Project Development Phase
Delivery of Sprint - 4

Date	04 November 2022
Team ID	PNT2022TMID01470
Project Name	Detect Parkinson's disease

Creating improvised model & Integrate With Flask Web Page

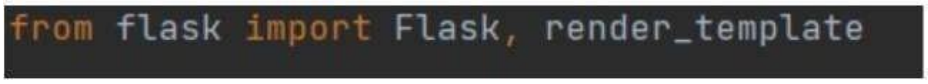
Let us build our flask application which will be running in our local browser as an user interface.

In the flask application, users will upload images and the result if they are healthy or diagnosed with Parkinson will be displayed.

Building Python Code

1: Importing Libraries

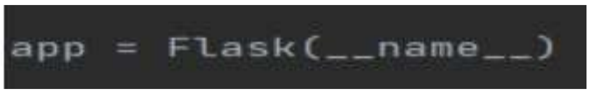
The first step is usually importing the libraries that will be needed in theprogram.



```
from flask import Flask, render_template
```

Importing the flask module into the project is mandatory. An object of theFlask class is our WSGI application. Flask constructor takes the name of thecurrent module (_name _).

2: Creating our flask application and loading

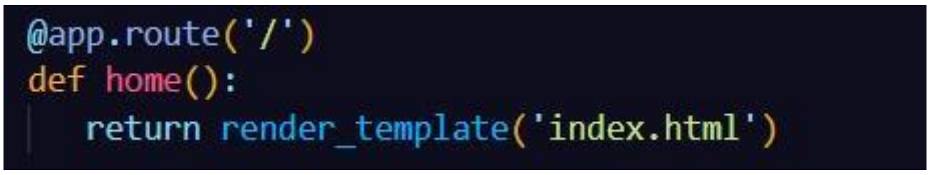


```
app = Flask(__name__)
```

3: Routing to the Html Page

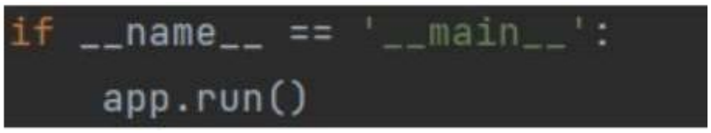
Here, the declared constructor is used to route to the HTML page createdearlier.

The '/' route is bound with the bot function. Hence, when the home page of a web server is opened in the browser, the HTML page will be rendered.



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

Main Function
This is used to run the application in local host.



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

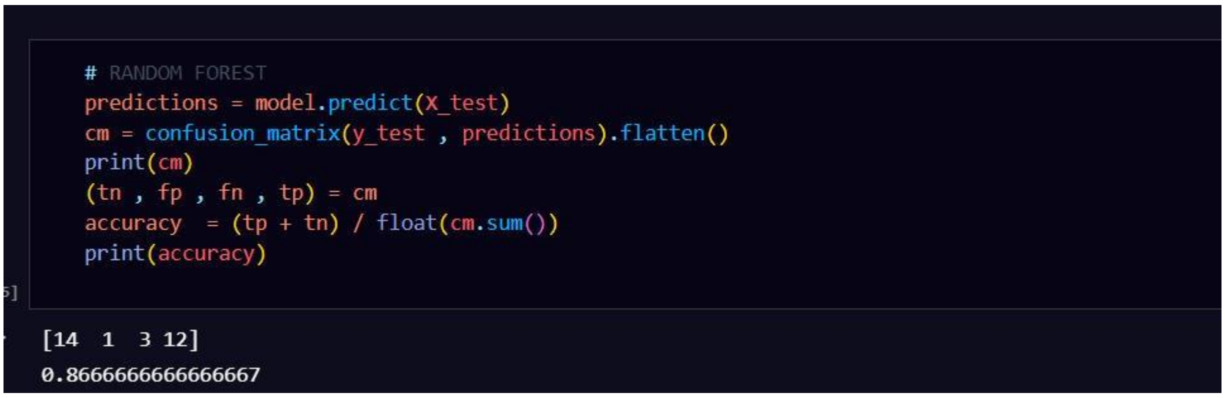
Building HTML Code

We have used HTML to create the front-end part of the web page.

Here, we have created "index.html" displays the home page which gets integrated with WatsonAssistant.

