ANIMAL HEALTH CLASSIFICATION SYSTEM WITH MACHINE LEARNING

TEAM MEMBERS:

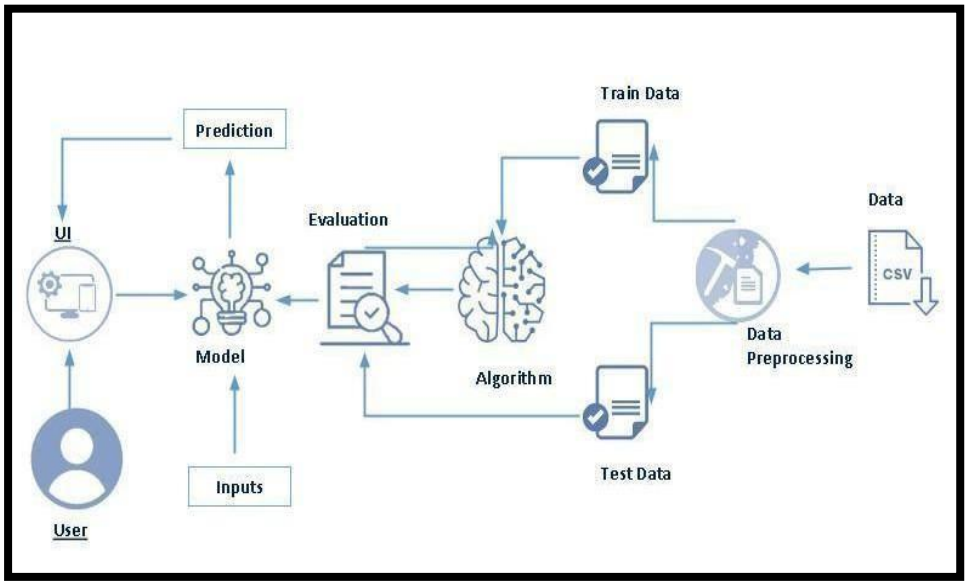
1. AASHISH NAVAL (23CS001)
2. ANKUSH SONI (23CS016)
3. KUSUM KEER (23CS050)
4. MAYURI MANGAL (23CS059)

Made By Students of Anand International College of Engineering, Jaipur

Animal Health Classification System Using Machine Learning

Harmonizing Health and Machine is an AI-driven initiative aimed at transforming animal health assessment through advanced machine learning and deep learning techniques. By analyzing behavioral, physiological, and environmental data from a variety of animal species, the system offers early detection of illnesses, continuous monitoring, and objective diagnosis. Whether on farms, in zoos, or at home with pets, this innovative platform enhances animal welfare by providing timely, data-backed insights—bridging the gap between traditional care and intelligent automation.

TECHINICAL ARCHITECTURE:



Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI To accomplish this, we have to complete all the activities listed below:
  - Define problem / Problem understanding
  - Specify the problem statement
  - Business Requirements
  - Social or Business Impact
  - Data Collection and Preparation
  - Collect the dataset
  - Data Preparation
  - Exploratory Data Analysis
  - Descriptive statistical
  - Visual Analysis
  - Model Building
  - Train-test split
  - Training and testing the Models using multiple algorithms
  - Performance Testing
  - Comparing model accuracy
  - Model Deployment
  - Save the best model
  - Test the model
  - Integrate with Web Framework
  - GUI
  - Project Demonstration & Documentation

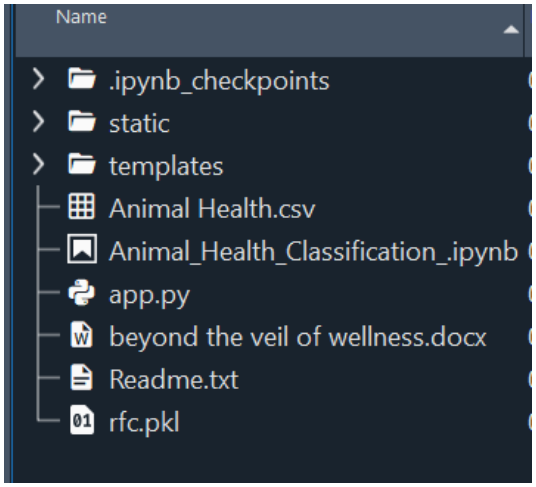
Prior Knowledge:

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

- ML Concepts
  - Supervised learning: https://www.javatpoint.com/supervised-machine-learning
  - Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
    - Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
    - Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
    - KNN: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
    - Xgboost: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/
    - Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/
- Flask Basics : https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:

Create the Project folder which contains files as shown below



- The data obtained is in csv files splitted for training and testing.
- We are building a Flask application which will require the html files to be stored in the templates folder.
- app.py file is used for routing purposes using scripting.
- rfc.pkl is the saved model. This will further be used in the Flask integration.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the Problem Statement.

