ITTE105b – Analytics Application
Model Deployment – Estimation of Obesity Levels Based on Eating Habits and Physical Condition

Objectives

The objective of this activity is to deploy the best-performing machine learning model that predicts obesity levels based on individual health and lifestyle factors, and implement it as a web service using FastAPI and Docker.

I. Preparation of the Jupyter Notebook for Model Deployment

The notebook file [obesity levels] was used to train the model, with Support Vector Machines (SVM) identified as the best-performing model based on accuracy and other classification metrics. Before deploying the trained model, all necessary components, including the model itself as well as the preprocessing tools, such as the encoder and scaler, were serialized using the **joblib** library, which is designed for saving and loading Python objects.

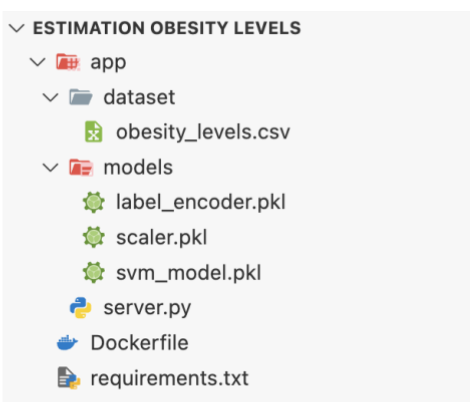
```
...
import joblib;
...

svm = SVC(C=1000, gamma=0.001, probability=True)
svm.fit(X_train, y_train)

joblib.dump(svm, 'svm_model.pkl')
joblib.dump(scaler, 'scaler.pkl')
joblib.dump(encoder, 'label_encoder.pkl')
```

II. Model Deployment using Fast API and Docker

After developing the model trained using SVM and serializing the preprocessing components, the next step is to deploy it as a web service using FastAPI and Docker. FastAPI is a Python framework used for building web APIs, while Docker is a platform that enables containerization of applications to ensure consistent deployment across different environments.

A. Folder Structure

The project folder [obesity_levels_pred] follows the structure shown below.



The **app directory** acts the core component which contains the application logic, dataset and serialized model files necessary for the operation of Fast API service.

The **DockerFile** defines the instructions for building the Docker Image of the Fast API application. It specifies the base environment, dependencies, working directory, and startup commands.

```
FROM python:3.11

WORKDIR /code

COPY ./requirements.txt /code/requirements.txt

RUN pip install --no-cache-dir -r /code/requirements.txt

COPY ./app /code/app

EXPOSE 8000

CMD ["uvicorn", "app.server:app", "--host", "0.0.0.0", "--port", "8000"]
```

The **requirement.txt** file includes all Python dependencies required by the application, including **FastAPI**, **Scikit-learn**, **NumPy**, **Uvicorn** and **Pandas**.

```
scikit-learn==1.3.1
fastapi
numpy
uvicorn
pandas
```

The **app/dataset** folder contains the dataset used for training and validating the model. Meanwhile, the **app/models** folder includes the trained SVM model for predicting obesity levels, along with the serialized preprocessing tools generated from the **jupyter notebook file**. These tools include the **StandardScaler** object, which normalizes feature inputs during preprocessing, and the **LabelEncoder** object which converts categorical variables into numeric form.

The **server.py** implements the FastAPI web service. It loads the serialized model and preprocessing tools, defines API endpoints, and handles HTTP requests for prediction. The API receives input data, applies scaling and encoding, and returns predicted obesity levels and confidence scores in JSON format.

```
...
def predict(data: dict):
...
    # Scale the data from the dataframe
    new_data_scaled = scaler.transform(new_data)
    # Predict the obesity levels and probabilities
    prediction = svm.predict(new_data_scaled)
    probabilities = svm.predict_proba(new_data_scaled)
    # Get individual confidence scores for each predictions
    results = []
    for i in range(len(prediction)):
        pred_class = encoder.inverse_transform([prediction[i]])[0]
        confidence = float(probabilities[i][prediction[i]])
        results.append({
            "predicted_class": pred_class,
            "confidence": round(confidence, 2)
        })

    return {"results": results}
```

B. Build the Docker Image

To build the Docker image using Visual Studio Code (VSCode), first open your project folder in the editor. Once the folder is open, launch the terminal within VSCode and run the following **docker build** command. The image is labeled as **obesity_levels_pred_img**. Once the command is executed successfully, Docker processes each instruction from the **Dockerfile**, installing dependencies, copying the FastAPI application files, and setting up the environment.

```
1. docker build -t obesity_levels_pred_img .
```

After successfully building the Docker image, the next step is to run the container to make the FastAPI service accessible.

```
1. docker run --name obesity_levels_pred -p 8000:8000 obesity_levels_pred_img
```

When executed, Docker launches the containerized FastAPI application and exposes the API endpoints defined in the **server.py** file. If the application runs successfully without errors, the corresponding container will appear in the Docker dashboard, indicating that the deployment is active.