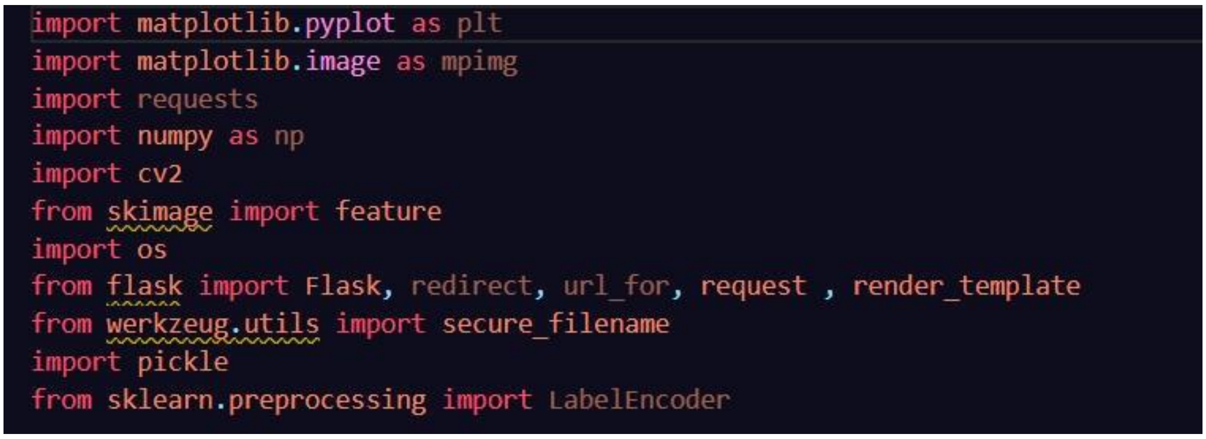
Run the application

Open Jupyter notebook (anaconda3)
Navigate to the folder where app.ipynb resides.
Run the python code
Open a browser and type this URL
It launches the web application integrated with pickle file of the generated model and IBM Watson Studio.

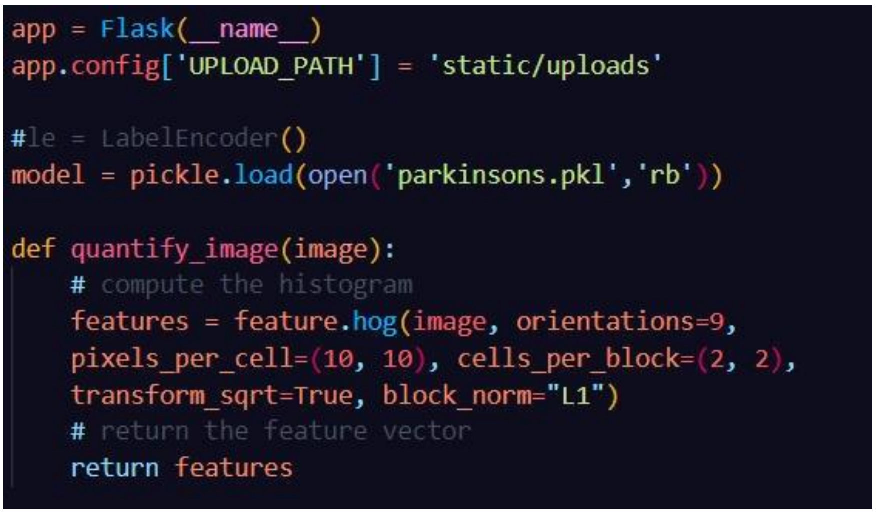


```
# RANDOM FOREST
predictions = model.predict(X_test)
cm = confusion_matrix(y_test , predictions).flatten()
print(cm)
(tn , fp , fn , tp) = cm
accuracy  = (tp + tn) / float(cm.sum())
print(accuracy)
```

```
[14  1  3 12]
0.8666666666666667
```



