DEPARTMENT OF MATHEMATICS

COMSATS UNIVERSITY ISLAMABAD

FINAL YEAR PROJECT REPORT

PRESENTED BY

SADAQAT ALI
CIIT/SP21-BSM-013

PREDICTING LUMPY SKIN DISEASE USING MACHINE LEARNING

OVERVIEW

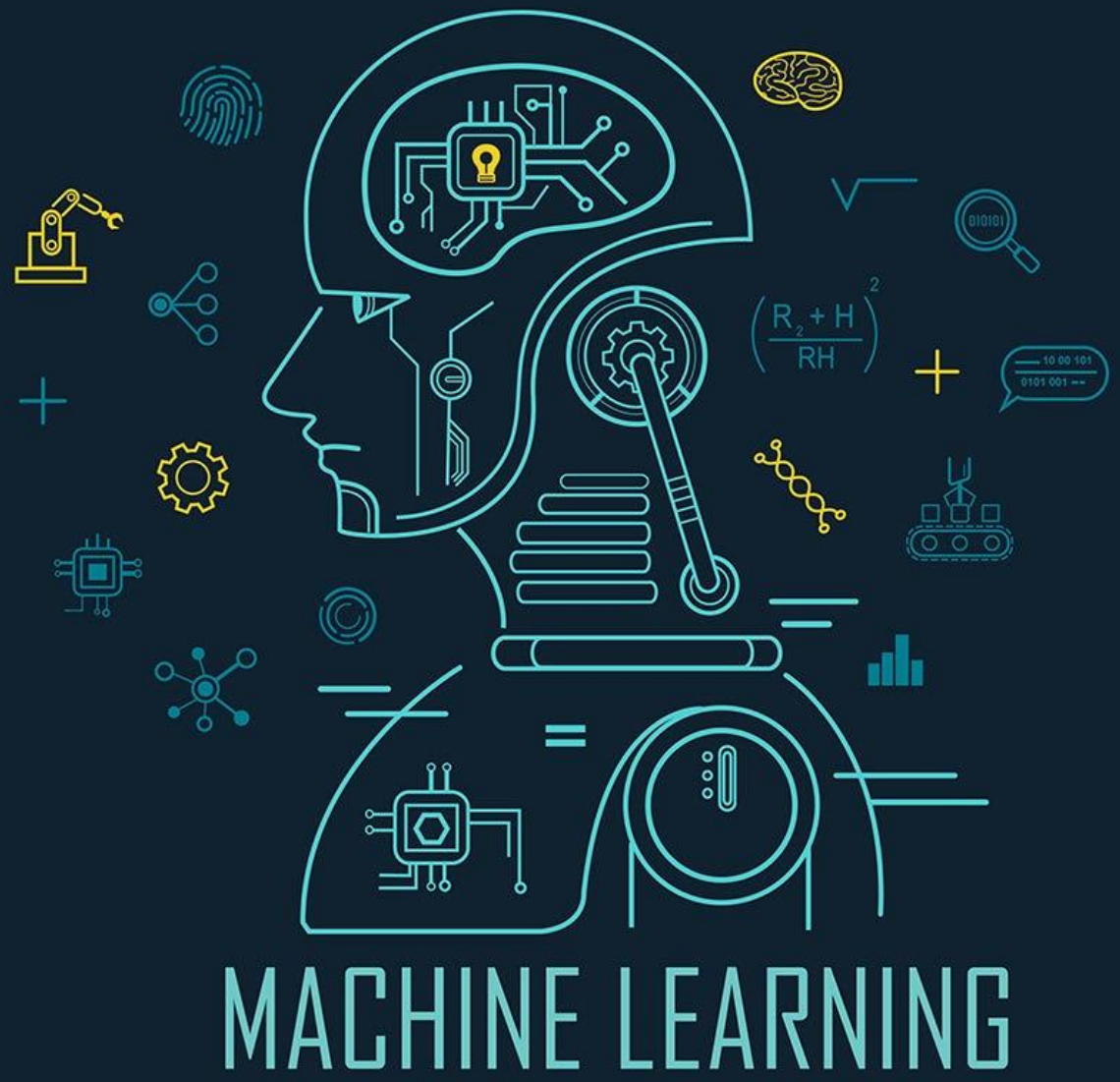
- Introduction to Lumpy Skin Disease (LSD).
- A Machine Learning approach to LSD Outbreak
- Introduction to Machine Learning (ML).
- Supervised Learning.
- Classification
- Programming Language and Libraries used.
- Methodology
- Conclusion

INTRODUCTION TO LUMPY SKIN DISEASE

Lumpy skin disease (LSD) is a viral disease affecting cattle, characterized by the development of skin nodules (lumps) over the body.

A MACHINE LEARNING APPROACH TO LSD OUTBREAK PREDICTION

Lumpy Skin Disease (LSD) inflicts significant economic losses and animal suffering. Its rapid spread complicates control efforts. Machine learning offers a proactive solution by predicting outbreaks before they occur. This enables timely interventions and minimizes the disease's impact.



