Avian Prophet:

Machine Learning for
Bird Flu Detection

SmartInternz
www.smartinternz.com

Project Description:

This report aims to analyze and predict avian influenza (bird flu) trends using a machine learning approach. Avian influenza poses significant risks to both poultry and human health, making accurate prediction tools invaluable for timely intervention. The dataset utilized in this project is sourced from Kaggle and contains diverse features related to bird flu outbreaks

Scenario 1: Poultry Farm Management

Background

A large commercial poultry farm is in an area prone to avian influenza outbreaks. The farm manager needs to protect the health of thousands of birds.

Application

The farm manager uses the Flask web app to input data about the environment and bird populations.

Outcome

The model predicts a high risk of an H5 HPAI outbreak. The manager enhances biosecurity, increases health monitoring, and administers preventive treatments. This helps prevent an outbreak, safeguarding the farm's productivity.

Scenario 2: Wildlife Conservation

Background

A wildlife conservation organization monitors migratory birds in a wetland area, which can carry avian influenza and pose risks to nearby poultry farms.

Application

Conservationists input data on bird sightings and environmental conditions into the web app.

Outcome

The model indicates a moderate risk of an outbreak. Conservationists increase surveillance, launch an awareness campaign, and coordinate with local authorities. These actions mitigate the risk of the virus spreading to domestic poultry and humans.

Scenario 3: Government Health Agencies

Background

A national health agency manages animal health risks, including avian influenza, and needs to allocate resources effectively.

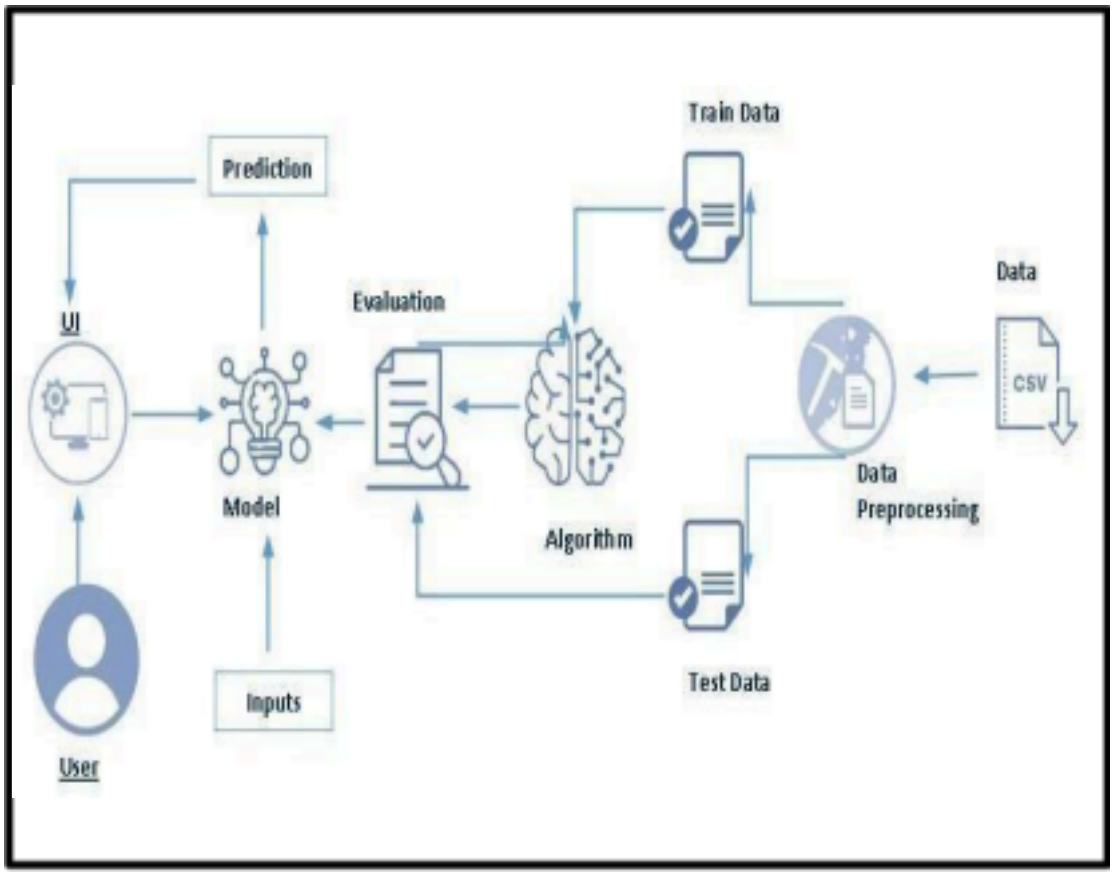
Application

The agency inputs regional data into the web app to predict H5 HPAI outbreaks.

Outcome

The model identifies high-risk areas. The agency deploys resources, issues advisories to farmers, and develops contingency plans. This proactive approach minimizes the impact of potential outbreaks on animal and human health.

Technical Architecture:



Prerequisites:

To complete this project, you must require the following software, concepts and packages

- **Anaconda Navigator :**

  o Refer to the link below to download Anaconda Navigator
  o Link: https://youtu.be/1ra4zH2G4o0

- **Python packages:**

  o Open anaconda prompt as administrator
  o Type “pip install numpy” and click enter.
  o Type “pip install pandas” and click enter.
  o Type “pip install scikit-learn” and click enter.
  o Type “pip install matplotlib” and click enter.
  o Type “pip install scipy” and click enter.
  o Type “pip install pickle-mixin” and click enter.
  o Type “pip install seaborn” and click enter.
  o Type “pip install Flask” and click enter.

Prior Knowledge:

You must have prior knowledge of the following topics to complete this project.

- **Concepts**

  o Supervised learning:
  https://www.javatpoint.com/supervised-machine-learning

  o Unsupervised learning:
  https://www.javatpoint.com/unsupervised-machine-learning

  o Decision Tree:
  https://scikit-learn.org/stable/modules/tree.html

  - Random forest Classifier:
  https://www.javatpoint.com/machine-learning-random-forest-algorithm

  o Evaluation metrics:
  https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluatio n-error-metrics/

- **Flask Basics**: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Objectives:

By the end of this project, you will:
● Know fundamental concepts and techniques used for machine learning.
● Gain a broad understanding of data.
● Know pre-processing the data/transformation techniques and some visualization concepts.