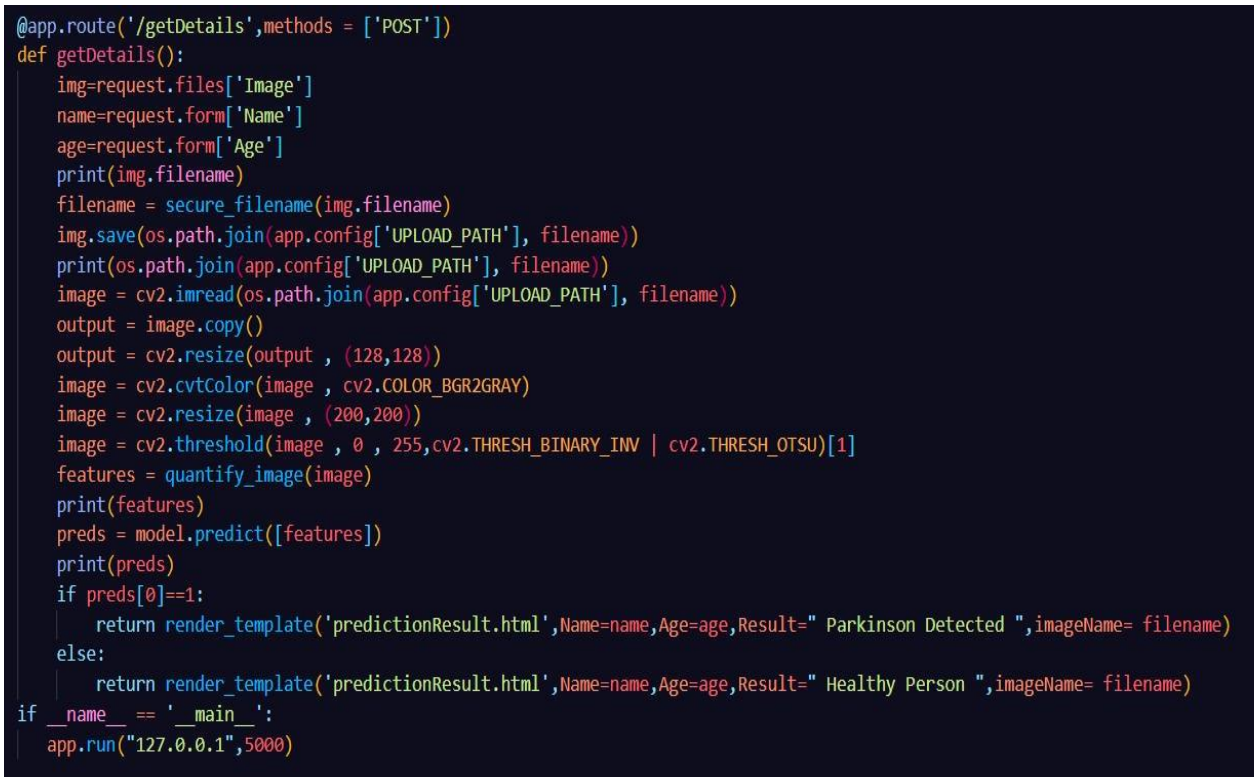
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import requests
import numpy as np
import cv2
from skimage import feature
import os
from flask import Flask, redirect, url_for, request , render_template
from werkzeug.utils import secure_filename
import pickle
from sklearn.preprocessing import LabelEncoder
```



```
app = Flask(__name__)
app.config['UPLOAD_PATH'] = 'static/uploads'

#le = LabelEncoder()
model = pickle.load(open('parkinsons.pkl','rb'))

def quantify_image(image):
    # compute the histogram
    features = feature.hog(image, orientations=9,
    pixels_per_cell=(10, 10), cells_per_block=(2, 2),
    transform_sqrt=True, block_norm="L1")
    # return the feature vector
    return features
```



```python
@app.route('/getDetails',methods = ['POST'])
def getDetails():
    img=request.files['Image']
    name=request.form['Name']
    age=request.form['Age']
    print(img.filename)
    filename = secure_filename(img.filename)
    img.save(os.path.join(app.config['UPLOAD_PATH'], filename))
    print(os.path.join(app.config['UPLOAD_PATH'], filename))
    image = cv2.imread(os.path.join(app.config['UPLOAD_PATH'], filename))
    output = image.copy()
    output = cv2.resize(output , (128,128))
    image = cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image , (200,200))
    image = cv2.threshold(image , 0 , 255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    features = quantify_image(image)
    print(features)
    preds = model.predict([features])
    print(preds)
    if preds[0]==1:
        return render_template('predictionResult.html',Name=name,Age=age,Result=" Parkinson Detected ",imageName= filename)
    else:
        return render_template('predictionResult.html',Name=name,Age=age,Result=" Healthy Person ",imageName= filename)
if __name__ == '__main__':
    app.run("127.0.0.1",5000)
```