The purpose of the Beyond the Veil of Wellness: Machine Learning's Unique Journey in Animal Health Classification is to assess the health condition of animals and its status whether it is in critical or in normal state.

Activity 2: Business Requirements.

Risk assessment: Conduct a comprehensive risk assessment by analyzing factors such as animal age, breed, vaccination history, and environmental conditions to determine the likelihood of disease outbreaks. Use historical data and predictive models to prioritize high-risk groups for proactive monitoring and intervention.

Financial planning: Create a budget that includes provisions for routine healthcare, emergency veterinary expenses, and potential losses due to illness. Establish an emergency fund to cover unexpected costs, and explore insurance options for long-term financial security in case of severe health issues in your animals.

Early warning system: Develop an early warning system that integrates sensor technology to monitor vital signs and behavior patterns in animals. Implement machine learning algorithms to detect deviations from baseline health indicators, enabling timely intervention and reducing the impact of diseases on livestock or pets.

Milestone 2: Data Collection & Preparation

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: Collect the dataset

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

In this project we have used .csv 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/saumyamohandas/animal-health-classification?select=Animal+Health.csv

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 a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualisation style as fivethirtyeight.

```
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt # plotting
import numpy as np # linear algebra
import os # accessing directory structure
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

#Import the libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import f1_score, make_scorer, accuracy_score, recall_score, precision_score
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

import random

import warnings
warnings.filterwarnings("ignore")
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the
#input directory
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, 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 directory of the csv file.

```
# Reading the csv file
df=pd.read_csv("insurance_claims.csv")
df.head()
```

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.91	0	466132
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.22	5000000	468176
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.14	5000000	430632
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.74	6000000	608117
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.91	6000000	610706

5 rows × 40 columns

Activity 2: Data Preparation

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 in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling Outliers

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 2.1: Getting the Preliminary Information about the Dataset

- For checking the null values, df.isna().any( ) function is used. To sum those null values we use .sum() function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 871 entries, 0 to 870
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   AnimalName         871 non-null    object
 1   BloodBrainDisease  871 non-null    object
 2   AppearenceDisease  871 non-null    object
 3   GeneralDisease     871 non-null    object
 4   LungDisease        871 non-null    object
 5   AbdominalDisease   871 non-null    object
 6   HealthStatus       869 non-null    object
dtypes: object(7)
memory usage: 47.8+ KB
```

```
df.describe()
```

	AnimalName	BloodBrainDisease	AppearenceDisease	GeneralDisease	LungDisease	AbdominalDisease	HealthStatus
count	871	871	871	871	871	871	869
unique	41	5	6	13	6	6	2
top	Buffaloes	anemia	emaciation	vomiting	halitosis	upset stomach	Critical
freq	129	207	168	79	173	164	849

Activity 2.2: Handling missing values:

Check and number of missing values for the all the columns and filled the missing values only for required columns (Input).

For checking the null values, df.isnull() function is used. To sum those null values, we use .sum()

```
df.isnull().sum()
```

```
AnimalName           0
BloodBrainDisease    0
AppearenceDisease    0
GeneralDisease       0
LungDisease          0
AbdominalDisease     0
HealthStatus         2
dtype: int64
```

```
df['HealthStatus'].unique()
```

```
array(['Critical', 'Normal', nan], dtype=object)
```

```
df['HealthStatus'].value_counts()
```

```
HealthStatus
Critical    849
Normal       20
Name: count, dtype: int64
```

```
df['HealthStatus'].fillna('Critical',inplace=True)
```

Now we replaced 2 null values from Health status column with the mode value of column as below.

```
df['HealthStatus'].fillna('Critical',inplace=True)
```

```
df.isnull().sum()
```

```
AnimalName           0
BloodBrainDisease    0
AppearenceDisease    0
GeneralDisease       0
LungDisease          0
AbdominalDisease     0
HealthStatus         0
dtype: int64
```

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study of the basic features of data with the statistical process. Here pandas have a worthy function called describe. With this describe 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.