INTRODUCTION TO MACHINE LEARNING (ML)

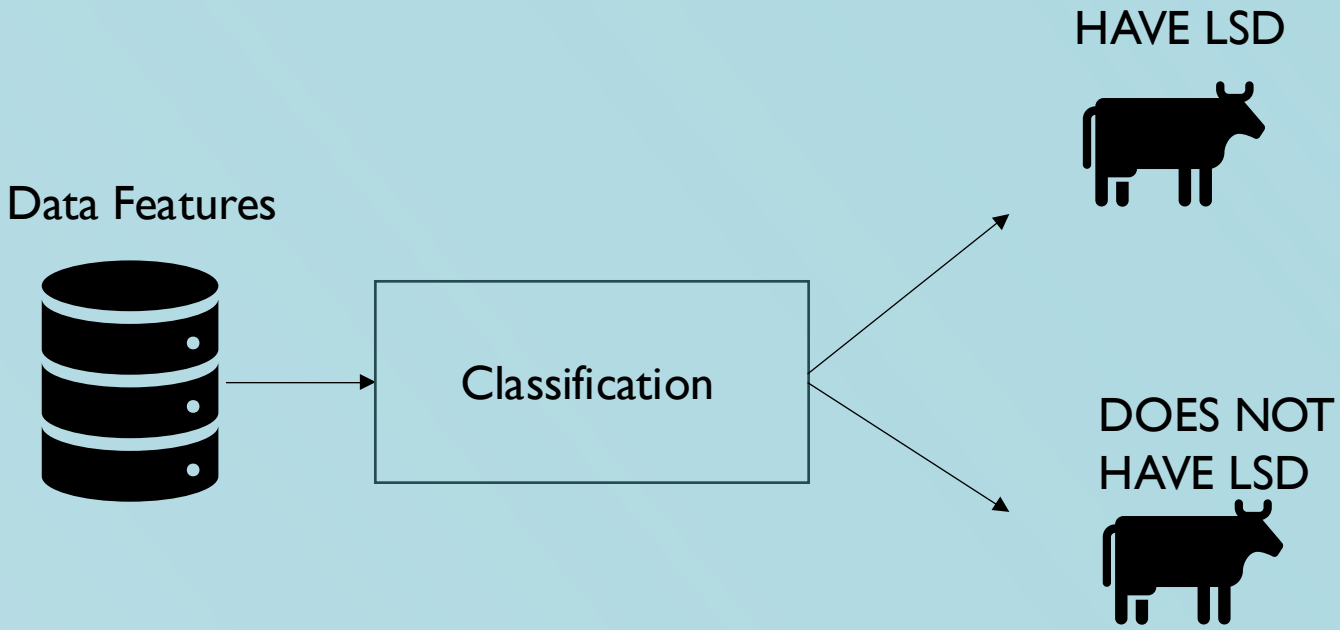
Machine learning is a field of artificial intelligence that empowers computers to learn from data without being explicitly programmed.

SUPERVISED LEARNING

Supervised learning trains an algorithm on labeled data (input-output pairs) to predict outputs for new inputs.

It has two main types

1. Classification (predicting categories)
2. Regression (predicting values).



CLASSIFICATION

Classification predicts the class or category of an input based on labeled training data. It learns to map input features to discrete categories, enabling the classification of new, unseen data points.

PROGRAMMING LANGUAGE AND LIBRARIES

- Python (General-purpose programming language used for model implementation)
- Pandas (Library for data manipulation, cleaning, and preparation)
- NumPy (Library for numerical computing and array operations)
- Plotly (Library for creating interactive visualizations of data and results)
- Scikit-learn (Library containing machine learning algorithms and tools)

METHODOLOGY

- Data Description
- Data Pre-processing
- Creating Baseline Models
- Oversampling
- Voting Classifier

Feature	Description	Variable Type	Example Values
Age	Age of the animal	Categorical	Calf, Heifer, Adult
Breed	Breed of the cattle	Categorical	Holstein, Jersey, Brahman, Local Breed
Sex	Sex of the animal	Categorical	Male, Female
Vaccination Status	Whether the animal has been vaccinated against LSD	Categorical	Vaccinated, Not Vaccinated
Previous LSD Infection	Whether the animal has had a previous LSD infection	Categorical	Yes, No
Current Health Status	Current health condition of the animal	Categorical	Healthy, Sick, Recovering
Geographic Location (Latitude)	Latitude of the farm or location	Numerical	34.0522° N, -118.2437° W
Geographic Location (Longitude)	Longitude of the farm or location	Numerical	34.0522° N, -118.2437° W
Altitude	Elevation of the farm or location above sea level	Numerical	1500 meters, 500 feet
Temperature (Average)	Average temperature in the area (e.g., annual or seasonal average)	Numerical	25°C, 77°F
Humidity (Average)	Average humidity in the area (e.g., annual or seasonal average)	Numerical	60%, 0.6
Rainfall (Average)	Average rainfall in the area (e.g., annual or seasonal average)	Numerical	1000 mm, 40 inches
Proximity to Water Bodies	Proximity to rivers, ponds, or other water sources	Numerical	1 km, 500 meters, Near, Far
Livestock Density	Number of livestock per unit area in the region	Numerical	100 cattle/km², 50 cattle/acre
Farm Size	Size of the farm	Numerical	10 hectares, 25 acres
Housing Type	Type of housing provided to the cattle	Categorical	Open Grazing, Confined Housing, Mixed
Quarantine Practices	Measures taken to isolate sick animals	Categorical	Yes (strict), Yes (partial), No
Insecticide Use	Use of insecticides on the farm	Numerical	1 (Yes) , 0 (No)
Lumpy	Does the cattle have Lumpy Skin Disease	Numerical	1 (Yes) , 0 (No)

DATA DESCRIPTION

Source (Kaggle)

Samples (24,804)

Features 19 (8 Categorical Variables, 11 Numerical Variables)

Target Variable (Lumpy)

DATA PRE-PROCESSING

Data Pre-processing transforms raw data into a usable format for machine learning. It involves cleaning

1. Converting Categorical to Numerical Values

2. Handling Missing Values

3. Removing Outliers

to improve model performance and accuracy.

