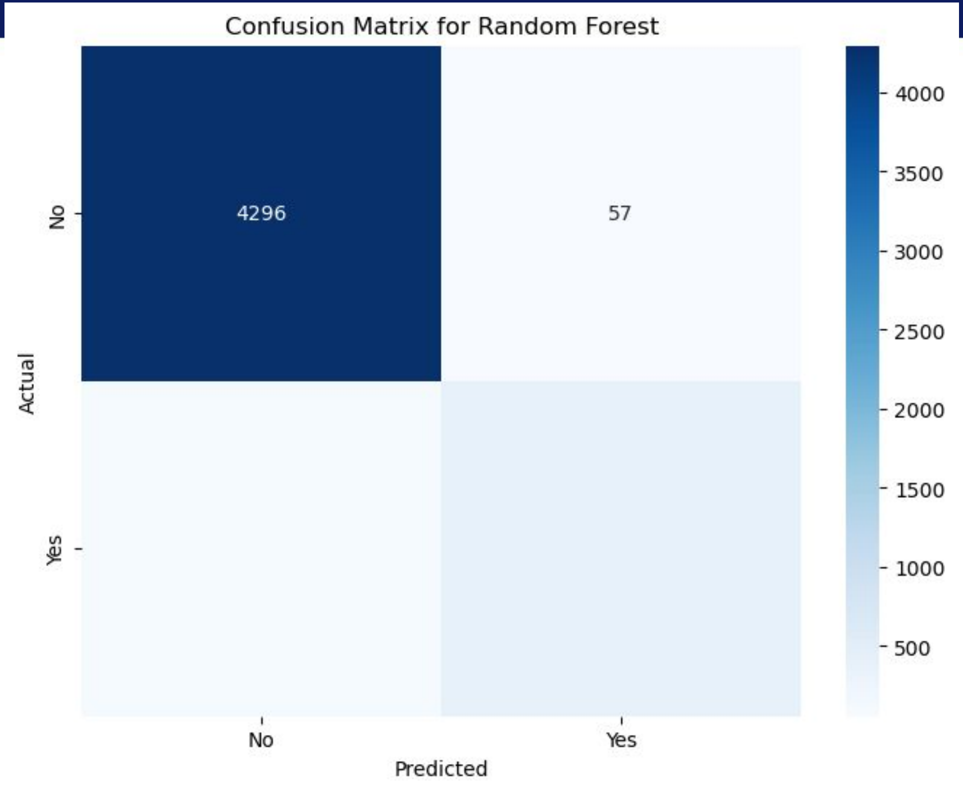
CONFUSION_MATRIX

True Negative (Top-left: 4323): The model correctly predicted "No Disease" when there was indeed no disease.

False Positive (Top-right: 30): The model incorrectly predicted "Lumpy Disease" when there was actually no disease.

False Negative (Bottom-left: 178): The model incorrectly predicted "No Disease" when there was actually "Lumpy Disease."

True Positive (Bottom-right: 308): The model correctly predicted "Lumpy Disease" when the disease was present.



2- XGBOOST

```
# XGBoost Hyperparameter Tuning
xgb_params = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0]
}
xgb_model = XGBClassifier(random_state=42, eval_metric='logloss')
xgb_grid = GridSearchCV(estimator=xgb_model, param_grid=xgb_params, cv=3, n_jobs=-1, verbose=2)
xgb_grid.fit(X_train_selected, y_train)
best_xgb_model = xgb_grid.best_estimator_
```

```
Fitting 3 folds for each of 243 candidates, totalling 729 fits
```

```python
# XGBoost Validation
y_val_pred_xgb = best_xgb_model.predict(X_val_selected)
xgb_accuracy = accuracy_score(y_val, y_val_pred_xgb)
print(f"XGBoost Validation Accuracy: {xgb_accuracy}")
print("XGBoost Validation Report:")
print(classification_report(y_val, y_val_pred_xgb))
```

```
XGBoost Validation Accuracy: 0.9617689605290349
XGBoost Validation Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.98      4353
           1       0.84      0.77      0.80       486

    accuracy                           0.96      4839
   macro avg       0.91      0.88      0.89      4839
weighted avg       0.96      0.96      0.96      4839
```