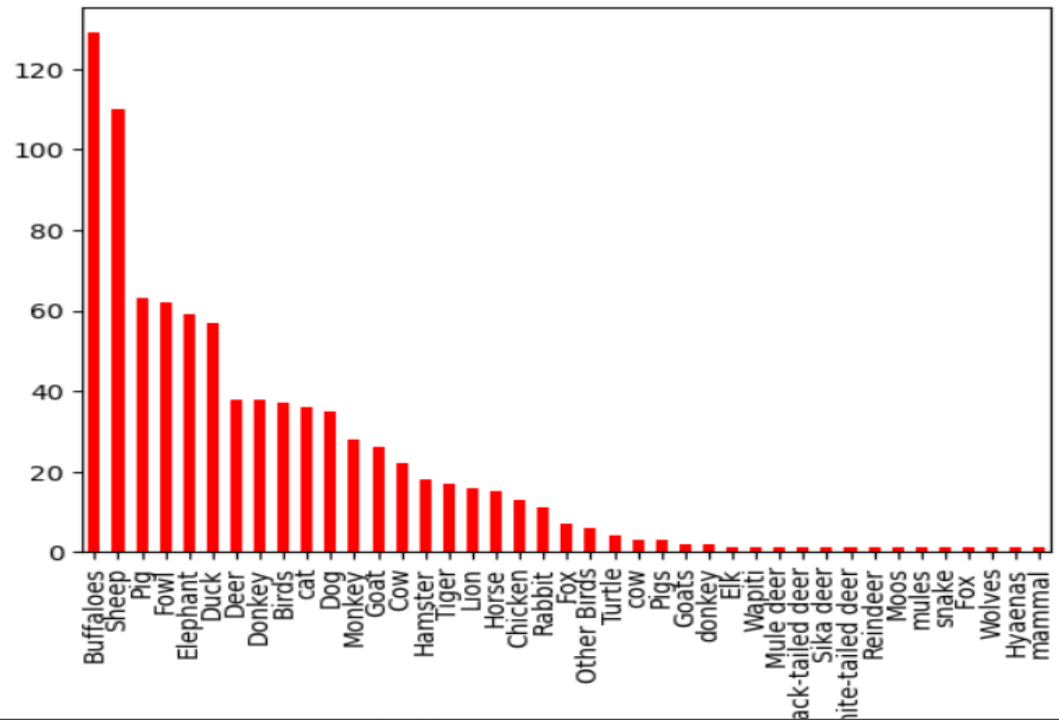
```
df.nunique()
```

```
AnimalName            41
BloodBrainDisease      5
AppearenceDisease      6
GeneralDisease        13
LungDisease            6
AbdominalDisease       6
HealthStatus           2
dtype: int64
```

Activity 2.1 Class Analysis:

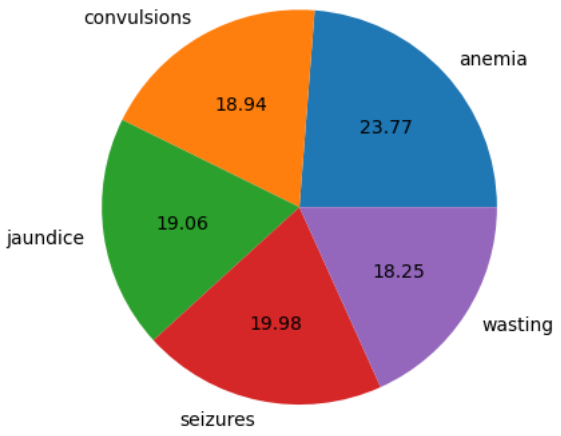
```
df['AnimalName'].value_counts().plot(kind='bar',color='Red')
```

```
<Axes: xlabel='AnimalName'>
```