CONVERTING CATEGORICAL TO NUMERICAL VARIABLES

Converting categorical variables to numerical values is essential as most machine learning algorithms require numerical input. This can be done using techniques like

- **Label Encoding**
- **One Hot Encoding**

which make categorical data suitable for model training. Label Encoding is applied when unique values in a feature is less than two else one hot encoding is applied.

Vaccination Status
Yes
No
No
Yes

Label Encoding

Vaccination Status
1
0
0
1

LABEL ENCODING

Label Encoding assigns each unique category in a categorical variable with an integer. No new columns are created.

Housing Type
Open Housing
Confined Housing
Open Housing
Mixed
Mixed
Mixed

One Hot Encoding

Open Housing	Confined Housing	Mixed
1	0	0
0	1	0
1	0	0
0	0	1
0	0	1
0	0	1

ONE HOT ENCODING

One-hot encoding creates a new column for each unique category in a categorical variable. Each observation recieves a ‘1’ in the column for its corresponding category and a ‘0’ in all other new columns.

```python
## Importing Libraries.

import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder

## Reading the File
train = pd.read_csv('/Users/sadaqat/Downloads/Lumpy skin disease data.csv')

# Create a label encoder object
le = LabelEncoder()
le_count = 0
ohe_count = 0  # Counter for One-Hot Encoded columns

# Iterate through the columns
for col in train:
    if train[col].dtype == 'object':  # Check if column is categorical
        if len(train[col].unique()) <= 2:  # For columns with 2 or fewer unique values
            le.fit(train[col])
            train[col] = le.transform(train[col])  # Apply Label Encoding
            le_count += 1
        else:  # For columns with more than 2 unique values
            # Apply One-Hot Encoding
            # drop_first=True avoids dummy variable trap
            train = pd.get_dummies(train, columns=[col], drop_first=True)
            ohe_count += 1

print(f'Number of columns label encoded: {le_count}')
print(f'Number of columns one-hot encoded: {ohe_count}')
```

LABEL & ONE-HOT ENCODING IMPLEMENTION IN PYTHON

- Import Libraries
- Read the File
- Create Encoder Object
- Iterate through Columns

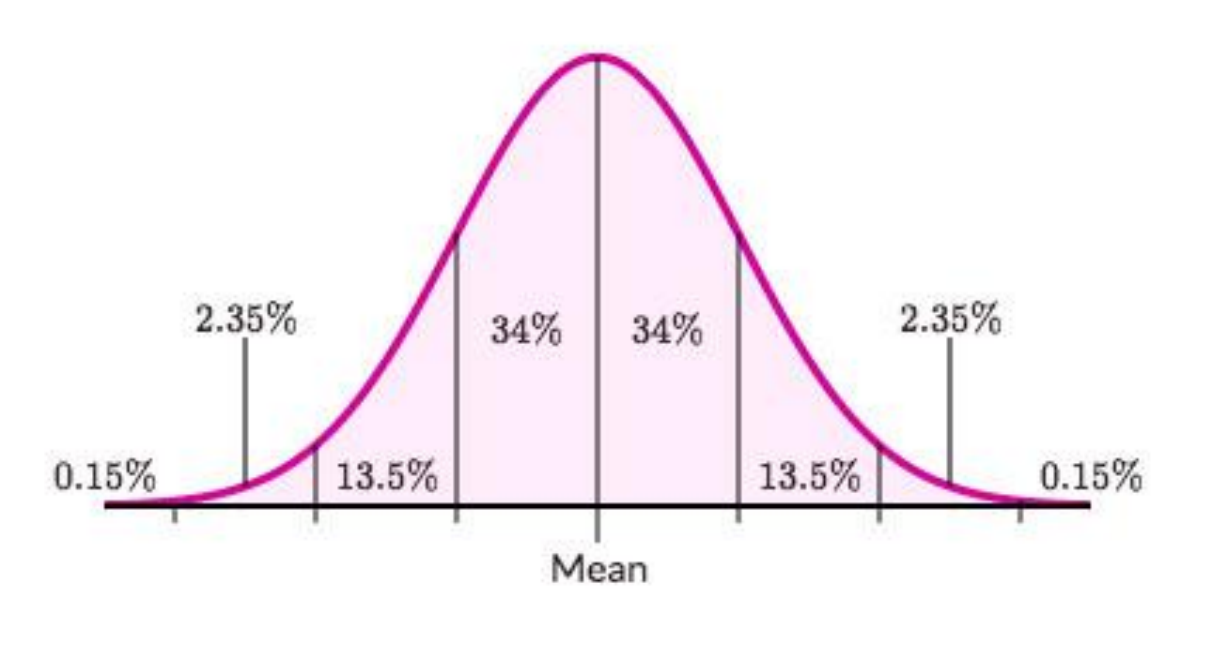
Age	Altitude	Temp	Breed	Sex	lumpy
Calf	1500	25	Holstein	Female	1
Adult	600	34	Jersy	Male	0
Heifer	N/A	N/A	Brahman	Male	1
N/A	1000	20	Local	Male	0
Calf	800	22	N/A	Female	0
N/A	700	32	Brahman	N/A	0
Adult	N/A	22	Jersy	Female	1
Heifer	800	19	Local	Male	0
Calf	1500	N/A	N/A	Male	1