XGBoost Testing

```python
# XGBoost Testing
y_test_pred_xgb = best_xgb_model.predict(X_test_selected)
xgb_test_accuracy = accuracy_score(y_test, y_test_pred_xgb)
print(f"XGBoost Test Accuracy: {xgb_test_accuracy}")
print("XGBoost Test Report:")
print(classification_report(y_test, y_test_pred_xgb))
print("XGBoost Confusion Matrix:")
print(confusion_matrix(y_test, y_test_pred_xgb))
```

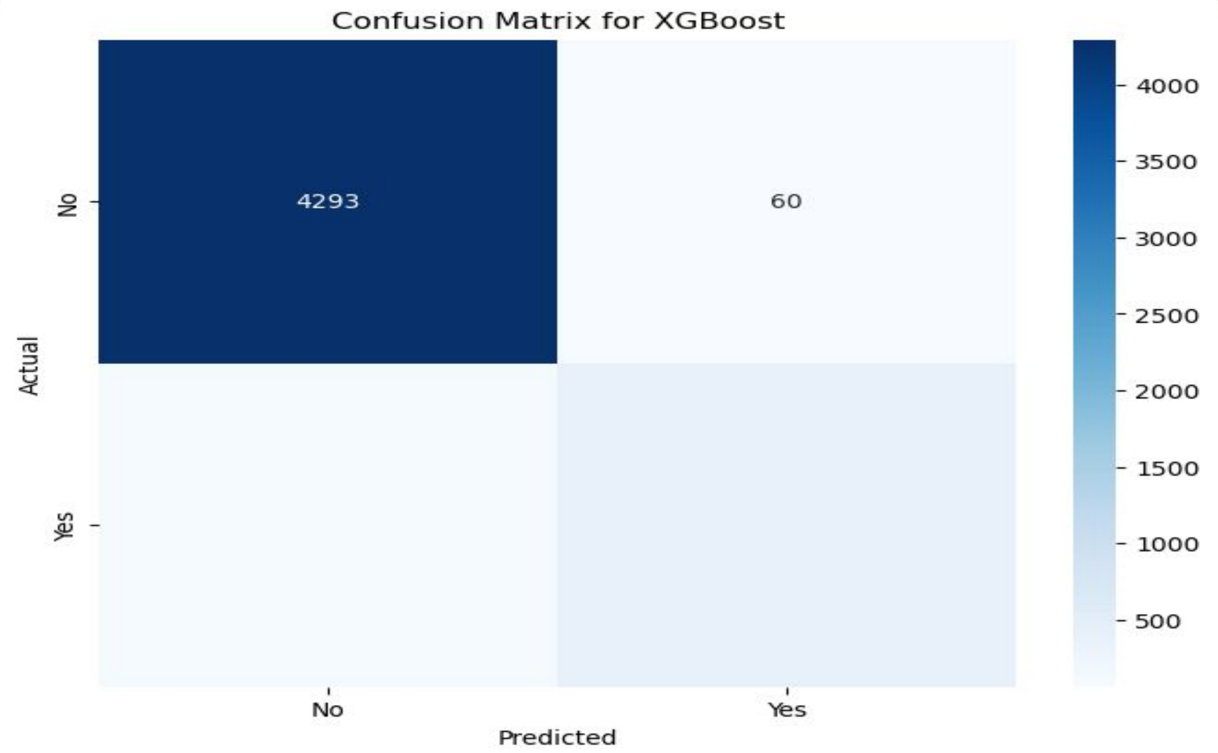
```
XGBoost Test Accuracy: 0.9669353172143005
XGBoost Test Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      4353
           1       0.87      0.79      0.83       486

    accuracy                           0.97      4839
   macro avg       0.92      0.89      0.91      4839
weighted avg       0.97      0.97      0.97      4839

XGBoost Confusion Matrix:
[[4293   60]
 [ 100  386]]
```