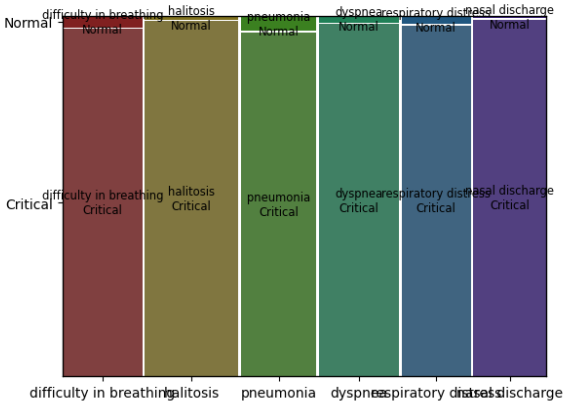


```
df.groupby('BloodBrainDisease').size().plot(kind='pie', autopct='%.2f')
```

```
<Axes: >
```

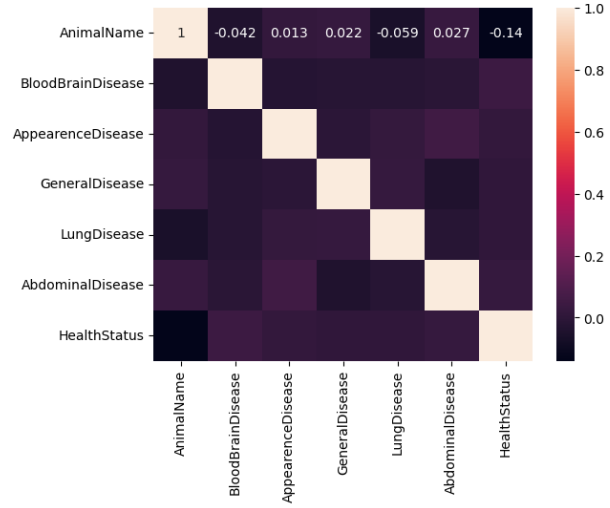



```
import matplotlib.pyplot as plt
from statsmodels.graphics.mosaicplot import mosaic
mosaic(df, ['LungDisease','HealthStatus'])
plt.show()
```




```
sns.heatmap(df.corr(),annot=True)
```

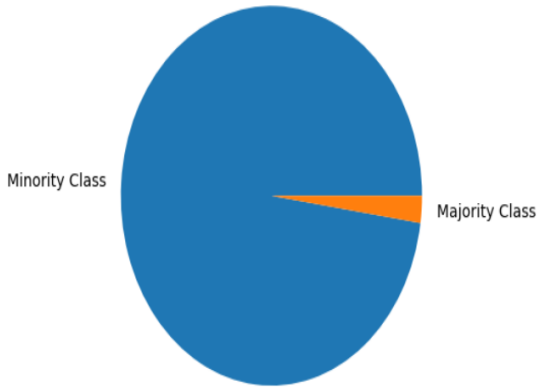
```
<Axes: >
```



In the graph we can see that most of the column is filled with critical , so we need to balance the dataset.

```
class_division = [ df[df['HealthStatus'] == 'Critical' ].shape[0], df[df['HealthStatus'] == 'Normal' ].shape[0] ]
my_labels = ['Minority Class', 'Majority Class']

plt.pie(class_division, labels = my_labels)
plt.show()
```



Activity 3: Encoding Data

All the columns we are having in our data are categorical so we need to encode them .

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

```
df['AnimalName']=le.fit_transform(df['AnimalName'])
```

```
df['BloodBrainDisease']=le.fit_transform(df['BloodBrainDisease'])
df['AppearenceDisease']=le.fit_transform(df['AppearenceDisease'])
df['GeneralDisease']=le.fit_transform(df['GeneralDisease'])
df['LungDisease']=le.fit_transform(df['LungDisease'])
df['AbdominalDisease']=le.fit_transform(df['AbdominalDisease'])
df['HealthStatus']=le.fit_transform(df['HealthStatus'])
```

```
df
```

	AnimalName	BloodBrainDisease	AppearenceDisease	GeneralDisease	LungDisease	AbdominalDisease	HealthStatus
0	6	0	4	2	0	5	0
1	6	0	0	4	0	3	0
2	6	2	1	4	2	3	0
3	6	4	2	6	4	5	0
4	6	1	3	12	1	2	0
...	...	...	...	...	...	...	...
866	2	3	1	12	3	1	0
867	2	1	2	10	4	3	0
868	2	4	1	3	4	3	0
869	2	0	1	4	0	1	0
870	2	0	5	12	1	3	0