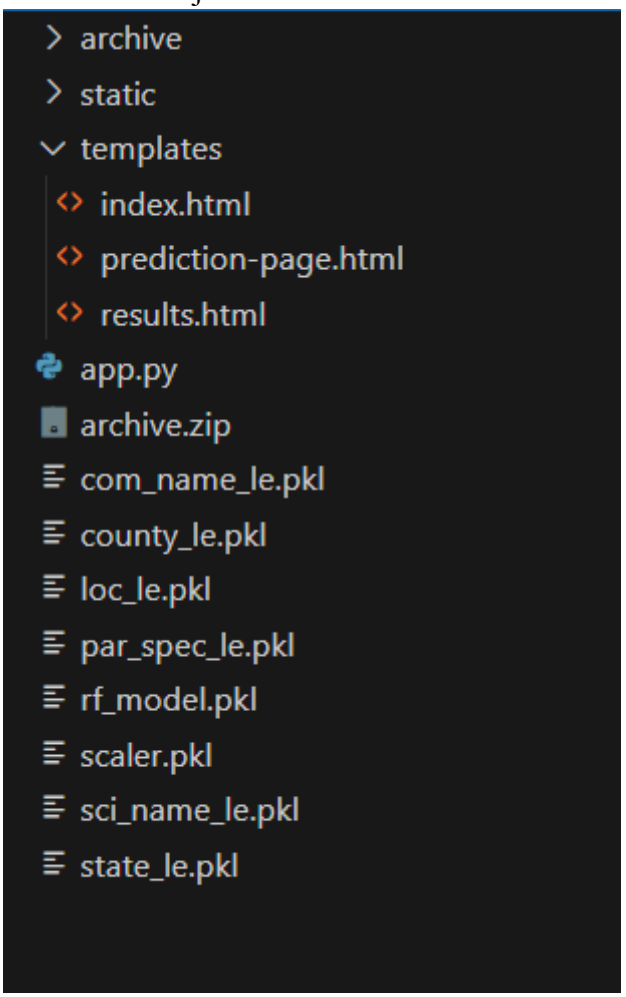
Project Flow:

● The user interacts with the UI to enter the inputs.
● Entered inputs are analyzed by the model which is integrated.
● Once the model analyzes the inputs the prediction is showcased the UI To accomplish this, we have to complete all the activities listed below,

- Data collection
  - o Collect the dataset
- Visualizing and analyzing data
  - o Univariate analysis
  - o Bivariate analysis
  - o Multivariate analysis
  - o Descriptive analysis
- Data pre-processing
  - o Checking for null values
  - o Feature Engineering
  - o Data Transformation
  - o Splitting the Data
- Model building
  - o Import the model-building libraries
  - o Initializing the model
  - o Training and testing the model
  - o Evaluating the performance of the model
  - o Save the model
- Application Building
  - o Create an HTML file
  - o Build Python code

Project Structure:

Create the Project folder with the Project name



We have to store our files and folders

Milestone 1: Data Collection

ML depends heavily on data, It is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

Activity 1: Download the dataset

There are many popular open sources for collecting the data.
Eg: kaggle.com, UCI repository, etc.

In this project, we have used "Bird Flu (Avian Influenza)" data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: https://www.kaggle.com/datasets/jasmeet0516/bird-flu-dataset-avian-influenza

Activity 2: Importing the libraries

Import the necessary libraries as shown in the image

```
# importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
import pickle
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import StratifiedKFold
```

Activity 3: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, etc. We can read the dataset with the help of pandas.
In pandas, we have a function called read_csv() to read the dataset. As a parameter, we have to give the name of the CSV file. We have one dataset“Avian Influenza (HPAI).csv” data and let's check the shape of our datasets.

```
df = pd.read_csv("/content/Avian Influenza (HPAI).csv", index_col = '_id')
```

```
df.shape
```

```
(16304, 16)
```

Milestone 2: Visualizing and analysing the data

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are several techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

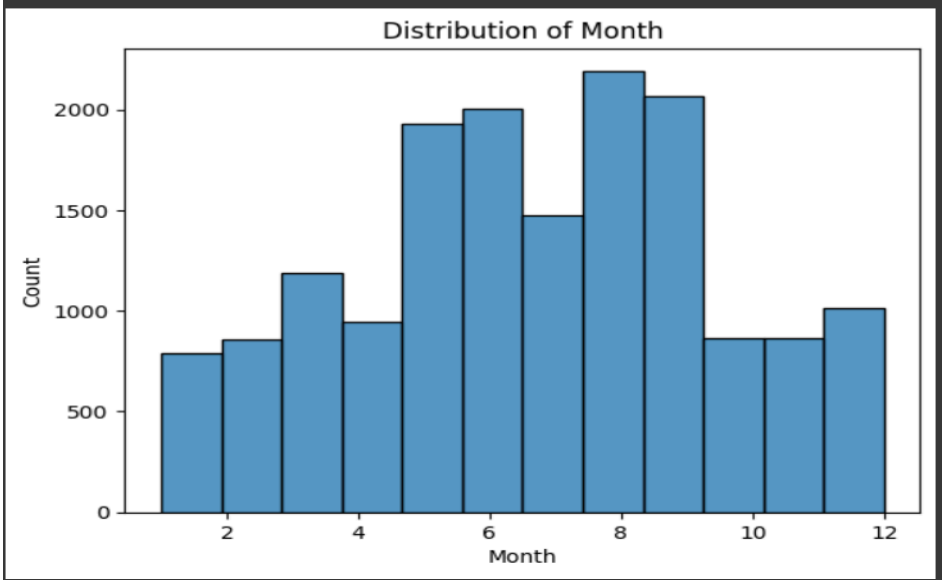
Activity 1: Univariate analysis

Univariate analysis involves the examination of a single variable to understand its distribution, central tendency, and variability. This type of analysis is fundamental in exploratory data analysis (EDA) because it helps to summarise and find patterns in individual variables, which can guide further analysis and modelling decisions.

Histogram (Histplot)

- **Purpose**: To visualise the distribution of a single continuous variable.
- **Interpretation**: The x-axis represents the variable values, while the y-axis represents the frequency (or count) of observations within each bin.
- **Insights**: Histograms can reveal the central tendency, variability, and shape of the data distribution (e.g., normal, skewed, bimodal).

```
1 # Distribution of Month
2 sns.histplot(df['Month'], bins=12)
3 plt.title('Distribution of Month')
4 plt.show()
5
```

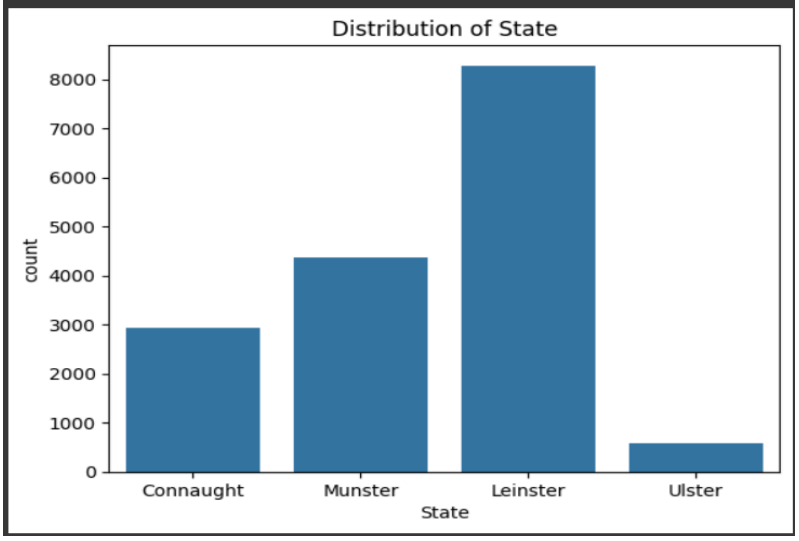


Count Plot

A count plot in Seaborn is a type of bar plot that is used to display the count of observations in each categorical bin using bars. It is particularly useful for visualizing the distribution of a single categorical variable. The length of each bar corresponds to the count of occurrences for each category, making it easy to compare the frequency of different categories at a glance.