HANDLING MISSING VALUES

Handling missing values is crucial in data pre-processing because most machine learning algorithms cannot handle missing data directly and may produce inaccurate or biased results.

HANDLING MISSING VALUES USING PYTHON

```python
def missing_values_table(train):
    # Total missing values
    mis_val = train.isnull().sum()

    # Percentage of missing values
    mis_val_percent = 100 * train.isnull().sum() / len(train)

    # Make a table with the results
    mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)

    # Rename the columns
    mis_val_table_ren_columns = mis_val_table.rename(
        columns={0: 'Missing Values', 1: '% of Total Values'})

    # Sort the table by percentage of missing values in descending order
    mis_val_table_ren_columns = mis_val_table_ren_columns[
        mis_val_table_ren_columns.iloc[:, 1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)

    # Print some summary information
    print(f"Your selected dataframe has {train.shape[1]} columns.\n"
          f"There are {mis_val_table_ren_columns.shape[0]} columns that have missing va

    # Return the dataframe with missing information
    return mis_val_table_ren_columns

missing_values = missing_values_table(train)
print(missing_values.head(20))

# Function to fill missing values
def impute_missing_values(train):
    for column in train.columns:
        if train[column].dtype == 'object':  # Categorical column
            train[column].fillna(train[column].mode()[0], inplace=True)
        else:  # Numerical column
            train[column].fillna(train[column].mean(), inplace=True)
    print("Missing values have been imputed.")
    return train

# Use the missing_values_table function to print missing statistics
# Impute missing values in the dataframe
train = impute_missing_values(train)
```



REMOVING OUTLIERS

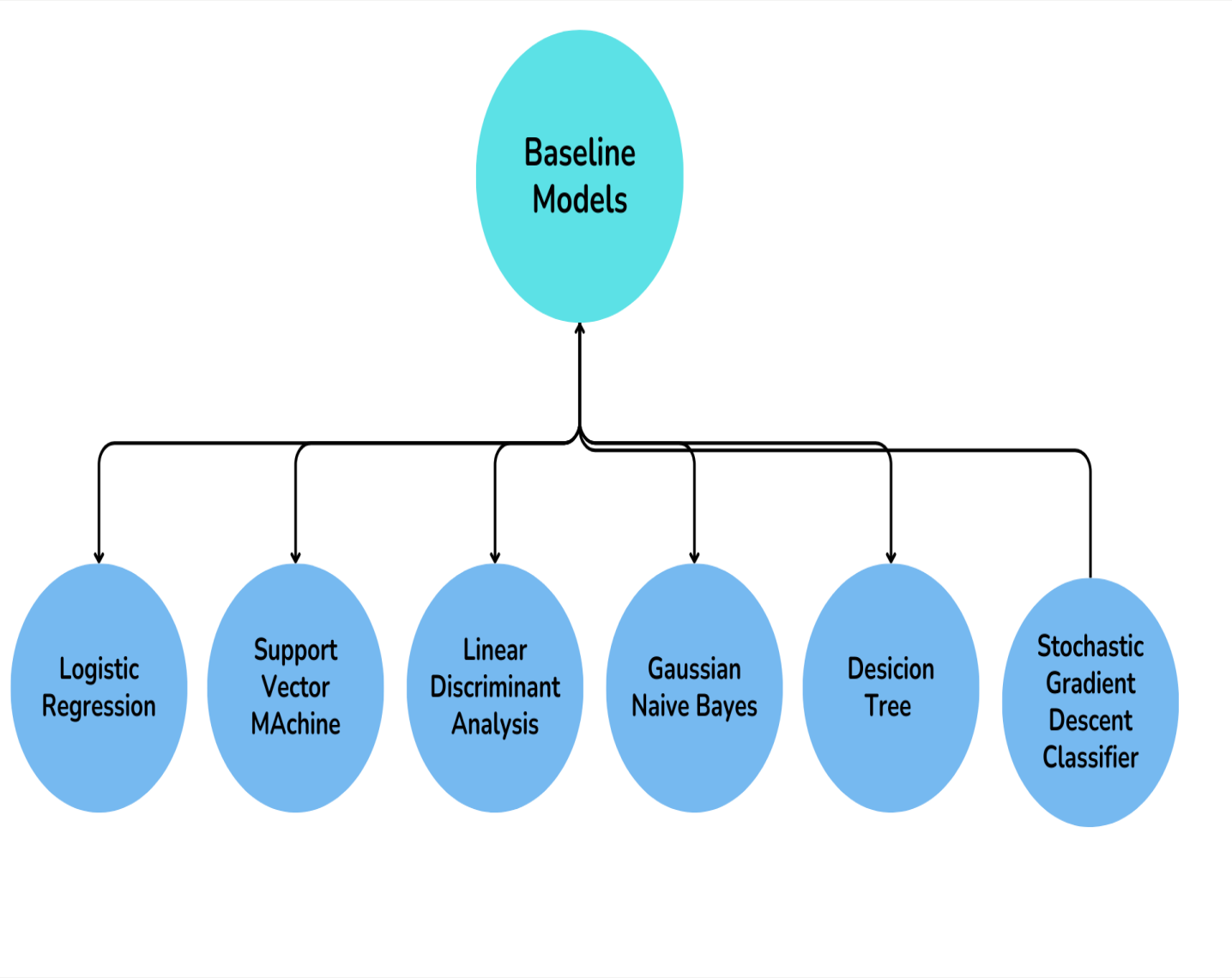
Outliers, which deviate significantly from other data points, were removed using the **Z-Score Method**. Data points with Z-scores beyond three standard deviations were identified as outliers and excluded. The formula used is