871 rows × 7 columns

Milestone 4: Model Building

Activity 1 Dealing with Imbalanced data

Target class Imballance : SMOTE

very low % of the companies has Bankrupted in the dataset,making it imballanced target class probelm. Hence positive target class(Bankrupt=1) is under-represented.This could be challenging as lack of positive class in the train data may lead machine learning model to have poor performance in terms of detecting positive class in the unseen data. SMOTE(Synthetic Minority Oversampling Techniwque) proposed by Chawla et al 2002, is a well applied technique to handle such scenerio. SMOTE actually creates as many synthetic examples for minority class as are requirred so that finally two target class are well represented. It does so by synthesising samples that are close to the feature space ,for the minority target class. More about SMOTE

```
from imblearn.over_sampling import SMOTE
```

```
smote = SMOTE(sampling_strategy='auto', random_state=42)
xbal,ybal=smote.fit_resample(xtrain,ytrain)
```

```
ybal.value_counts()
```

```
HealthStatus
0    595
1    595
Name: count, dtype: int64
```

Activity 2 Train-test split

Now let's split the Dataset into train and test sets. First split the dataset into X and y and then split the data set.

```
x = df.drop(['HealthStatus'],axis=1)
y = df['HealthStatus']
```

```
from sklearn.preprocessing import StandardScaler
stx= StandardScaler()
X = stx.fit_transform(x)
```

Splitting data

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.3, random_state=42, stratify=y)
```

Here X and y variables are created. On X variable, data is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing X, y, test_size, random_state.

Splitting data into train and test

Now let's split the Dataset into train and test sets. First split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

```
x=daata1.iloc[:,0:30]
y=daata1.iloc[:,30:]
```

```
[ ] from sklearn.model_selection import train_test_split
    X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

Handling Imbalanced dataset

- Imbalanced data is a common problem in machine learning and data analysis, where the number of observations in one class is significantly higher or lower than the other class. Handling imbalanced data is important to ensure that the model is not biased towards the majority class and can accurately predict the minority class.
- Here we are using SMOTE Technique.

```
[ ] from imblearn.over_sampling import SMOTE
    smt=SMOTE()
    X_train, y_train = smt.fit_resample(X_train, y_train)
```

Scaling

- Scaling is a technique used to transform the values of a dataset to a similar scale to improve the performance of machine learning algorithms. Scaling is important because many machine learning algorithms are sensitive to the scale of the input features.
- Here we are using Standard Scaler.
- This scales the data to have a mean of 0 and a standard deviation of 1. The formula is given by:
  X_scaled = (X - X_mean) / X_std

```
from sklearn.preprocessing import StandardScaler
Std_scaler=StandardScaler()
```

```
X_train = Std_scaler.fit_transform(X_train)
X_train = pd.DataFrame(X_train, columns=x.columns)
```

```
X_test = Std_scaler.transform(X_test)
X_test = pd.DataFrame(X_test, columns=x.columns)
```

Activity 3: Training the model in multiple algorithms

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

Activity 1.1: Decision tree model

First Decision Tree is imported from sklearn Library then DecisionTreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. We can find the Train and Test accuracy by X_train and X_test.

```
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
dtc.fit(X_train,y_train)
y_pred=dtc.predict(X_test)
dtc_train_acc=accuracy_score(y_train,dtc.predict(X_train))
dtc_test_acc=accuracy_score(y_test,y_pred)
```

Activity 1.2: Random forest model

First Random Forest Model is imported from sklearn Library then RandomForestClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in a new variable. We can find the Train and Test accuracy by X_train and X_test.

```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(criterion='entropy',max_depth=10,max_features='sqrt',min_samples_leaf= 1, min_samples_split= 3, n_estimators= 140)
rfc.fit(X_train,y_train)
y_pred=rfc.predict(X_test)
rfc_train_acc=100*accuracy_score(y_train,rfc.predict(X_train))
rfc_test_acc=100*accuracy_score(y_test,y_pred)
```

Activity 1.3: KNN model

KNN Model is imported from sklearn Library then KNeighborsClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=30)
knn.fit(X_train,y_train)
y_pred=knn.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