In Seaborn, the countplot function simplifies the creation of these plots. It accepts parameters such as the data source and the specific column to be plotted. Additionally, it allows customization of colours, orientations, and other aesthetic properties to enhance the visual appeal and readability of the plot. For instance, setting the hue parameter can introduce another level of categorization, allowing for the comparison of counts across multiple categorical variables within the same plot.

```
1 # Distribution of State
2 sns.countplot(x='State', data=df)
3 plt.title('Distribution of State')
4 plt.show()
```



Activity 2: Bivariate analysis

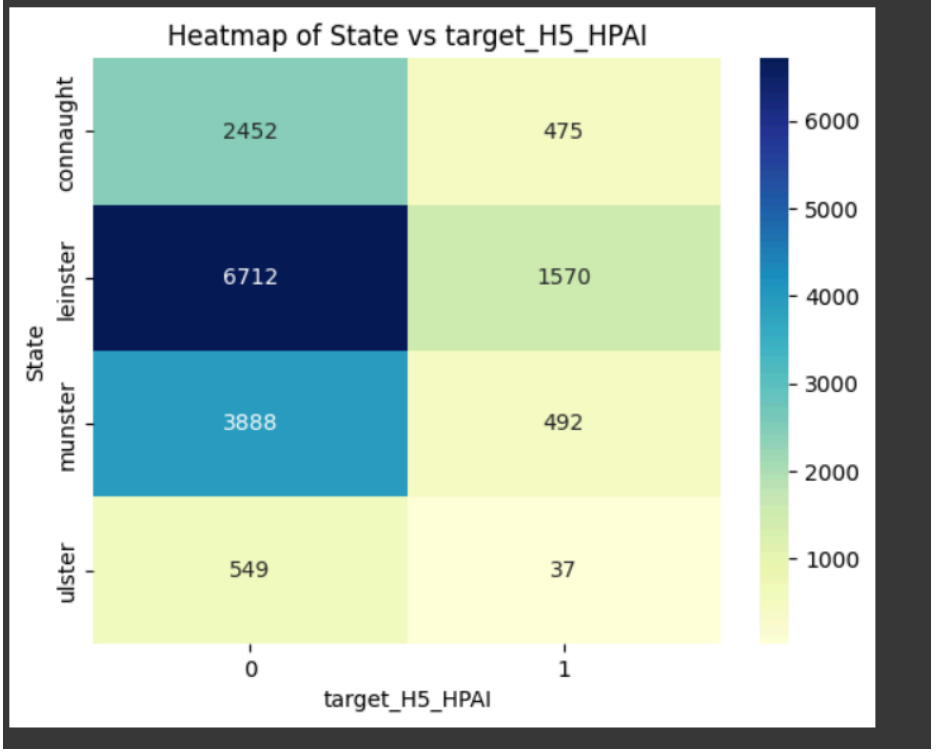
Bivariate analysis involves the exploration of the relationship between two variables. It is a fundamental step in data analysis that helps to understand the correlation, causation, and potential interactions between variables. This type of analysis can reveal how one variable affects or is associated with another, providing deeper insights that are essential for model development and feature selection.

Heat Map

A heatmap visualizes the frequency or relationship between two categorical variables using colors. When applied to a crosstab (cross-tabulation), which presents the joint distribution of two or more variables, heatmaps provide an intuitive way to grasp patterns and trends within the data.

The heatmap representation of a crosstab typically involves shading cells with colors corresponding to the frequency or proportion of observations falling into each cell. Darker colors often indicate higher frequencies or proportions, while lighter colors represent lower values. This color gradient allows for quick identification of areas with higher or lower concentrations of data.

```
# Crosstab and heatmap for State and target_H5_HPAI
crosstab = pd.crosstab(df['State'], df['target_H5_HPAI'])
sns.heatmap(crosstab, annot=True, fmt='d', cmap='YlGnBu')
plt.title('Heatmap of State vs target_H5_HPAI')
plt.xlabel('target_H5_HPAI')
plt.ylabel('State')
plt.show()
```



Activity 3: Multivariate analysis

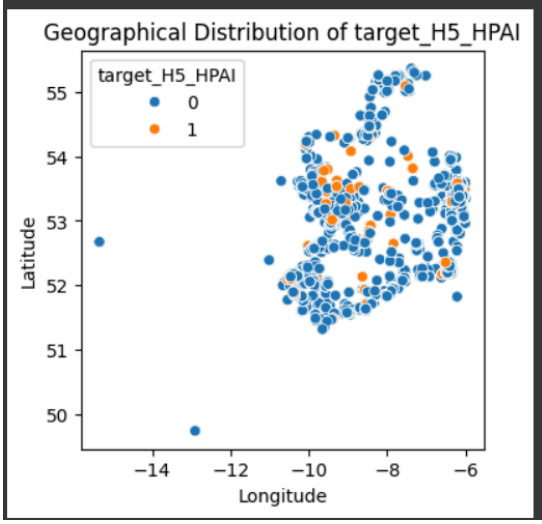
Multivariate analysis involves examining more than two variables simultaneously to understand the relationships and interactions among them. This type of analysis is essential in complex datasets where interactions between variables can provide deeper insights that are not apparent in univariate or bivariate analyses.

Scatter Plot

The scatter plot with a hue parameter allows for a concise yet informative visualization of multivariate relationships. By plotting two numerical variables against each other while incorporating a categorical variable as hue, this method offers a comprehensive view of how the relationship between the numerical variables varies across different categories. Through color differentiation, the plot highlights distinct patterns or trends within each category, facilitating a deeper understanding of the interplay between variables and their association with categorical

distinctions. This approach enhances data exploration and analysis by providing insights into multivariate relationships in a visually intuitive manner.

```
# Scatter plot for Latitude, Longitude colored by target_H5_HPAI
plt.figure(figsize=(4, 4))
sns.scatterplot(x='Longitude', y='Latitude', hue='target_H5_HPAI', data=df)
plt.title('Geographical Distribution of target_H5_HPAI')
plt.show()
```

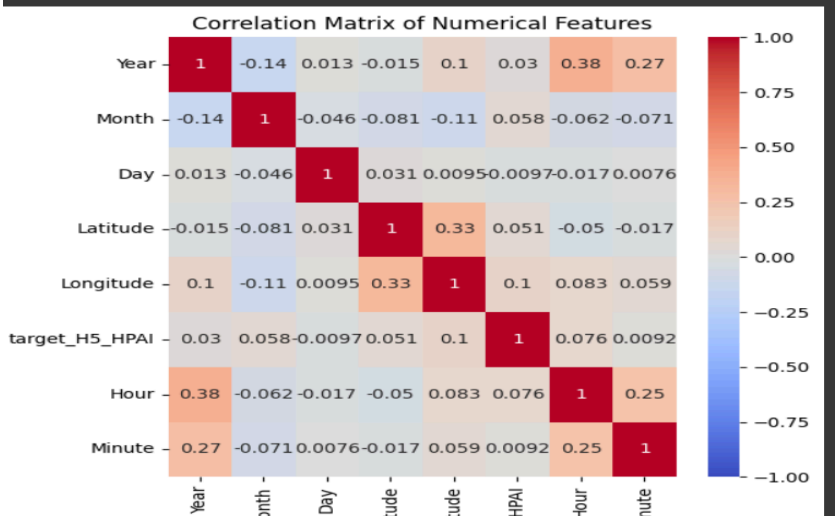


Heatmap

- **Purpose**: To visualize the strength and direction of relationships between multiple variables.