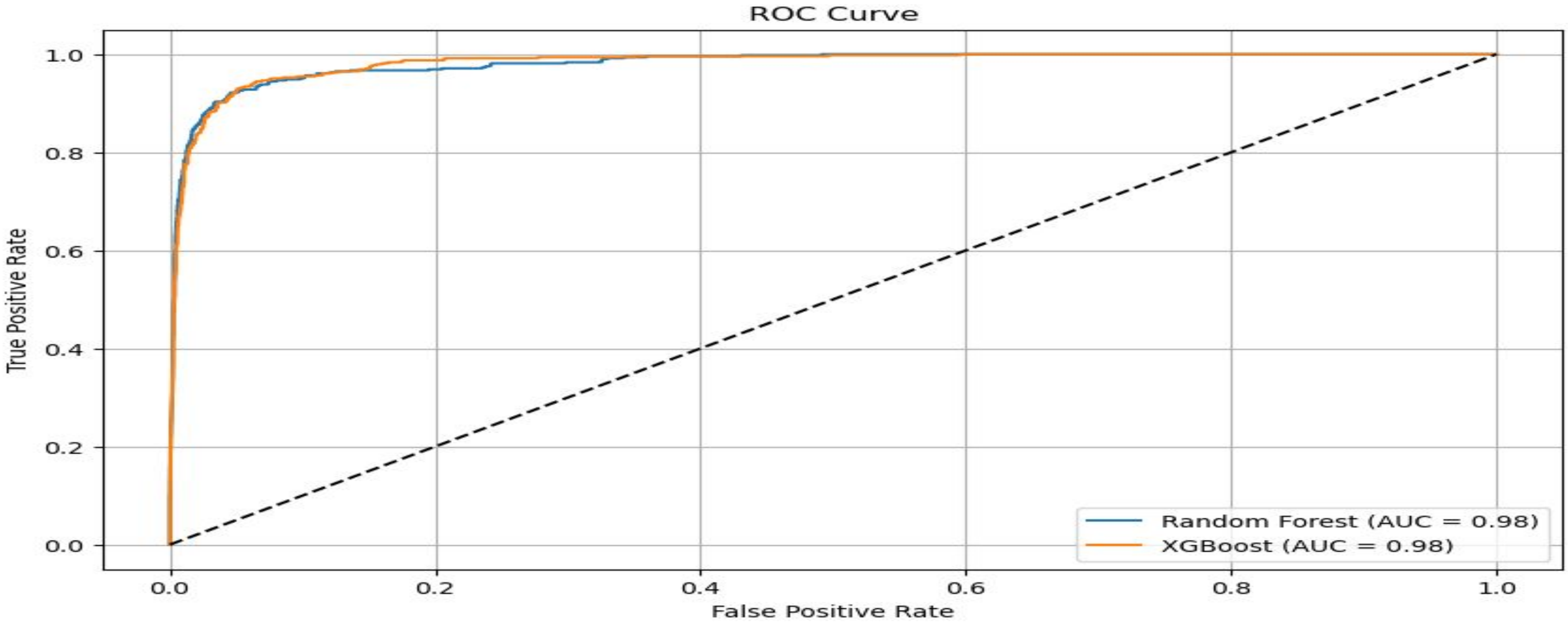
CONFUSION_MATRIX



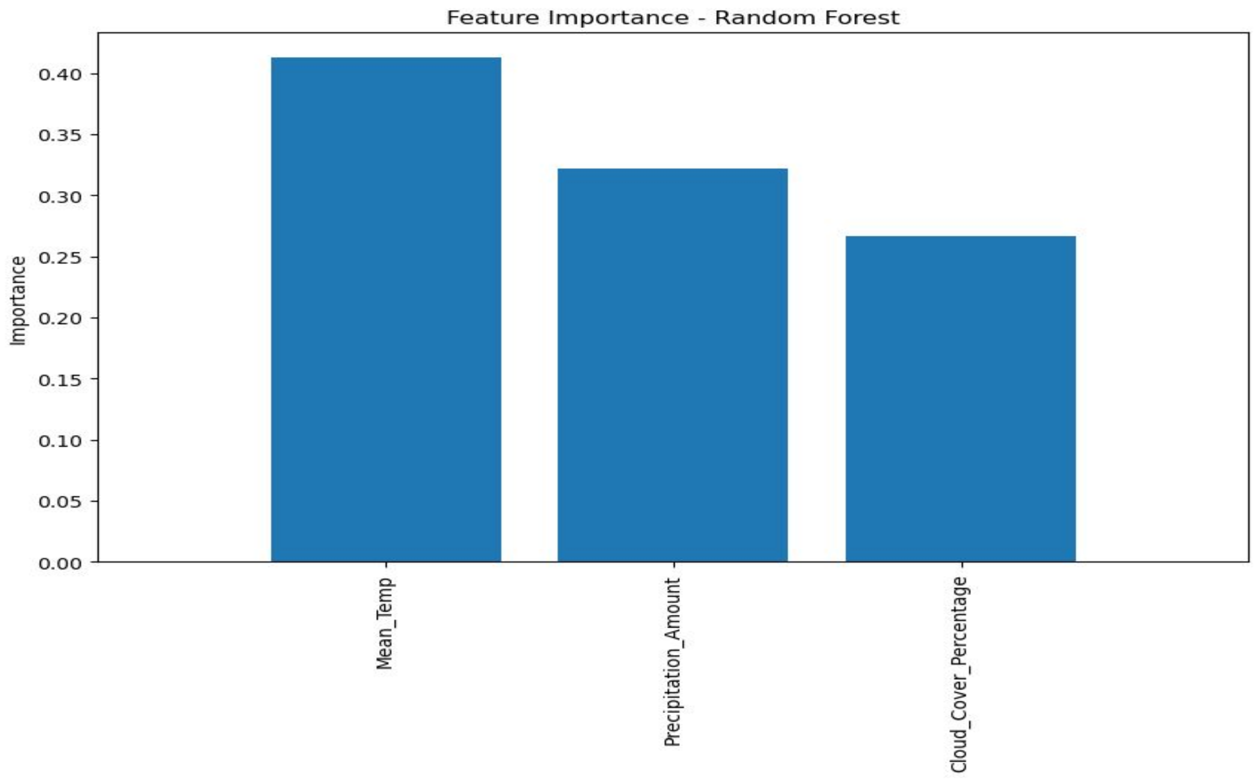
Compare Between Two Models

```
Comparison of Models:
Random Forest Test Accuracy: 0.9687952056209961
XGBoost Test Accuracy: 0.9669353172143005
Random Forest performed better on the test data.
```