Activity 1.4: Logistic Regression model

Logistic Regression Model is imported from sklearn Library then Logistic Regression algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix is done.

```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, classification_report
lg = LogisticRegressionCV(solver='lbfgs', max_iter=5000, cv=10)
lg.fit(X_train, y_train)
print(confusion_matrix(y_test,lrg_pred))
```

Activity 1.5: Naïve Bayes model

Naïve Bayes Model is imported from sklearn Library then Naïve Bayes algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. We can find the Train and Test accuracy by X_train and X_test.

```
from sklearn.naive_bayes import CategoricalNB,GaussianNB
gnb=GaussianNB()
model_2=gnb.fit(X_train,y_train)
predict_log=model_2.predict(X_test)
print("Training Accuracy",100*accuracy_score(model_2.predict(X_train),y_train))
print("Testing Accuracy",100*accuracy_score(y_test,predict_log))
```

Activity 1.6: SVM model

SVM Model is imported from sklearn Library then SVM algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
: from sklearn.svm import SVC

  svc = SVC()
  svc.fit(X_train, y_train)
  y_pred = svc.predict(X_test)
```

```
: svc_train_acc = accuracy_score(y_train, svc.predict(X_train))
  svc_test_acc = accuracy_score(y_test, y_pred)
  print(f"Training accuracy of SVC : {svc_train_acc}")
  print(f"Test accuracy of SVC : {svc_test_acc}")
  print(confusion_matrix(y_test,y_pred))
  print(classification_report(y_test,y_pred))
```

Activity 2: Testing the model

Here we have tested with Decision Tree algorithm. You can test with all algorithm. With the help of predict() function.

```
b_dtc.predict([[328,521585,2012,12,250,1000,1406.91,5600,1,100,25,25,50000,0,120,23,56,52,1,123,2,3,1,0,2,1,150000,2,25,2002]])
```

```
]: array([0])
```

Milestone 5: Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with multiple evaluation metrics

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using evaluation metrics for classification tasks including accuracy, precision, recall, support and F1-score.

Activity 1.1: Compare the model

For comparing the above five models, the accuracy_df function is used.

Below is the accuracy comparison of all the models and we can clearly see that accuracy for Logistics Regression, Decision Tree and Random Forest is 95 percent so we can take any of this model for our classification purpose.

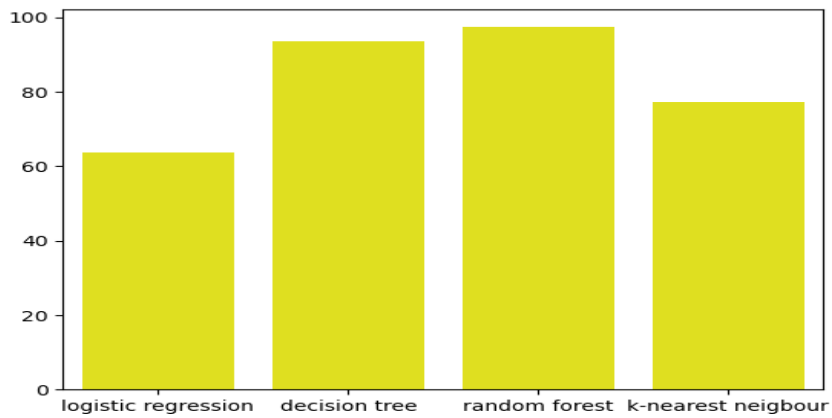
```
accuracy_df=pd.DataFrame({'model':['logistic regression','decision tree','random forest','k-nearest neigbour'],
                          'Accuracy':[acc_lr*100,acc_dt*100,acc_rfc*100,acc_knn*100]})
print(accuracy_df)
```

```
                model   Accuracy
0  logistic regression  63.740458
1        decision tree  93.511450
2        random forest  96.946565
3   k-nearest neigbour  77.099237
```

Activity 2: Graphical representation of the model comparison

```
models=['logistic regression','decision tree','random forest','k-nearest neigbour']
accuracy=[acc_lr*100,acc_dt*100,acc_rfc*100,acc_knn*100]
sns.barplot(x=models,y=accuracy,color='Yellow')
plt.show()
```

```
C:\Users\boina\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categ
be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\boina\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categ
be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\boina\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1765: FutureWarning: unique w
es, Index, ExtensionArray, or np.ndarray is deprecated and will raise in a future version.
  order = pd.unique(vector)
C:\Users\boina\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categ
be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



Milestone 6: Model Deployment

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```
import pickle
pickle.dump(rfc,open("rfc.pkl","wb"))
```

Activity 2: Test the model:

Let's test the model first in python notebook itself.

As we have 5 features in this model, let's check the output by giving all the inputs.