- **Interpretation**: Each cell in the heatmap represents the correlation coefficient between two variables. The color indicates the magnitude and direction of the correlation (e.g., positive or negative).
- **Insights**: Heatmaps make it easy to identify strong correlations, weak correlations, and potential multicollinearity issues in the dataset.

```
# Correlation matrix for numerical features
plt.figure(figsize=(6, 6))
correlation_matrix = df[numerical_columns].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```



Activity 4: Descriptive analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas have a worthy function called describe. With this described function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
1 df.describe()
```

	Year	Month	Day	Latitude	Longitude	target_H5_HPAI	Hour	Minute
count	16175.000000	16175.000000	16175.000000	16175.000000	16175.000000	16175.000000	16175.000000	16175.000000
mean	2017.016754	6.690015	15.943617	53.026753	-7.693278	0.159134	10.213168	17.505842
std	2.450732	2.971545	8.439803	0.752326	1.585539	0.365813	4.746544	18.468534
min	1980.000000	1.000000	1.000000	49.733900	-15.345800	0.000000	0.000000	0.000000
25%	2016.000000	5.000000	9.000000	52.518100	-9.079650	0.000000	8.000000	0.000000
50%	2018.000000	7.000000	16.000000	53.267300	-6.924400	0.000000	11.000000	12.000000
75%	2019.000000	9.000000	23.000000	53.388900	-6.197400	0.000000	14.000000	30.000000
max	2020.000000	12.000000	31.000000	55.370600	-5.996000	1.000000	21.000000	59.000000

```
1 df.describe(include=['O'])
```

	Scientific_Name	Common_Name	Date	Time	Country	Country_State_County	State	County	Locality	Parent_Species
count	16175	16175	16175	16175	16175	16175	16175	16175	16175	16175
unique	410	410	1492	621	1	25	4	26	1287	338
top	Erithacus rubecula	European Robin	2019-05-25T00:00:00	0:00	Ireland	IE-L-DN	Leinster	Dublin	Rogerstown Estuary--Turvey Hide	Erithacus rubecula
freq	414	414	107	1804	16175	5191	8282	5191	464	414

Milestone 3: Data Pre-processing

As we have understood how the data is, let's pre-process the collected data. The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Scaling Techniques
- Handling class imbalance
- Splitting dataset into training and test set

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 1: Checking for null values and dropping unwanted cols

- Let’s find the shape of our dataset first, To find the shape of our data. To find the data type, the df.info() function is used and in our dataset,

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 16175 entries, 1 to 16304
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Scientific_Name       16175 non-null  object
 1   Common_Name           16175 non-null  object
 2   Date                  16175 non-null  object
 3   Year                  16175 non-null  int64
 4   Month                 16175 non-null  int64
 5   Day                   16175 non-null  int64
 6   Time                  16175 non-null  object
 7   Country               16175 non-null  object
 8   Country_State_County  16175 non-null  object
 9   State                 16175 non-null  object
 10  County                16175 non-null  object
 11  Locality              16175 non-null  object
 12  Latitude              16175 non-null  float64
 13  Longitude             16175 non-null  float64
 14  Parent_Species        16175 non-null  object
 15  target_H5_HPAI        16175 non-null  int64
 16  Hour                  16175 non-null  int64
 17  Minute                16175 non-null  int64
dtypes: float64(2), int64(6), object(10)
memory usage: 2.3+ MB
```

- For checking the null values, df.isna() function is used. To sum those null values we use .sum() function. From the below image, we found no null values are present in our dataset.

```
1 df.isna().sum()
```

```
Scientific_Name           0
Common_Name               0
Date                      0
Year                      0
Month                     0
Day                       0
Time                      0
Country                   0
Country_State_County      0
State                     0
County                    0
Locality                  0
Latitude                  0
Longitude                 0
Parent_Species            0
target_H5_HPAI            0
Hour                      0
Minute                    0
dtype: int64
```

Activity 2: Data Cleaning and feature engineering

- We have string data, so are making it lower string for better analysis
- Convert date and time columns to DateTime format.
- Extract year, month, day, hour, and minute from datetime columns.
- Drop unnecessary columns.

```
# Correcting the values with non alpha values
df['Locality'] = df['Locality'].replace('Inishmore (Inis M�r)','Inishmore')
```

```
# Seprating the time into hour and minute columns with a colon(:)
def change(x) :
  x = str(x)
  minutes = x[-2:]
  hours = x[:-2]
  if len(minutes) <= 1 :
    minutes = '0' + minutes
  if len(hours) <= 1 :
    hours = '0' + hours
  return hours + ':' + minutes
```

```
df['Time'] = df['Time'].apply(change)
```

```
# Extracting the minute and hour from the time column
df['Hour'] = df['Time'].apply(lambda x : x.split(':')[0])
df['Minute'] = df['Time'].apply(lambda x : x.split(':')[1])
```

```
# converting the new columns into int64
df['Hour'] = df['Hour'].astype('int64')
df['Minute'] = df['Minute'].astype('int64')
```

```
df['Scientific_Name'] = df['Scientific_Name'].str.lower()
df['Common_Name'] = df['Common_Name'].str.lower()
df['County'] = df['County'].str.lower()
df['Locality'] = df['Locality'].str.lower()
df['Parent_Species'] = df['Parent_Species'].str.lower()
df['State'] = df['State'].str.lower()
```

Outliers

Traditional outlier detection methods might not be as meaningful for datetime columns like Year, Month, Day, Hour, and Minute. Instead, we should think about outliers in terms of contextually unusual values or data entry errors. Here are some specific types of potential outliers and how you might identify them:

Potential Outliers in Datetime Columns

1. **Unusual Dates or Times:**

- Invalid Dates: Dates that don't exist, such as February 30th or November 31st.
- Future Dates: Dates that are unreasonably in the future given the context of the data.
- Past Dates: Dates that are unreasonably far in the past.
- Out-of-Range Times: Times that don't fall within the expected operational hours (e.g., data for business operations outside of typical working hours).

2. **Inconsistent Temporal Data:**

- Leap Years: Dates that should not exist in non-leap years, like February 29th.
- Timezone Issues: datetimes that are inconsistent due to timezone conversions.
- Daylight Saving Time (DST) Changes: Times that may need adjustment due to DST transitions.

Identifying Outliers

Basic Validity Checks:

1. Ensure that all datetime values are valid and make sense in the context of your dataset.

