Activity 1.1: Importing the libraries

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

```
: import numpy as np # scientific computation
  import pandas as pd # loading dataset file
  import matplotlib.pyplot as plt # Visulization
  import nltk  # Preprocessing our text
  from nltk.corpus import stopwords # removing all the stop words
  from nltk.stem.porter import PorterStemmer # stemming of words
```

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.

```
#Load our dataset
df = pd.read_csv("spam.csv",encoding="latin")
df.head()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

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 categorical data
- Handling Imbalance Data

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: Handling missing values

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

```
#Give concise summary of a DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   v1          5572 non-null   object
 1   v2          5572 non-null   object
 2   Unnamed: 2  50 non-null     object
 3   Unnamed: 3  12 non-null     object
 4   Unnamed: 4  6 non-null      object
dtypes: object(5)
memory usage: 217.8+ KB
```

- For checking the null values, df.isnull() 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.

```
]: #Returns the sum fo all na values
   df.isna().sum()
```

```
]: v1              0
   v2              0
   Unnamed: 2    5522
   Unnamed: 3    5560
   Unnamed: 4    5566
   dtype: int64
```

- From the above code of analysis, we can infer that columns such as V1 and v2 are not having missing columns,unnamed columns are not required for analysis

- Renaming the columns according the requirement

```
: df.rename({"v1":"label","v2":"text"},inplace=True,axis=1)
```

```
: # bottom 5 rows of the dataframe
  df.tail()
```

:

	label	text	Unnamed: 2	Unnamed: 3	Unnamed: 4
5567	spam	This is the 2nd time we have tried 2 contact u...	NaN	NaN	NaN
5568	ham	Will Ì_ b going to esplanade fr home?	NaN	NaN	NaN
5569	ham	Pity, * was in mood for that. So...any other s...	NaN	NaN	NaN
5570	ham	The guy did some bitching but I acted like i'd...	NaN	NaN	NaN
5571	ham	Rofl. Its true to its name	NaN	NaN	NaN

Activity 2.2: Handling Categorical Values

As we can see our dataset has categorical data we must convert the categorical data to integer encoding or binary encoding.

To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using manual encoding with the help of list comprehension.

- In our project,we have text column so we will be using natural language processing for processing the data. Output column is having classes we Converting into 0 and 1 by applying label encoding

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['label'] = le.fit_transform(df['label'])
```

Activity 2.3:Handling Imbalance Data

Data Balancing is one of the most important step, which need to be performed for classification models, because when we train our model on imbalanced dataset ,we will get biassed results, which means our model is able to predict only one class element For Balancing the data we are using the SMOTE Method.

SMOTE: Synthetic minority over sampling technique, which will create new synthetic data points for under class as per the requirements given by us using KNN method.

```
#Splitting data into train and validation sets using train_test_split

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

##train size 80% and test size 20%
```

```
### Given data is imbalanced one, we are balancing the data
```

```
print("Before OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train == 0)))

# import SMOTE module from imblearn library
# pip install imblearn (if you don't have imblearn in your system)
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 2)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel())

print('After OverSampling, the shape of train_X: {}'.format(X_train_res.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_res.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train_res == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res == 0)))
```

```
Before OverSampling, counts of label '1': 581
Before OverSampling, counts of label '0': 3876

After OverSampling, the shape of train_X: (7752, 7163)
After OverSampling, the shape of train_y: (7752,)

After OverSampling, counts of label '1': 3876
After OverSampling, counts of label '0': 3876
```

From the above picture, we can infer that ,previously our dataset had 581 class 1, and 3876 class 0 items, after applying smote technique on the dataset the size has been changed for minority class.