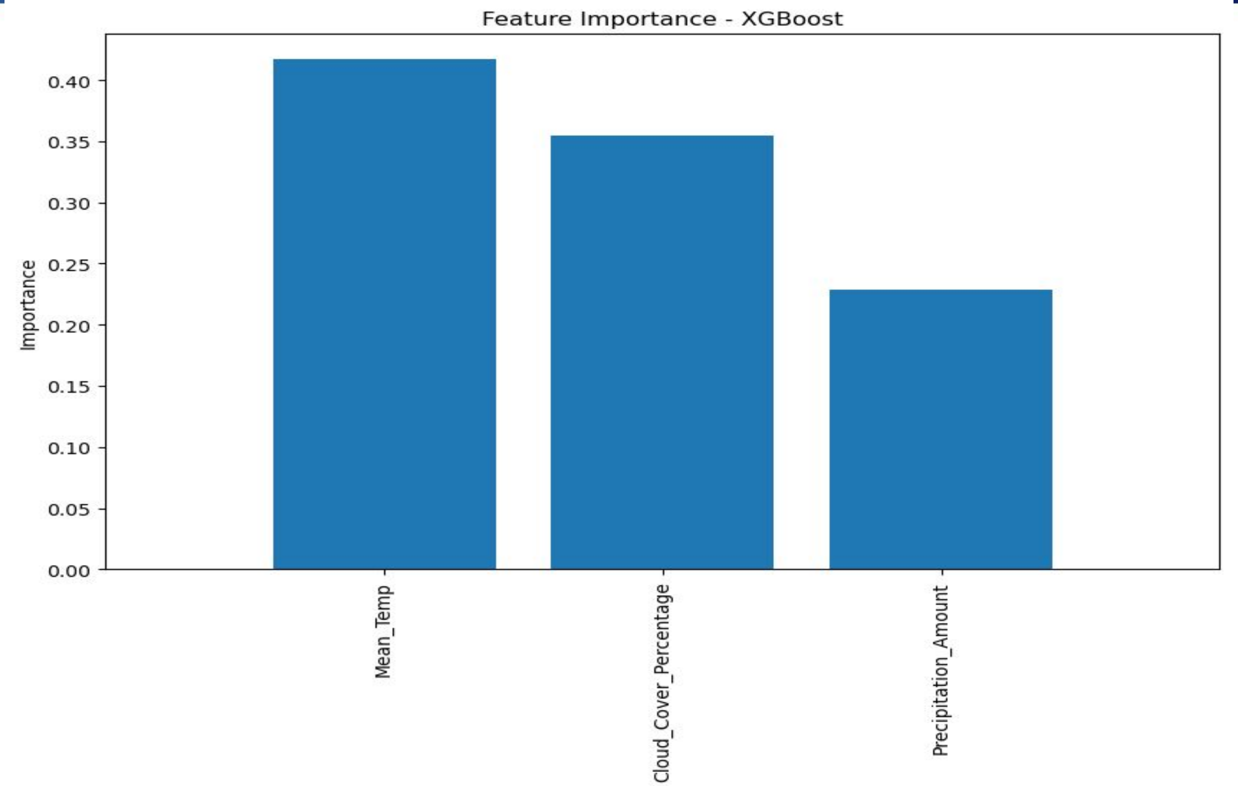
Visualization of ROC Curves



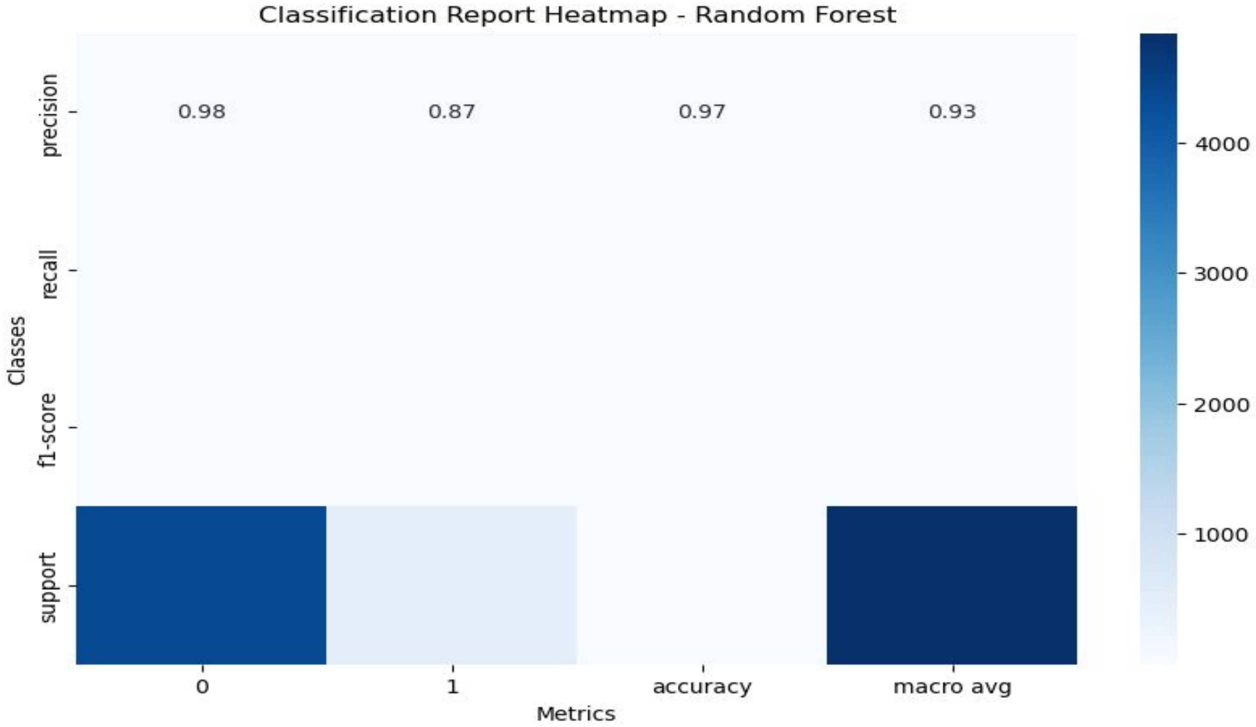
Feature Importance Plot_Random Forest



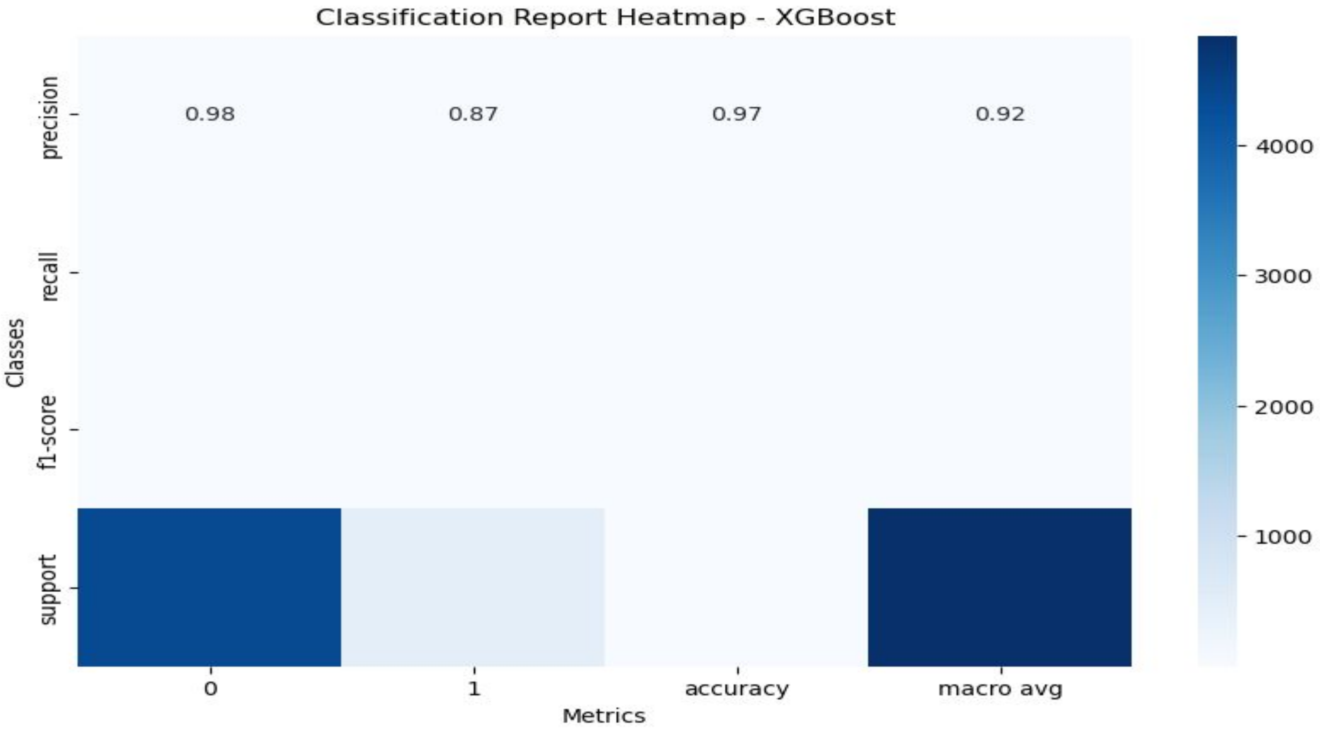
Feature Importance Plot_XGBOOST



Report_HeatMap_RandomForest



Report_HeatMap_XGBOOST



Save the model

```python
# Save the Random Forest model to a pickle file
import pickle
with open('best_random_forest_model.pkl', 'wb') as file:
    pickle.dump(best_rf_model, file)

print("Random Forest model saved successfully!")
```

```
Random Forest model saved successfully!
```

Finally

- We developed a predictive model for Lumpy Skin Disease using the forest random algorithm.
- The model demonstrates strong performance metrics, with 96% accuracy
- enabling proactive disease management in livestock.

Thank you