```
# Convert columns to datetime, handling errors
data['datetime'] = pd.to_datetime(data[['Year', 'Month', 'Day', 'Hour', 'Minute']], errors='coerce')
invalid_dates = data[data['datetime'].isna()]
print(invalid_dates)

```

```
Empty DataFrame
Columns: [Scientific_Name, Common_Name, Date, Year, Month, Day, Time, Country, Country_State_County, State,
Index: []
```

Activity 3: Data Transformation

We have a lot of categorical features in our dataset and we are going to use label encoder to transform the categorical features into numericals.

```
sci_name_le = LabelEncoder().fit(df['Scientific_Name'])
df['Scientific_Name'] = sci_name_le.transform(df['Scientific_Name'])
```

```
com_name_le = LabelEncoder().fit(df['Common_Name'])
df['Common_Name'] = com_name_le.transform(df['Common_Name'])
```

```
state_le = LabelEncoder().fit(df['State'])
df['State'] = state_le.transform(df['State'])
```

```
county_le = LabelEncoder().fit(df['County'])
df['County'] = county_le.transform(df['County'])
```

```
loc_le = LabelEncoder().fit(df['Locality'])
df['Locality'] = loc_le.transform(df['Locality'])
```

```
par_spec_le = LabelEncoder().fit(df['Parent_Species'])
df['Parent_Species'] = par_spec_le.transform(df['Parent_Species'])
```

Activity 4: Scaling Techniques

First, split the dataset into x and y and then split the data set
Here x and y variables are created. On the x variable, df is passed by dropping the target variable. And on y target variable is passed.
For the standard scaler we have to take only x columns, for x columns we do standard as shown in below.
Standardscaler is particularly used for scaling numerical features to have a mean of 0 and a standard deviation of 1. This process is important because it helps bring features to a similar scale, which can improve the performance of certain algorithms and models.

```
1 X = df.drop(['target_H5_HPAI'], axis = 1)
2 y = df['target_H5_HPAI']
```

+ Code + Text

```
1 scaler = StandardScaler().fit(X)
2 X_scaled = pd.DataFrame(scaler.transform(X), columns = X.columns)
```

```
1 X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size = 0.2, random_state = 42)
```

```
1 y_train.value_counts()
```

```
target_H5_HPAI
0    10873
1     2067
Name: count, dtype: int64
```

For splitting training and testing data we are using the train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.
Our Dataset is highly imbalance dataset, so we are going to smote a technique to balance the label classes

Imbalanced Learning:

Imbalanced classification involves developing predictive models on classification datasets that have a severe class imbalance.
The challenge of working with imbalanced datasets is that most machine learning techniques will ignore, and in turn have poor performance on, the minority class, although typically it is the performance of the minority class that is most important.
One approach to addressing imbalanced datasets is to oversample the minority class. The simplest approach involves duplicating examples in the minority class, although these examples don't add any new information to the model. Instead, new examples can be synthesized from the existing examples. This is a type of data augmentation for the minority class and is referred to as the **Synthetic Minority Oversampling Technique**, or **SMOTE** for short.

```
smote = SMOTE(random_state = 42)
X_train, y_train = smote.fit_resample(X_train, y_train)
```

```
y_train.value_counts()
```

```
target_H5_HPAI
0    10873
1    10873
Name: count, dtype: int64
```

Milestone 4: Model Building

Now our data is cleaned and it's time to build the model. We can train our data on ML Regression algorithms. For this project, we are applying four different regression models. The best model is saved based on its performance.

Activity 1: Decision Tree Classification

A function named DecisionTreeClassifier is created and train and test data are passed as the parameters. Inside the function, the Linear regression algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, accuracy_score, confusion matrix and classification report is used.

```
dt  = DecisionTreeClassifier()
dt.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

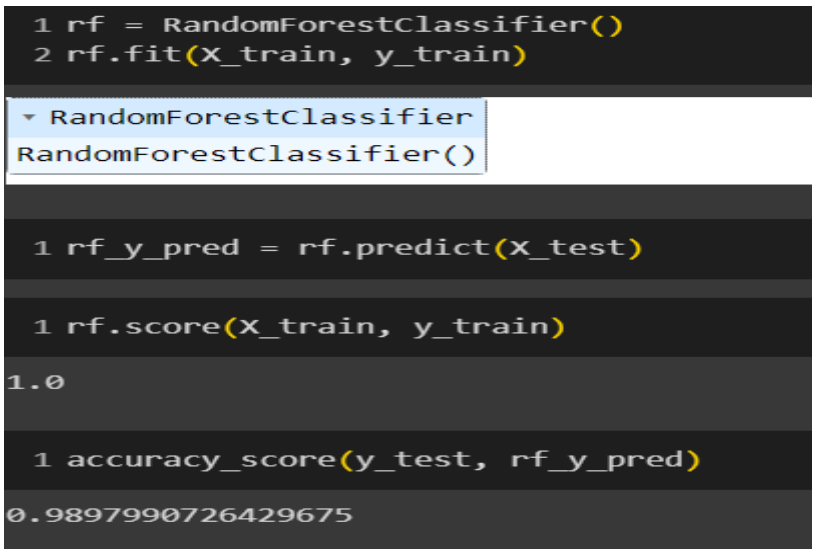
```
dt_pred = dt.predict(X_test)
```

```
accuracy_score(y_test, dt_pred)
```

```
0.9891808346213292
```

Activity 2: Random Forest Classification

A function named RandomForestClassifier is created and train and test data are passed as the parameters. Inside the function, the RandomForestRegressor algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, a accuracy_score, confusion matrix and classification report is used.



Activity 3: Evaluating the models

Confusion Matrix:

A confusion matrix is a table that is often used to evaluate the performance of a classification model. It provides a comprehensive summary of how well the model is performing in terms of making predictions correctly or incorrectly across different classes. The matrix consists of four terms:

- True Positives (TP):

The cases where the model correctly predicts the positive class.

- True Negatives (TN):

The cases where the model correctly predicts the negative class.

- False Positives (FP):

Also known as Type I error, these are the cases where the model incorrectly predicts the positive class when it's actually negative.

- False Negatives (FN):

Also known as Type II error, these are the cases where the model incorrectly predicts the negative class when it's actually positive.

From the confusion matrix, various performance metrics such as accuracy, precision, recall, and F1-score can be calculated.

Classification Report:

A classification report is a textual representation of the performance of a classification model. It provides a summary of different evaluation metrics for each class in the dataset. The common metrics included in a classification report are:

- Precision:

The ratio of correctly predicted positive observations to the total predicted positives (TP / (TP + FP)).