Activity 2.3: Cleaning the text data

```
nltk.download("stopwords")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\smart\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
True
```

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
```

```
import re
corpus = []
length = len(df)
```

```
for i in range(0,length):
    text = re.sub("[^a-zA-Z0-9]"," ",df["text"][i])
    text = text.lower()
    text = text.split()
    pe = PorterStemmer()
    stopword = stopwords.words("english")
    text = [pe.stem(word) for word in text if not word in set(stopword)]
    text = " ".join(text)
    corpus.append(text)
```

Text pre-processing includes

- Removing punctuation from the text using regular expression library
- Converting the sentence into lower case
- Tokenization – splitting the sentence into words
- Removing stop words from the data
- Stemming – stemming is the process of brining all the words into base form

```
[18]: corpus
```

```
[18]: ['go jurong point crazi avail bugi n great world la e buffet cine got amor wat',
 'ok lar joke wif u oni',
 'free entri 2 wkli comp win fa cup final tkt 21st may 2005 text fa 87121 receiv entri question std txt rate c appli 08452810
075over18',
 'u dun say earli hor u c alreadi say',
 'nah think goe usf live around though',
 'freemsg hey darl 3 week word back like fun still tb ok xxx std chg send 1 50 rcv',
 'even brother like speak treat like aid patent',
 'per request mell mell oru minnaminungint nurungu vettam set callertun caller press 9 copi friend callertun',
 'winner valu network custom select receivea 900 prize reward claim call 09061701461 claim code kl341 valid 12 hour',
 'mobil 11 month u r entitl updat latest colour mobil camera free call mobil updat co free 08002986030',
 'gonna home soon want talk stuff anymor tonight k cri enough today',
 'six chanc win cash 100 20 000 pound txt csh11 send 87575 cost 150p day 6day 16 tsandc appli repli hl 4 info',
 'urgent 1 week free membership 100 000 prize jackpot txt word claim 81010 c www dbuk net lccltd pobox 4403ldnw1a7rw18',
 'search right word thank breather promis wont take help grant fulfil promis wonder bless time',
 'date sunday',
 'xxxmobilemovieclub use credit click wap link next txt messag click http wap xxxmobilemovieclub com n qjkgighjjgcbl',
 'oh k watch',
 'eh u rememb 2 spell name ye v naughti make v wet',
```

```
[19]: from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=35000)
X = cv.fit_transform(corpus).toarray()
```

- After applying all the above functions, we will get corpus

- Converting the corpus into Document Term matrix using Count vectorizer

```
import pickle ## importing pickle used for dumping models
pickle.dump(cv, open('cv1.pkl', 'wb')) ## saving to into cv.pkl file
```

Saving the count vectorizer function for future use

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has 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.describe()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
count	5572	5572	50	12	6
unique	2	5169	43	10	5
top	ham	Sorry, I'll call later	bt not his girlfrnd... G o o d n i g h t . . .@"	MK17 92H. 450Ppw 16"	GNT:-)"
freq	4825	30	3	2	2

```
df.shape
```

```
(5572, 5)
```

Activity 2: Visual analysis

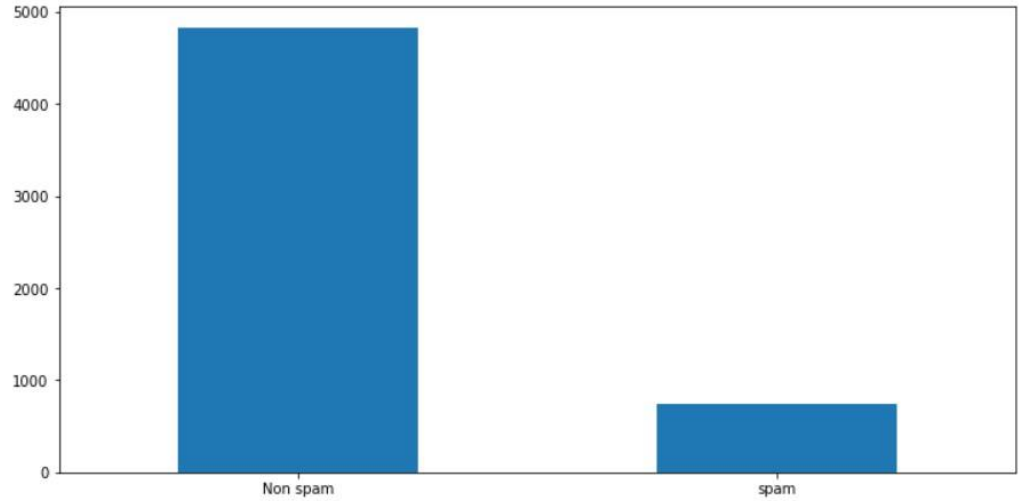
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 2.1: Univariate analysis

In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed two different graphs such as distplot and countplot.

- The Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature. To make multiple graphs in a single plot, we use subplot.