$$Z = \frac{X - \mu}{\sigma}$$

where X is the data point, μ is the mean, and σ is the standard deviation.

REMOVING OUTLIERS WITH PYTHON

```python
def remove_outliers_zscore(train, threshold=3):

    numerical_features = train.select_dtypes(include=[np.number]).columns
    for col in numerical_features:
        # Calculate the Z-score
        z_scores = (train[col] - train[col].mean()) / train[col].std()

        # Keep rows where the Z-score is within the threshold
        train = train[np.abs(z_scores) <= threshold]

    print(f"Outliers removed using Z-score method with threshold {threshold}.")
    return train

# Remove outliers from the 'train' dataframe
train = remove_outliers_zscore(train)
```



CREATING BASELINE MODELS

To establish a performance benchmark and evaluate the effectiveness of subsequent data pre-processing and modelling techniques, several baseline machine learning algorithms were initially trained and evaluated. These algorithms represent a range of model types commonly used for classification tasks.

BASELINE MODELS IN PYTHON

```python
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.metrics import precision_score, recall_score, accuracy_score

# Splitting the data into training (75%) and testing (25%) sets
X = train.drop(columns=['lumpy'])
y = train['lumpy']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Define a function to train the model and calculate metrics
def train_and_evaluate_model(model, X_train, X_test, y_train, y_test):
    # Train the model
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict(X_test)

    # Calculate scores
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)

    # Print scores
    print(f"Model: {model.__class__.__name__}")
    print(f"Accuracy: {accuracy:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print("-" * 30)

# Instantiate the classifiers
models = [
    DecisionTreeClassifier(),
    SVC(),
    GaussianNB(),
    LinearDiscriminantAnalysis(),
    LogisticRegression(),
    SGDClassifier()
]

# Train and evaluate each model
for model in models:
    train_and_evaluate_model(model, X_train, X_test, y_train, y_test)
```

Algorithms	Accuracy	Precision	Recall
Logistic Regression	0.9911	0.9723	0.9624
SVM	0.9176	0.7702	0.5928
GaussianNB	0.9927	0.9449	0.9717
Stochastic Gradient Descent	0.9236	0.9987	0.9994
Decision tree	0.9998	0.9987	0.9994
Linear Discriminant Analysis	0.9852	1	0.9366