- Recall (also called Sensitivity):

The ratio of correctly predicted positive observations to the all observations in actual class (TP / (TP + FN)).

- F1-score:

The harmonic mean of precision and recall, which provides a balance between the two metrics.

- Support:

The number of actual occurrences of the class in the specified dataset.

Classification reports are useful for understanding how well a model performs on each class and can help in identifying classes where the model might be performing poorly.

ROC Curve (Receiver Operating Characteristic Curve):

The ROC curve is a graphical plot that illustrates the diagnostic ability of a binary classification model across various thresholds. It is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The TPR is also known as sensitivity, while the FPR is the complement of specificity.

The area under the ROC curve (AUC-ROC) is often used as a summary measure of the model's performance. A higher AUC-ROC value indicates a better-performing model, with an AUC of 1 representing a perfect classifier and an AUC of 0.5 indicating a random classifier.

The ROC curve is particularly useful when the classes are imbalanced or when you want to understand the trade-off between sensitivity and specificity at different thresholds.

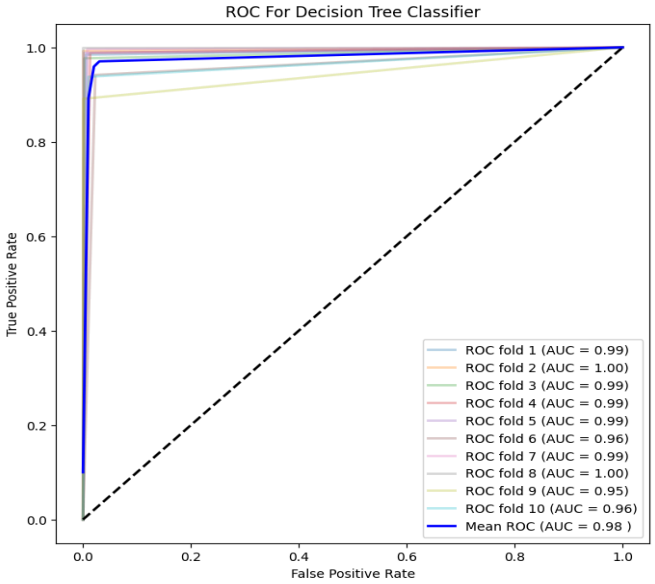
Decision Tree:

```
1 print(classification_report(y_test, dt_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      2728
           1       0.95      0.98      0.96       507

    accuracy                           0.99      3235
   macro avg       0.97      0.99      0.98      3235
weighted avg       0.99      0.99      0.99      3235
```

```
1 print(confusion_matrix(y_test, dt_pred))
```

```
[[2699   29]
 [   8  499]]
```



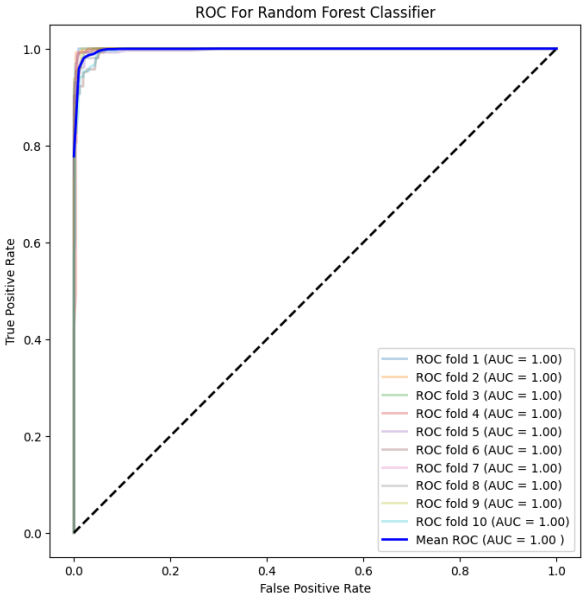
Random Forest Classifier :

```
1 print(classification_report(y_test, rf_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2728
           1       0.96      0.97      0.97       507

    accuracy                           0.99      3235
   macro avg       0.98      0.98      0.98      3235
weighted avg       0.99      0.99      0.99      3235
```

```
1 confusion_matrix(y_test, rf_y_pred)
```

```
array([[2709,   19],
       [  14,  493]])
```



Activity 4:Testing the model

Here we got good accuracy for RandomForestClassification, so we take that model for testing and prediction

```
tem1 = ['Ardea cinerea', 'Gray Heron', 2005, 8, 6, 'Munster', 'Clare', "Ballyvaughan Harbour", 53.1192, -9.1532, "Ardea cinerea", 9, 0]
```

```
for i in range(len(tem1)) :
  if type(tem1[i]) == str :
    tem1[i] = tem1[i].lower()
```

```
tem1[0] = sci_name_le.transform([tem1[0]])[0]
tem1[5] = state_le.transform([tem1[5]])[0]
tem1[6] = county_le.transform([tem1[6]])[0]
tem1[7] = loc_le.transform([tem1[7]])[0]
tem1[10] = par_spec_le.transform([tem1[10]])[0]
tem1[1] = com_name_le.transform([tem1[1]])[0]
```

```
tem1 = scaler.transform([tem1])
```

```
dt.predict(tem1), rf.predict(tem1)
```

```
(array([0]), array([0]))
```

Activity 5: Pickle Model save

RandomForestRegressor gives good accuracy so we save this model using the pickle for web integration**.** We can save our model using the below code.

```
pickle.dump(rf, open('rf_model.pkl', 'wb'))
```

```
pickle.dump(sci_name_le, open('sci_name_le.pkl', 'wb'))
pickle.dump(com_name_le,open('com_name_le.pkl', 'wb'))
pickle.dump(state_le, open('state_le.pkl', 'wb'))
pickle.dump(county_le, open('county_le.pkl', 'wb'))
pickle.dump(loc_le, open('loc_le.pkl', 'wb'))
pickle.dump(par_spec_le, open('par_spec_le.pkl', 'wb'))
pickle.dump(scaler, open('scaler.pkl', 'wb'))
```

After the pickle model is saved we have to download and save it in the project Folder.

Milestone 5: Application Building

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI. This section has the following tasks

- Building HTML Pages
- Building server side script
- Run the web application

Activity1: Building Html Pages:

For this project create three HTML files namely

- index.html
- prediction-page.html
- results.html

and save them in the templates folder in the project folder.