```
print(rfc.predict([[4,3,2,6,4,5]]))
```

```
[0]
```

We can see above that out model has predicted "0", that means model has classified this as Critical condition.

Hence, we can conclude that, out model is giving the accurate results.

Activity 3: Integrate with Web Framework

In this section, we will be building a web application that is integrated to 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.

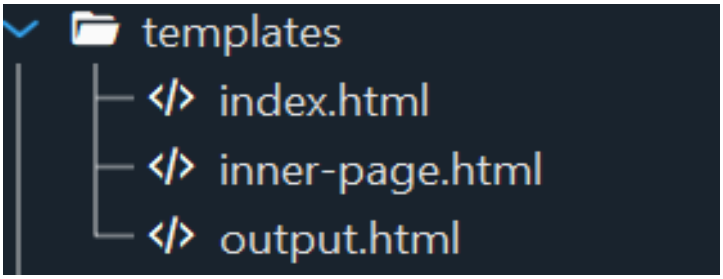
This section has the following tasks:

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

Activity 3.1: Building HTML pages:

For this project create two HTML files namely and save them in the templates folder.

- Index.html
- Predict .html



Activity 3.2: Build Python code

Create a new app.py file which will be store in the Flask folder.

- Import the necessary Libraries.

Render HTML page:

```
from flask import Flask, request, render_template


app = Flask(__name__)
model = pickle.load(open(r'rfc.pkl', 'rb'))
#scale = pickle.load(open(r'scale1.pkl','rb'))

@app.route('/') # rendering the html template
def home():
    return render_template('index.html')
```

```
@app.route('/predict',methods=["POST","GET"]) # rendering the html template
def predict() :
    return render_template("inner-page.html")

@app.route('/submit',methods=["POST","GET"])# route to show the predictions in a web UI
def submit():
    #  reading the inputs given by the user
    input_feature=[int(x) for x in request.form.values() ]
    #input_feature = np.transpose(input_feature)
    input_feature=[np.array(input_feature)]
    print(input_feature)
    names = ['AnimalName', 'BloodBrainDisease', 'AppearenceDisease',
       'GeneralDisease', 'LungDisease', 'AbdominalDisease']
    data = pandas.DataFrame(input_feature,columns=names)
    print(data)

    #data_scaled = scale.fit_transform(data)
    #data = pandas.DataFrame(,columns=names)



     # predictions using the loaded model file
    prediction=model.predict(data)
    print(prediction)
    prediction = int(prediction)
    print(type(prediction))

    if (prediction == 0):
       return render_template("output.html",result ="According to our study we feel sad
    else:
       return render_template("output.html",result = "You are health is in normal condi
     # showing the prediction results in a UI
```

Setting up the application This code defines a Flask web application that uses a pre-trained Random Forest model to predict the status ASD detected or not. The joblib library is used to load the pre-trained Random Forest model from a .pkl file. Then, a route / is defined that renders the index.html template, which contains a form that allows the user to input the features. When the user submits the form, the predict() function is called, which retrieves the input values from the form and computes the Detect ASD. Then, the Random Forest model is used to predict based on the input values, and the predicted status is returned as the output.

Activity 4: GUI:

The GUI (Graphical User Interface) created in this Flask application is designed to predict the Animal Health Status bases on below features:

- AnimalName
- BloodBrainDisease
- AppearenceDisease
- GeneralDisease
- LungDisease
- AbdominalDisease

The user can input this feature in a form provided in the home page of the web application. After clicking on the "Lets check the health status" button, the application will predict the level of freedom of that country based on the rules and the random forest model defined in the Python script.

[Pictures of the WEB APP]

Milestone 7: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables

Activity 1:- Record explanation Video for project end to end solution

Activity 2:- Project Documentation-Step by step project development procedure