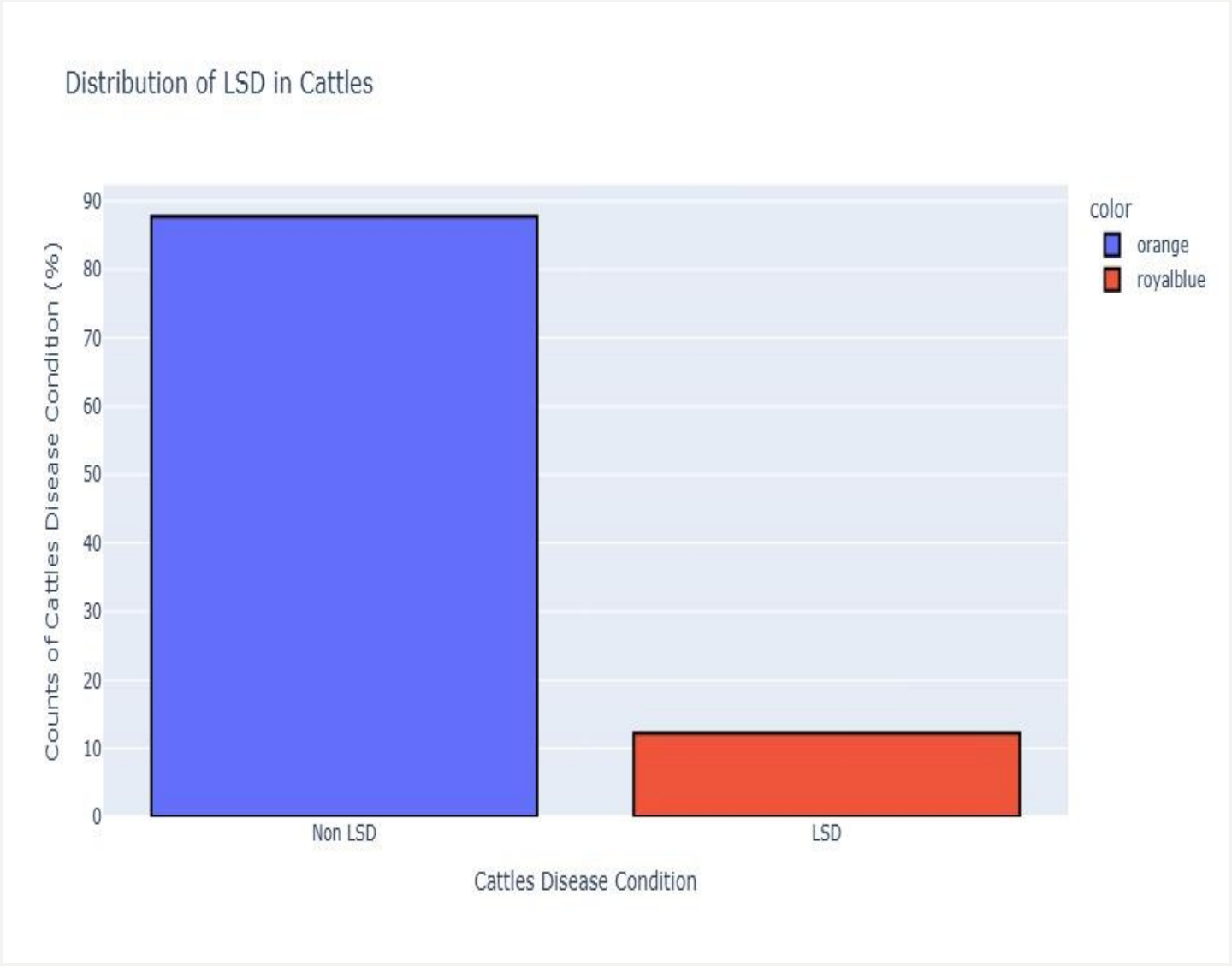
EVALUATION METRICS OF BASELINE MODELS

Three evaluation metrics were used in this project which are

Recall

Precison

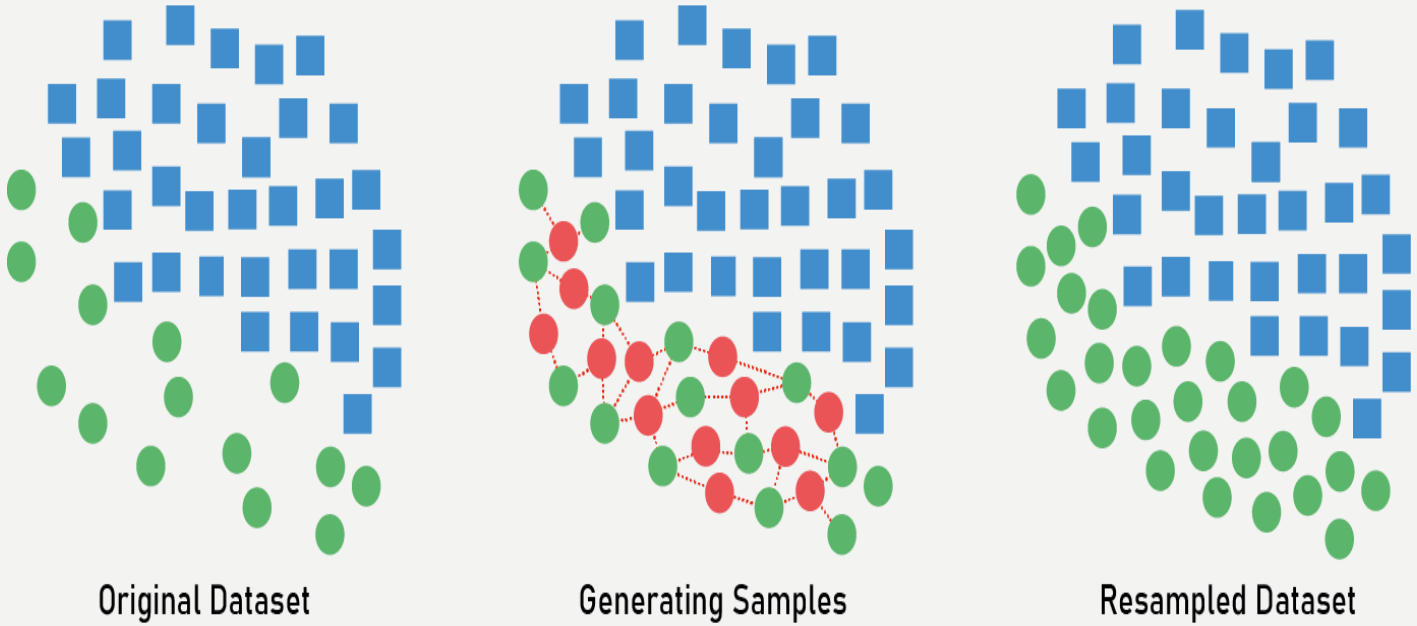
Accuracy.



DATA IMBALANCE

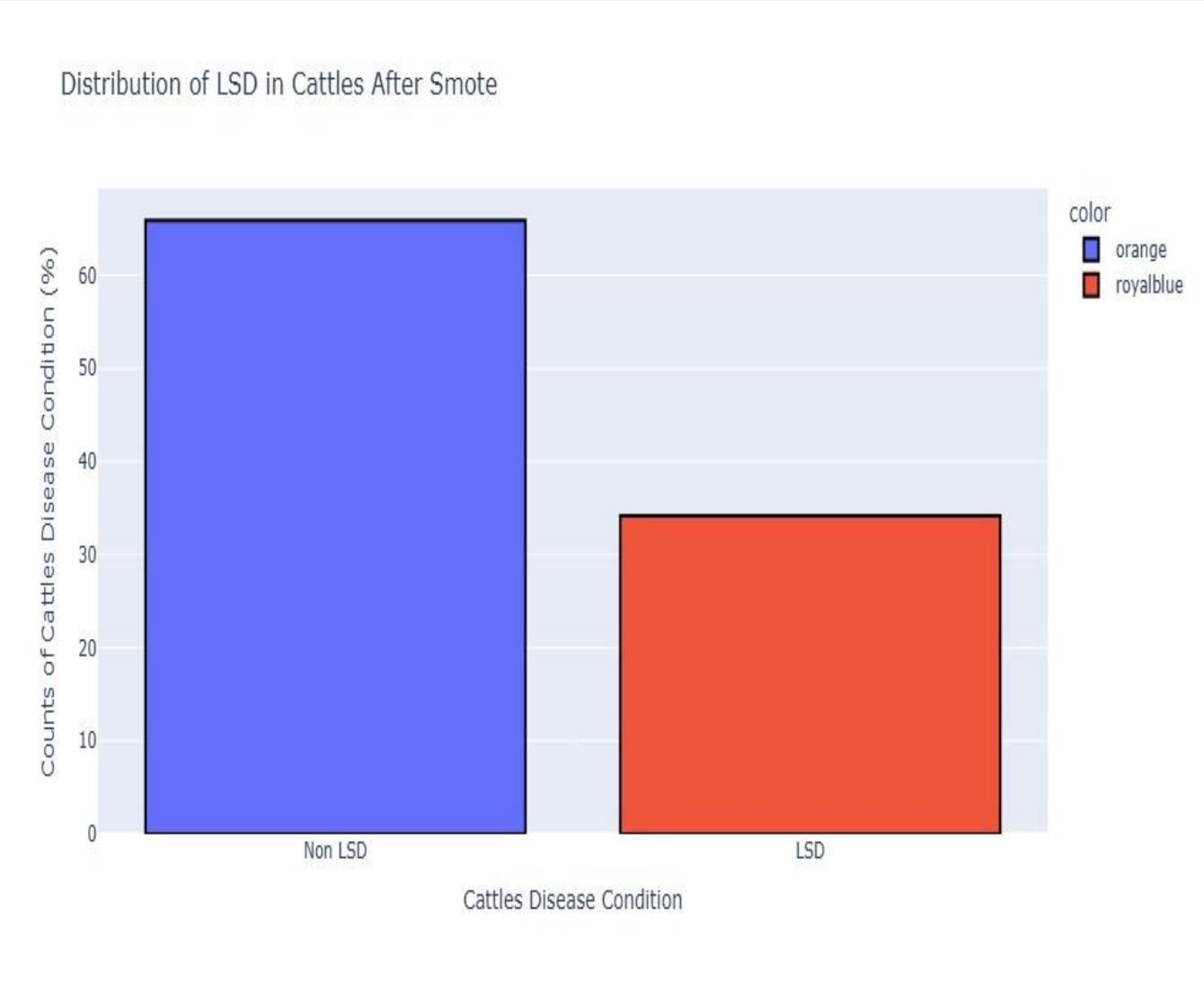
Class imbalance in machine learning is when one class is significantly under-represented. It poses a risk of bias in model training and leads to misleading metrics.

The figure shows the distribution of the target variable (Lumpy) which has 86% Non LSD and 14% LSD.



SYNTHETIC MINORITY OVER-SAMPLING TECHNIQUE (SMOTE)

SMOTE is an oversampling technique designed to address class imbalance by generating synthetic samples for the minority class. It creates new instances by interpolating between existing minority class samples and their nearest neighbors in the feature space.



DISTRIBUTION AFTER APPLYING SMOTE

The figure shows the distribuition of the Target Variable (LUMPY) after **SMOTE** has been applied. **SMOTE has generated synthetic samples which makes a new distribuition of the data with 65% Non LSD and 35% LSD.**

```python
# Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
print(f"After SMOTE, X_train shape: {X_train_smote.shape}, y_train shape: {y_train_smote.shape}")

# Define a function to train the model and calculate metrics
def train_and_evaluate_model(model, X_train, X_test, y_train, y_test):
    # Train the model
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict(X_test)

    # Calculate scores
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)

    # Print scores
    print(f"Model: {model.__class__.__name__}")
    print(f"Accuracy: {accuracy:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print("-" * 30)

# Instantiate the classifiers
models = [
    DecisionTreeClassifier(),
    SVC(),
    GaussianNB(),
    LinearDiscriminantAnalysis(),
    LogisticRegression(),
    SGDClassifier()
]

# Train and evaluate each model using SMOTE data
for model in models:
    train_and_evaluate_model(model, X_train_smote, X_test, y_train_smote, y_test)
```