Activity 2: Build Python code:

Import the libraries

```
from flask import Flask, request, render_template, redirect, url_for
from markupsafe import Markup
import joblib
import numpy as np

app = Flask(__name__)
```

Load the saved model. Importing the Flask module in the project is mandatory. An object of the Flask class is our WSGI application. The Flask constructor takes the name of the current module (__name__) as an argument.

```
# Load the trained model
model = joblib.load('rf_model.pkl')
scaler = joblib.load('scaler.pkl')
com_name_le = joblib.load('com_name_le.pkl')
county_le = joblib.load('county_le.pkl')
loc_le = joblib.load('loc_le.pkl')
par_spec_le = joblib.load('par_spec_le.pkl')
sci_name = joblib.load('sci_name.pkl')
state_le = joblib.load('state_le.pkl')

@app.route('/')
def index():
    return render_template('index.html')
```

Render HTML page:

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict', methods = ['GET', 'POST'])
def predict():
```

Here we will be using a declared constructor to route to the HTML page that we have created earlier.
In the above example, the '/' URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the HTML page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.
Retrieves the value from UI:

```python
@app.route('/predict', methods = ['GET', 'POST'])
def predict():
    """
    The `predict` function processes form data, transforms categorical features, scales values, makes a
    prediction using a machine learning model, and redirects to the results page.
    :return: The `predict` function is returning a redirect to the 'results' page with the predicted
    value as a parameter if the request method is 'POST'. If the request method is not 'POST', it is
    rendering the 'prediction-page.html' template.
    """
    if request.method == 'POST':
        # Extract form data
        features = [
            str(request.form['Scientific_Name']).lower(),
            str(request.form['Common_Name']).lower(),
            int(request.form['Year']),
            int(request.form['Month']),
            int(request.form['Day']),
            str(request.form['State']).lower(),
            str(request.form['County']).lower(),
            str(request.form['Locality']).lower(),
            float(request.form['Latitude']),
            float(request.form['Longitude']),
            str(request.form['Parent_Species']).lower(),
            int(request.form['Hour']),
            int(request.form['Minute'])
        ]

        # Convert features to a numpy array
        features_array = np.array(features)
```

We're converting string-type data into lowercase because our label encoders were trained with lowercase text. However, users may occasionally input incorrect names, such as spelling errors. In such cases, we handle this by providing a JSON file that displays the correct class labels for users to reference.

```python
    # Checking for valid labels
    if features_array[0].lower() not in sci_name_le.classes_ :
        return jsonify({'error': 'Unseen label detected', 'message': 'Please enter a correct label.', 'valid_labels': list(sci_name_le.classes_)})
    if features_array[1].lower() not in com_name_le.classes_ :
        return jsonify({'error': 'Unseen label detected', 'message': 'Please enter a correct label.', 'valid_labels': list(com_name_le.classes_)})
    if features_array[5].lower() not in state_le.classes_ :
        return jsonify({'error': 'Unseen label detected', 'message': 'Please enter a correct label.', 'valid_labels': list(state_le.classes_)})
    if features_array[6].lower() not in county_le.classes_ :
        return jsonify({'error': 'Unseen label detected', 'message': 'Please enter a correct label.', 'valid_labels': list(county_le.classes_)})
    if features_array[7].lower() not in loc_le.classes_ :
        return jsonify({'error': 'Unseen label detected', 'message': 'Please enter a correct label.', 'valid_labels': list(loc_le.classes_)})
    if features_array[10].lower() not in par_spec_le.classes_ :
        return jsonify({'error': 'Unseen label detected', 'message': 'Please enter a correct label.', 'valid_labels': list(par_spec_le.classes_)})

    # Using Label Encoder to transform categorical features
    features_array[0] = sci_name_le.transform([features_array[0]])[0]
    features_array[1] = com_name_le.transform([features_array[1]])[0]
    features_array[5] = state_le.transform([features_array[5]])[0]
    features_array[6] = county_le.transform([features_array[6]])[0]
    features_array[7] = loc_le.transform([features_array[7]])[0]
    features_array[10] = par_spec_le.transform([features_array[10]])[0]

    # Scaling the values
    array_scaled = scaler.transform([features_array])

    # Predict the class
    prediction = model.predict(array_scaled)[0]

    # Redirect to the prediction page
    return redirect(url_for('results', prediction=prediction))

return render_template('prediction-page.html')
```

This function retrieves all the values from the HTML page using a Post request. That is stored in an array. This array is passed on to the scaler model to normalise our data and then this array is passed to the model. predict() function. This function returns the prediction. This prediction value will be rendered to the text that we have mentioned in the prediction-page.html(results.html) page earlier.

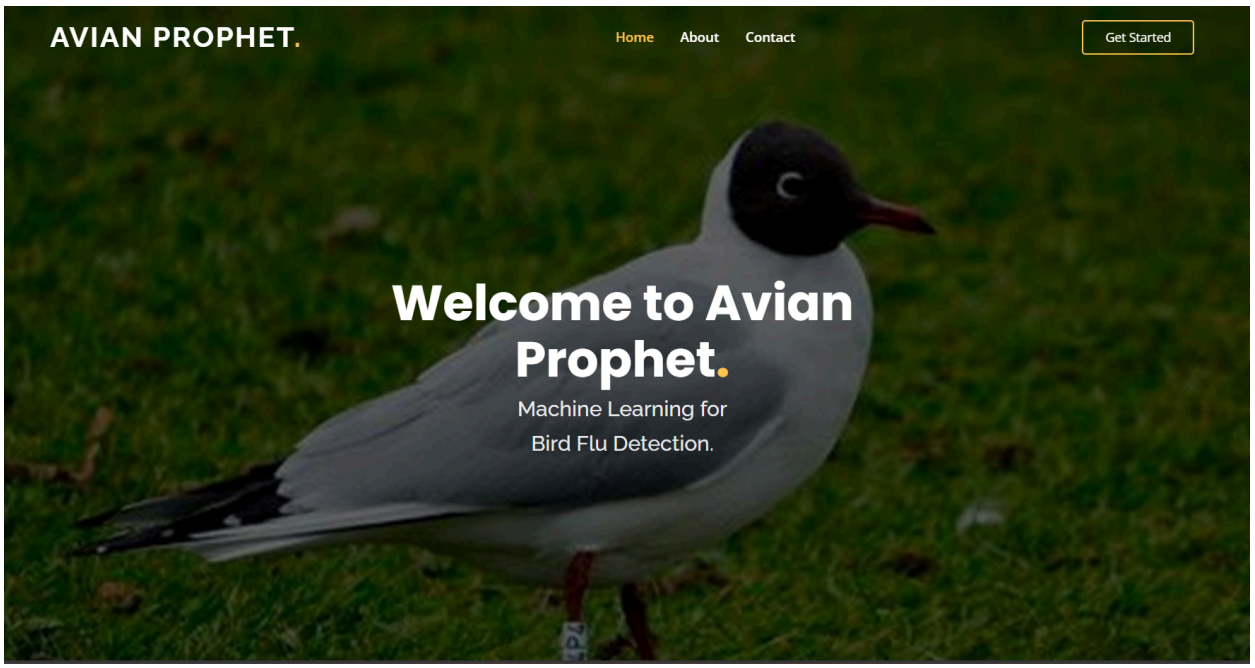
Main Function:

```python
def results():
    The `results` function retrieves a prediction from query parameters and renders it in a template
    called 'results.html'.
    :return: The `results()` function is returning a rendered template called 'results.html' with the
    prediction value passed as a parameter.
    """
    # Get the prediction from the query parameters
    prediction = request.args.get('prediction')

    # Making sense of the prediction
    if prediction :
        prediction = 'Your Bird might have the Bird Flu'
    else :
        prediction = 'Your Bird is safe!'