```
df["label"].value_counts().plot(kind="bar",figsize=(12,6))
plt.xticks(np.arange(2), ('Non spam', 'spam'),rotation=0);
```



- In our dataset we have some categorical features. With the count plot function, we are going to count the unique category in those features.

Countplot:-

A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable. The basic API and options are identical to those for barplot() , so you can compare counts across nested variables.

From the graph we can infer that , more data belongs class 0 than class 1

Scaling the Data

Scaling is one the important process, we have to perform on the dataset, because of data measures in different ranges can leads to mislead in prediction

Models such as KNN, Logistic regression need scaled data, as they follow distance based method and Gradient Descent concept.

```
# perfroming feature Scaling op[eration using standard scaller on X part of the dataset becau
# there different type of values in the columns
sc=StandardScaler()
x_bal=sc.fit_transform(x_bal)

x_bal = pd.DataFrame(x_bal,columns=names)
```

We will perform scaling only on the input values.Once the dataset is scaled, it will be converted into an array and we need to convert it back to a dataframe.

Splitting data into train and test

Now let's split the Dataset into train and test sets

Changes: 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 the train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

```
#Splitting data into train and validation sets using train_test_split

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

##train size 80% and test size 20%
```

Milestone 4: Model Building

Activity 1: 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 four classification algorithms. The best model is saved based on its performance.

Activity 1.1: Decision tree model

A function named decisionTree is created and train and test data are passed as the parameters. Inside the function, 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. For evaluating the model, a confusion matrix and classification report is done.

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(X_train_res, y_train_res)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

Activity 1.2: Random forest model

A function named randomForest is created and train and test data are passed as the parameters. Inside the function, 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. For evaluating the model, a confusion matrix and classification report is done.

```
from sklearn.ensemble import RandomForestClassifier
model1 =RandomForestClassifier()
model1.fit(X_train_res, y_train_res)
```

```
RandomForestClassifier
RandomForestClassifier()
```

Activity 1.3: Naïve Bayes model

A function named MultinomialNB is created and train and test data are passed as the parameters. Inside the function, MultinomialNB 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.naive_bayes import MultinomialNB
   model = MultinomialNB()
```

```
]: #Fitting the model to the training sets
   model.fit(X_train_res, y_train_res)
```

```
]: ▾ MultinomialNB
   MultinomialNB()
```

Activity 1.5: ANN model

Building and training an Artificial Neural Network (ANN) using the Keras library with TensorFlow as the backend. The ANN is initialised as an instance of the Sequential class, which is a linear stack of layers. Then, the input layer and two hidden layers are added to the model using the Dense class, where the number of units and activation function are specified. The output layer is also added using the Dense class with a sigmoid activation function. The model is then compiled with the Adam optimizer, binary cross-entropy loss function, and accuracy metric. Finally, the model is fit to the training data with a batch size of 100, 20% validation split, and 100 epochs.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
#Fitting the model to the training sets
model = Sequential()
```

```
X_train.shape
```

```
(4457, 7163)
```

```
model.add(Dense(units  = X_train_res.shape[1],activation="relu",kernel_initializer="random_uniform"))
```

```
model.add(Dense(units=100,activation="relu",kernel_initializer="random_uniform"))
```

```
model.add(Dense(units=100,activation="relu",kernel_initializer="random_uniform"))
```

```
model.add(Dense(units=1,activation="sigmoid"))
```

```
model.compile(optimizer="adam",loss="binary_crossentropy",metrics=['accuracy'])
```

```
generator = model.fit(X_train_res,y_train_res,epochs=10,steps_per_epoch=len(X_train_res)//64)
```

```
generator = model.fit(X_train_res,y_train_res,epochs=10,steps_per_epoch=len(X_train_res)//64)
```

```
Epoch 1/10
121/121 [==============================] - 26s 203ms/step - loss: 0.1404 - accuracy: 0.9565
Epoch 2/10
121/121 [==============================] - 24s 199ms/step - loss: 0.0220 - accuracy: 0.9950
Epoch 3/10
121/121 [==============================] - 23s 194ms/step - loss: 0.0155 - accuracy: 0.9965
Epoch 4/10
121/121 [==============================] - 23s 194ms/step - loss: 0.0163 - accuracy: 0.9962
Epoch 5/10
121/121 [==============================] - 24s 195ms/step - loss: 0.0158 - accuracy: 0.9965
Epoch 6/10
121/121 [==============================] - 23s 194ms/step - loss: 0.0132 - accuracy: 0.9974
Epoch 7/10
121/121 [==============================] - 23s 194ms/step - loss: 0.0130 - accuracy: 0.9972
Epoch 8/10
121/121 [==============================] - 24s 196ms/step - loss: 0.0120 - accuracy: 0.9974
Epoch 9/10
121/121 [==============================] - 23s 193ms/step - loss: 0.0103 - accuracy: 0.9977
Epoch 10/10
111/121 [==========================>...] - ETA: 1s - loss: 0.0128 - accuracy: 0.9972WARNING:tensorflow:Your input ran out of da
ta; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches (i
n this case, 1210 batches). You may need to use the repeat() function when building your dataset.
121/121 [==============================] - 23s 193ms/step - loss: 0.0128 - accuracy: 0.9972
```

Activity 2: Testing the model

```
[45]: y_pred=model.predict(X_test)
      y_pred
```

```
35/35 [==============================] - 2s 29ms/step
```

```
[45]: array([[1.5844109e-15],
             [4.4117199e-04],
             [1.1517070e-18],
             ...,
             [2.0661259e-08],
             [3.8018154e-17],
             [1.2099350e-12]], dtype=float32)
```

```
[46]: y_pr = np.where(y_pred>0.5,1,0)
```

```
[49]: y_test
```

```
[49]: array([0, 0, 0, ..., 0, 0, 0])
```

```
[50]: from sklearn.metrics import confusion_matrix,accuracy_score
      cm = confusion_matrix(y_test, y_pr)
      score = accuracy_score(y_test,y_pr)
      print(cm)
      print('Accuracy Score Is:- ' ,score*100)
```

```
[[937  12]
 [ 16 150]]
Accuracy Score Is:-  97.48878923766816
```

In ANN we first have to save the model to the test the inputs

This code defines a function named "new_review" which takes in a new_review as an input. The function then converts the input new_review from a list to a numpy array. It reshapes the new_review array as it contains only one record. Then, it applies feature scaling to the reshaped new_review array using a scaler object 'sc' that should have

been previously defined and fitted. Finally, the function returns the prediction of the classifier on the scaled new_review

```
: def new_review(new_review):
    new_review = new_review
    new_review = re.sub('[^a-zA-Z]', ' ', new_review)
    new_review = new_review.lower()
    new_review = new_review.split()
    ps = PorterStemmer()
    all_stopwords = stopwords.words('english')
    all_stopwords.remove('not')
    new_review = [ps.stem(word) for word in new_review if not word in   set(all_stopwords)]
    new_review = ' '.join(new_review)
    new_corpus = [new_review]
    new_X_test = cv.transform(new_corpus).toarray()
    print(new_X_test)
    new_y_pred = loaded_model.predict(new_X_test)
    print(new_y_pred)
    new_X_pred = np.where(new_y_pred>0.5,1,0)
    return new_y_pred
new_review = new_review(str(input("Enter new review...")))
```

```
Enter new review...hello fg hou hkl
[[0 0 0 ... 0 0 0]]
1/1 [==============================] - 0s 45ms/step
[[0.9784973]]
```

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 four models, the compareModel function is defined.

```
from sklearn.metrics import confusion_matrix,accuracy_score, classification_report
cm = confusion_matrix(y_test, y_pred)
score = accuracy_score(y_test,y_pred)
print(cm)
print('Accuracy Score Is Naive Bayes:- ' ,score*100)
```

```
[[935  14]
 [ 13 153]]
Accuracy Score Is:-  97.57847533632287
```

```
cm = confusion_matrix(y_test, y_pred)
score = accuracy_score(y_test,y_pred)
print(cm)
print('Accuracy Score Is:- ' ,score*100)

cm1 = confusion_matrix(y_test, y_pred1)
score1 = accuracy_score(y_test,y_pred1)
print(cm1)
print('Accuracy Score Is:- ' ,score1*100)
```

```
[[796 153]
 [ 17 149]]
Accuracy Score Is:-  84.75336322869956
[[855  94]
 [ 14 152]]
Accuracy Score Is:-  90.31390134529148
```

```
121/121 [==============================] - 24s 196ms/step - loss: 0.0120 - accuracy: 0.9974
Epoch 9/10
121/121 [==============================] - 23s 193ms/step - loss: 0.0103 - accuracy: 0.9977
Epoch 10/10
111/121 [==========================>...] - ETA: 1s - loss: 0.0128 - accuracy: 0.9972WARNING:tens
```

```
from sklearn.metrics import confusion_matrix,accuracy_score
cm = confusion_matrix(y_test, y_pr)
score = accuracy_score(y_test,y_pr)
print(cm)
print('Accuracy Score Is:- ' ,score*100)
```

```
[[937  12]
 [ 16 150]]
Accuracy Score Is:-  97.48878923766816
```

After calling the function, the results of models are displayed as output. From the five models ANN is performing well. From the below image, We can see the accuracy of the model. ANN is giving the 99.72% accuracy for the training data and 97.48 for testing data

Activity 2:Comparing model accuracy before & after applying hyperparameter tuning

Evaluating performance of the model From sklearn, cross_val_score is used to evaluate the score of the model. On the parameters, we have given rf (model name), x, y, cv (as 5 folds). Our model is performing well. So, we are saving the model by model.save(“model.h5”)

Note: To understand cross validation, refer to this link

```
]: from sklearn.metrics import confusion_matrix,accuracy_score
   cm = confusion_matrix(y_test, y_pr)
   score = accuracy_score(y_test,y_pr)
   print(cm)
   print('Accuracy Score Is:- ' ,score*100)
```

```
[[937  12]
 [ 16 150]]
Accuracy Score Is:-  97.48878923766816
```

Milestone 6: Model Deployment

Activity 1:Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. 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.

Saving our model

By comparing the all the model , we can come to a conclusion that ANN is the best model

```
]: model.save('spam.h5')
```

Activity 2: 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 2.1: Building Html Pages:

For this project create two HTML files namely

- index.html

- spam.html
- result.html

and save them in the templates folder.

Activity 2.2: Build Python code:

Import the libraries

```
# Importing essential libraries
from flask import Flask, render_template, request
import pickle
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from tensorflow.keras.models import load_model
# Load the Multinomial Naive Bayes model and CountVector
```

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

```
# Load the Multinomial Naive Bayes model
loaded_model = load_model('spam.h5')
cv = pickle.load(open('cv1.pkl','rb'))
app = Flask(__name__)
```

Render HTML page:

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

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with the home.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:

```
@app.route('/Spam',methods=['POST','GET'])
def prediction(): # route which will take you to the prediction page
    return render_template('spam.html')

@app.route('/predict',methods=['POST'])
def predict():
    if request.method == 'POST':
        message = request.form['message']
        data = message

    new_review = str(data)
    print(new_review)
    new_review = re.sub('[^a-zA-Z]', ' ', new_review)
    new_review = new_review.lower()
    new_review = new_review.split()
    ps = PorterStemmer()
    all_stopwords = stopwords.words('english')
    all_stopwords.remove('not')
    new_review = [ps.stem(word) for word in new_review if not word in   set(all_stopwords)]
    new_review = ' '.join(new_review)
    new_corpus = [new_review]
    new_X_test = cv.transform(new_corpus).toarray()
    print(new_X_test)
    new_y_pred = loaded_model.predict(new_X_test)
    new_X_pred = np.where(new_y_pred>0.5,1,0)
    print(new_X_pred)
    if new_review[0][0]==1:
       return render_template('result.html', prediction="Spam")
    else :
       return render_template('result.html', prediction="Not a Spam")
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

```
if __name__=="__main__":

    # app.run(host='0.0.0.0', port=8000,debug=True)    # running the app
    port=int(os.environ.get('PORT',5000))
    app.run(debug=False)
```

Activity 2.3: Run the web application

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

- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.