RETRAINING THE ALGORITHMS ON NEW SAMPLES

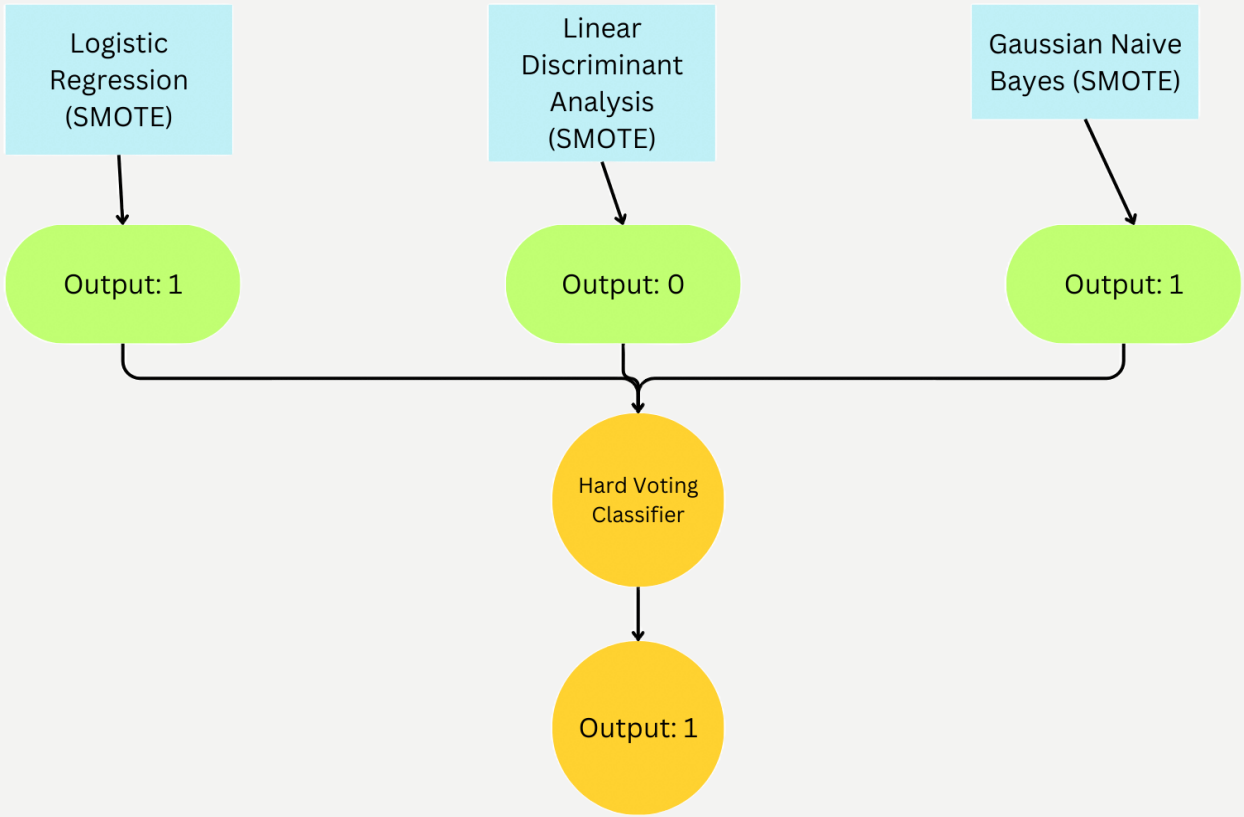
The algorithms were retrained on the samples generated by SMOTE to see if metrics have been improved.

Algorithms	Accuracy	Precision	Recall
Logistic Regression	0.9299	0.9863	0.9624
SVM	0.3589	0.7831	0.9443
GaussianNB	0.8549	0.9789	1
Stochastic Gradient Descent	0.5692	0.9002	0.8148
Decision tree	0.9987	0.99938	1
Linear Discriminant Analysis	0.9548	0.9892	0.9585

EVALUATION METRICS OF RE-TRAINED ALGORITHMS

The evaluation metrics after applying SMOTE and then retraining the algorithms are presented in the table.

Working of Voting Classifier



VOTING CLASSIFIER & MODEL SELECTION

Based on the evaluation of baseline models trained with resampling techniques, a **Hard Voting Classifier** has been implemented. HVC combines the predictions of multiple base learners by taking a majority vote. Three models which were selected for voting classifier are:

- LR with SMOTE
- LDA with SMOTE
- Gaussian Naïve Bayes with SMOte

EVALUATION METRICS OF VOTING CLASSIFIER

Alogirthm	Accuracy	Precision	Recall
Voting Classifier	0.9894	0.9526	0.9624

CONCLUSION

- In this project, machine learning has been effectively utilized to predict Lumpy Skin Disease (LSD), addressing a critical issue in livestock health management. By employing advanced data pre-processing techniques, handling class imbalance using SMOTE, and leveraging multiple algorithms, we achieved a high accuracy of 98.94% using a Voting Classifier.
- The results demonstrate the potential of machine learning in providing accurate and timely predictions for LSD outbreaks, allowing for proactive interventions to minimize economic losses and animal suffering. This study serves as a foundation for future research in applying AI-based solutions to other livestock diseases, ultimately contributing to improved animal health management systems.