    return render_template('results.html', prediction=prediction)

if __name__ == '__main__':
    app.run(debug=True)
```

Activity 3: Run the application

- Open the anaconda prompt from the start menu
- Navigate to the folder where your Python script is.
- Now type the “python app.py” command
- Navigate to the localhost where you can view your web page.

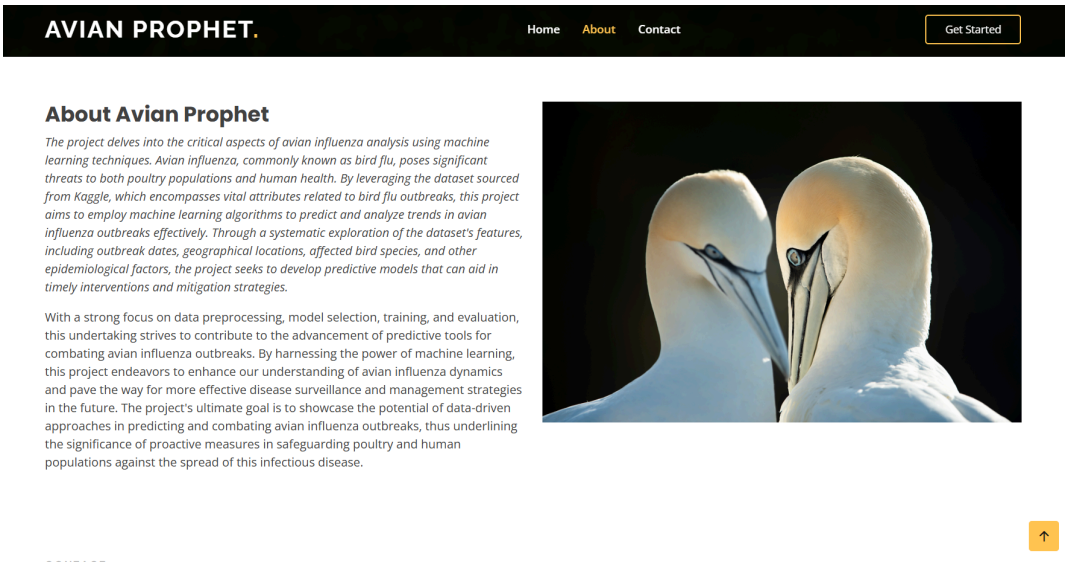
Now, Go to the web browser and write the localhost:5000 to get the below result Let’s see what our index.html page looks like:



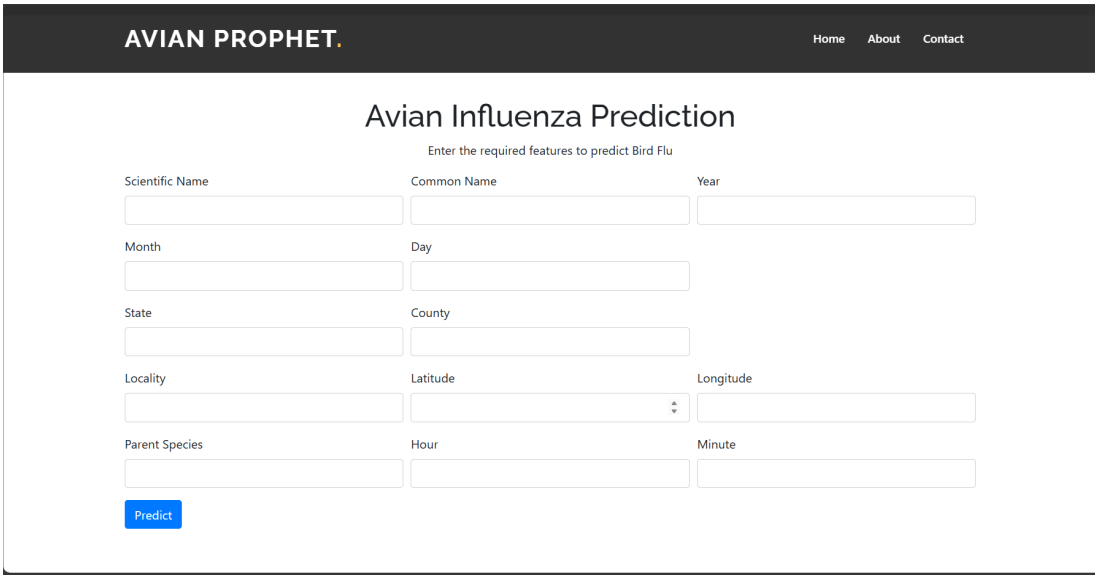
This is the home page of our web application. Now when you click on the **Get Started** page you will get redirected to predict.html When you click on the About page you will get about
this project is about page. We have a Home page, About page and

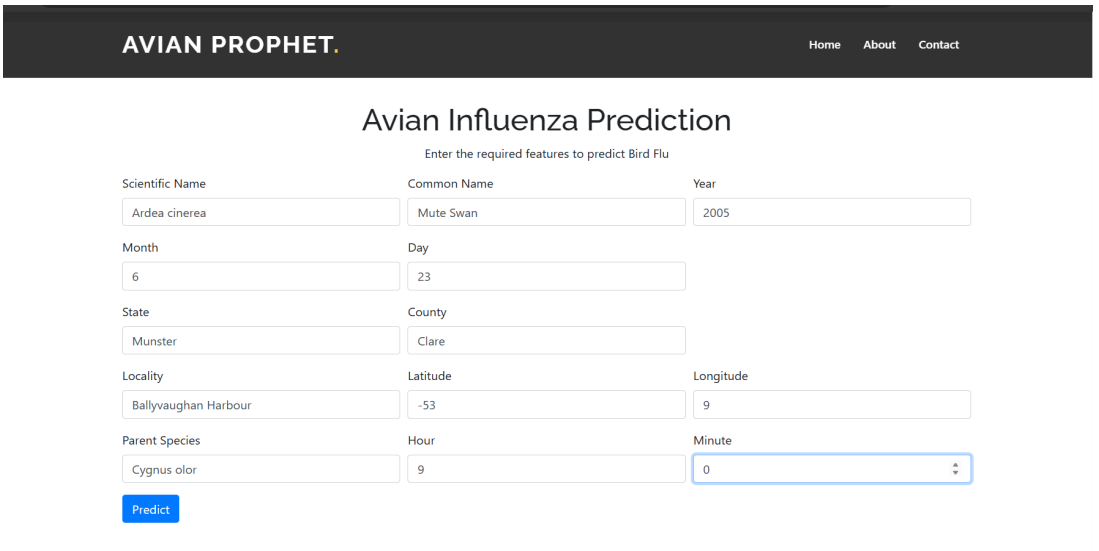
Get Started page(prediction-page.html). when we click any of them we directly go to that page.

Let’s see what our About page looks like:



Let’s see what our prediction-page.html page looks like:





Now when you click on the submit button you will get redirected to results.html.
Let's look at what our results.html file looks like: