Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Name : Jhanvi Parekh DIV : D11 SAP id : 60009210033

Subject: Applied Data Science (DJ19DSL703)

Experiment -2

(Project Deployment)

Aim: Project Deployment using Flask

Theory:

Flask is a micro web framework written in Python. It is designed to be lightweight and easy to use, making it a popular choice for building web applications. Flask provides a simple and flexible way to handle web requests and responses, manage routes, and work with templates. It follows the WSGI (Web Server Gateway Interface) standard, allowing it to run on various web servers. Flask also supports extensions that provide additional functionality, such as database integration, authentication, etc.

Creating a Simple Flask Application:
To create a simple Flask application, you need to follow these steps:

1. Install and Import Flask: Begin by installing Flask using pip, the Python package installer. Open your terminal or command prompt and run the following command:
   pip install flask
   from flask import Flask

2. Create an instance of the Flask application:
   app = Flask(__name__)
   The __name__ is a special Python variable that represents the name of the current module.

3. Define a route and view function:
   @app.route('/')
   def hello(): return "Hello, Flask!"
   This code creates a route that maps to the root URL ("/") of your application and defines a view function that returns the message "Hello, Flask!".

4. Run the application:
   if __name__== '__main__': app.run()
   This code ensures that the application is only run if the script is executed directly, not imported as a module.

5. Launch the application: In your terminal or command prompt, navigate to the directory where your script is located and run the following command:
   python your_script_name.py

This will start the Flask development server, and you can access your application by visiting **http://localhost:5000** in your web browser.

Flask Routes and Views:
Routes in Flask define the URL patterns that the application will respond to. Each route is associated with a view function that handles the request and returns a response.

- Route decorators: Use the @app.route() decorator to define a route. You can specify the URL pattern as an argument.
- HTTP methods: Routes can be associated with specific HTTP methods such as GET, POST, etc. Use the methods parameter in the decorator to specify the allowed methods.
- Dynamic routes: Flask supports dynamic routes where parts of the URL can be variables. You can specify dynamic segments using angle brackets (<variable>).
- View functions: Each route should have a corresponding view function that handles the request and generates a response. The function should return the response data.

Flask Templates:
Flask uses the Jinja2 templating engine to render HTML templates. Templates allow separating the presentation logic from the application logic. Following are some of the Flask templates:

- Template rendering: Use the render_template() function to render a template. It takes the template file name as an argument and can accept additional data to be passed to the template.
- Template inheritance: Jinja2 supports template inheritance, allowing you to create a base template with common elements and extend it in child templates with additional content.
- Template control structures: Jinja2 provides control structures like loops and conditionals, which allow you to dynamically generate content in your templates.

Deploying Flask Applications:
Following are the general steps of the deployment process:

1. Choose a hosting platform: Select a hosting platform that supports Flask applications. Popular options include Heroku, AWS, Google Cloud Platform, Postman and PythonAnywhere.
2. Set up the deployment environment: Follow the instructions provided by the hosting platform to set up the deployment environment. This usually involves creating an account, configuring the server, and installing any necessary dependencies.
3. Prepare your application: Ensure that the Flask application is ready for deployment. This includes making sure all the necessary dependencies are listed in a requirements.txt file, and any configuration settings are correctly set.
4. Deploy your application: Use the deployment tools or commands provided by the hosting platform to deploy the Flask application. This typically involves pushing your code to a Git repository, configuring the server, and starting the application.

5. Test and monitor: After deployment, thoroughly test your application to ensure it's functioning as expected. Set up monitoring and error tracking to receive notifications of any issues that arise.

Lab Assignment:

1. Implement the basic structure of flask using the concept of request, rendering, templates (Jinja2), and methods (GET and POST).

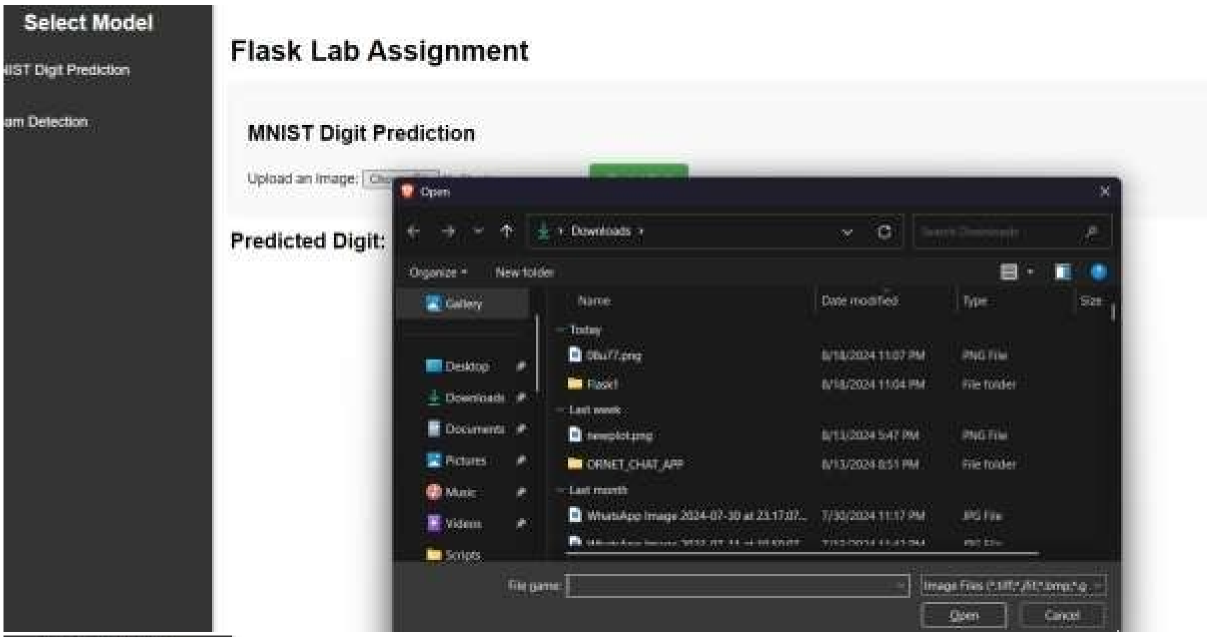
2. Implement RestAPI using flask and Postman on a static form using an appropriate GUI.

Dataset: mnist.csv

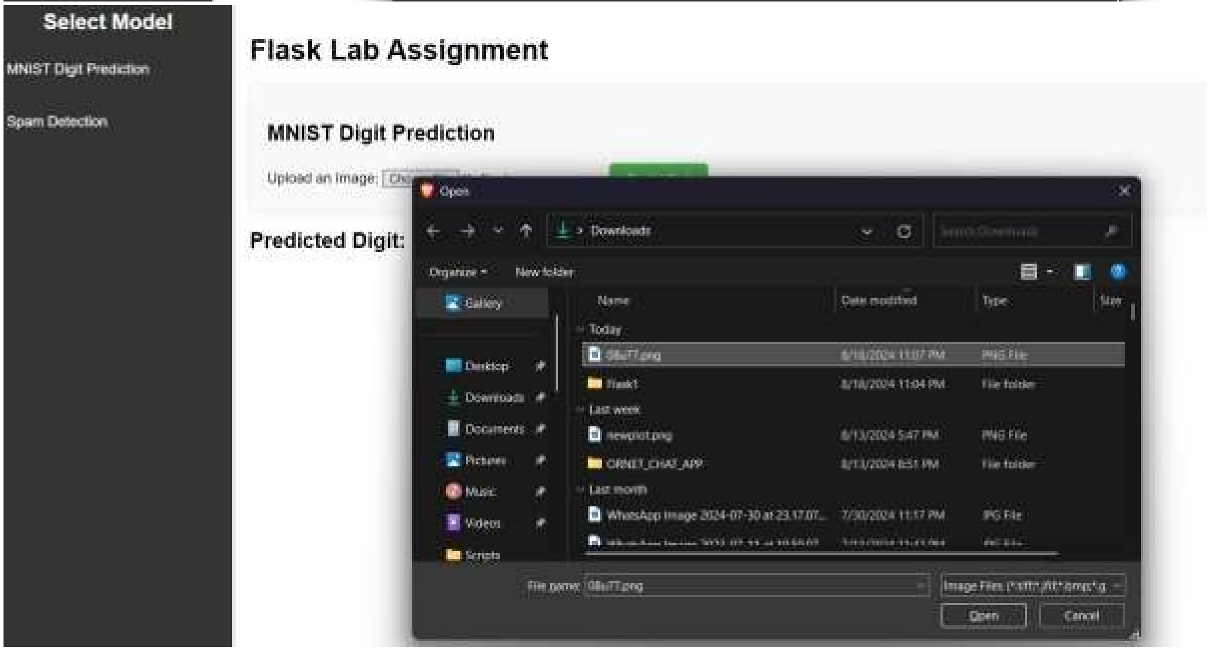
1. Implement a deep learning model on MNIST dataset at the backend to predict a digit and render it on the frontend using appropriate Flask methods.

Dataset: Spam.csv

1. Using the concept of natural language processing implement a model at the backend to predict whether a text is spam or not and render it on the frontend using appropriate GUI and Flask methods.



Name	Date modified	Type
Today		
08u77.png	8/18/2024 11:07 PM	PNG File
Flask1	8/18/2024 11:04 PM	File folder
Last week		
newplot.png	8/13/2024 5:47 PM	PNG File
ORNET_CHAT_APP	8/13/2024 8:51 PM	File folder
Last month		
WhatsApp Image 2024-07-30 at 23.17.07...	7/30/2024 11:17 PM	JPG File



Name	Date modified	Type
Today		
08u77.png	8/18/2024 11:07 PM	PNG File
Flask1	8/18/2024 11:04 PM	File folder
Last week		
newplot.png	8/13/2024 5:47 PM	PNG File
ORNET_CHAT_APP	8/13/2024 8:51 PM	File folder
Last month		
WhatsApp Image 2024-07-30 at 23.17.07...	7/30/2024 11:17 PM	JPG File

File name: 08u77.png

App.py

```
from flask import Flask, render_template, request
from tensorflow.keras.models import load_model
import numpy as np
from PIL import Image, ImageOps
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
```

```
import string
import joblib

# Download necessary NLTK data
nltk.download("stopwords")
nltk.download('punkt')

# Initialize Flask app
app = Flask(_name_)

# Load the trained models
mnist_model = load_model('mnist_model.h5')
spam_model = joblib.load('spam_model.pkl')
vectorizer = joblib.load('vectorizer.pkl')

# Text preprocessing function
ps = PorterStemmer()

def transform_text(text):
    text = text.lower()
    text = nltk.word_tokenize(text)

    y = []
    for i in text:
        if i.isalnum():
            y.append(i)

    text = y[:]
    y.clear()

    for i in text:
        if i not in stopwords.words('english') and i not in string.punctuation:
            y.append(i)

    text = y[:]
    y.clear()

    for i in text:
        y.append(ps.stem(i))

    return " ".join(y)

# Home route
```

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

# MNIST Prediction Route
@app.route('/predict-digit', methods=['POST'])
def predict_digit():
    file = request.files['image']
    img = Image.open(file.stream).convert('L')
    img = ImageOps.invert(img)
    img = img.resize((28, 28))
    img = np.array(img)
    img = img.reshape(1, 28, 28, 1)
    img = img / 255.0
    prediction = mnist_model.predict(img)
    digit = np.argmax(prediction)
    return render_template('index.html', prediction=digit, model='mnist')

# Spam Detection Route
@app.route('/predict-spam', methods=['POST'])
def predict_spam():
    text = request.form['text']

    # Preprocess the input text
    transformed_text = transform_text(text)

    # Transform the text using the loaded vectorizer
    text_vector = vectorizer.transform([transformed_text])

    # Predict using the spam model
    prediction = spam_model.predict(text_vector)
    result = 'Spam' if prediction[0] == 'spam' else 'Not Spam'

    return render_template('index.html', prediction=result, model='spam')

if__name__== '_main_':
    app.run(debug=True)
```

INDEX.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flask Lab Assignment</title>
  <link rel="stylesheet" href="../static/styles.css">
</head>

<body>
  <div class="sidebar">
    <h2>Select Model</h2>
    <ul>
          <li><a href="#" onclick="showForm('mnist')">MNIST Digit
Prediction</a></li>
      <li><a href="#" onclick="showForm('spam')">Spam Detection</a></li>
    </ul>
  </div>

  <div class="main-content">
    <h1>Flask Lab Assignment</h1>

    <!-- MNIST Form -->
    <div id="mnist-form" class="form-container">
      <h2>MNIST Digit Prediction</h2>
     <form action="/predict-digit" method="POST" enctype="multipart/form-data">
        <label for="image">Upload an Image:</label>
        <input type="file" id="image" name="image" accept="image/*" required>
        <button type="submit">Predict Digit</button>
      </form>
    </div>
```

```
        <!-- Spam Detection Form -->
        <div id="spam-form" class="form-container" style="display:none;">
            <h2>Spam Detection</h2>
            <form action="/predict-spam" method="POST">
                <label for="text">Enter Text:</label>
                <input type="text" id="text" name="text" required>
                <button type="submit">Check Spam</button>
            </form>
        </div>

        <!-- Prediction Result -->
        {% if prediction is not none %}
            <div class="result-container">
                {% if model == 'mnist' %}
                    <h2>Predicted Digit: {{ prediction }}</h2>
                {% elif model == 'spam' %}
                    <h2>Text is: {{ prediction }}</h2>
                {% endif %}
            </div>
        {% endif %}
    </div>

    <script>
        function showForm(formId) {
            document.getElementById('mnist-form').style.display = 'none';
            document.getElementById('spam-form').style.display = 'none';
            document.getElementById(formId + '-form').style.display = 'block';
        }
    </script>
</body>

</html>
```