		Name	Container ID	Image	Port(s)	Actions
☐	●	obesity_levels_pred	3bc9e8164422	obesity_levels_pred	8000:8000	

The root endpoint can be accessed at **localhost:8000**, which can also be used to verify whether the API is running.

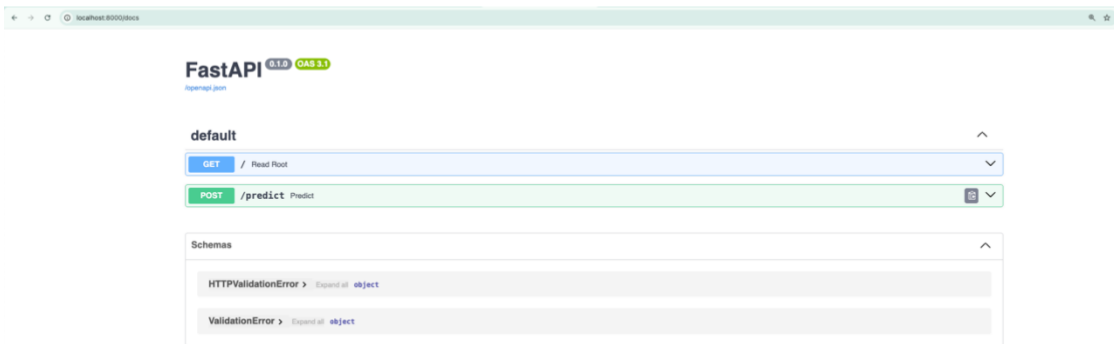
C. Access the Fast API Document

Once the Docker container is successfully running, the deployed model can be accessed through **Swagger UI or Redoc**, both of which are automatically generated interactive documentation provided by FastAPI. These interfaces allow users to explore, interact with, and test the API endpoints directly from a web browser.

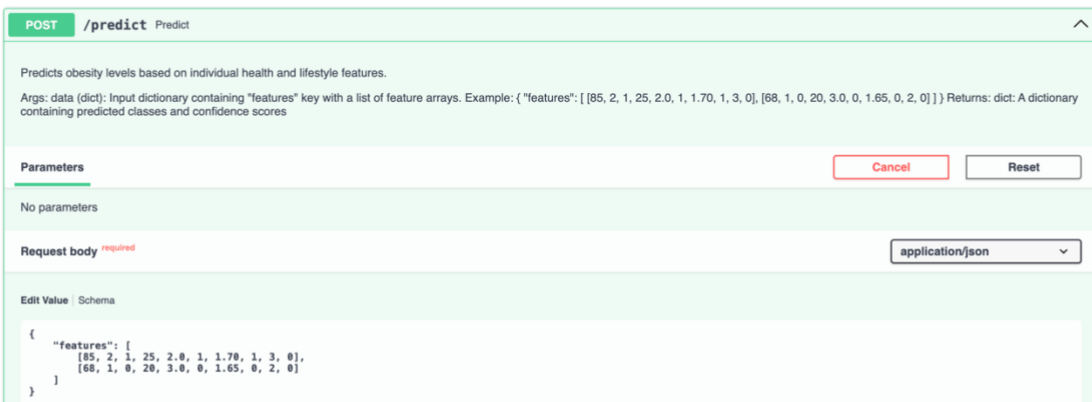
D. Steps in accessing the interactive documentations:

1. Open a web browser and go to http://localhost:8000/docs. This opens the **Swagger UI**, an interactive documentation tool that displays all available endpoints of the API. The API contains the following end points.
    - The root endpoint / provides a simple connection test message.

- The /predict endpoint allows users to send input data and receive prediction results.



2. After accessing the FastAPI documentation, you can test the deployed model's predictions by providing sample input data in the required format through the **/predict** endpoint.



The sample input data follow the order of the input features defined in the developed model, as described in the following list of input feature descriptions.

Features	Description	Example Value
Weight (kg)	The individual's body weight in kilograms.	70
CAEC (Consumption of Food Between Meals)	Encoded frequency of eating between meals: 0 = No, 1 = Sometimes, 2 = Frequently, 3 = Always.	2
family_history_with_overweight	Indicates if the individual has family members with overweight or obesity: 0 = No, 1 = Yes.	1
Age	The individual's age in years.	25
CH2O (Water Intake per Day)	Average daily water consumption: 1 = Less than 1L, 2 = 1–2L, 3 = More than 2L.	2.0
Height (m)	The individual's height in meters.	1.70

Gender	Gender of the individual: 0 = Female, 1 = Male.	1
FCVC (Frequency of Vegetable Consumption)	Frequency of eating vegetables: 1 = Never, 2 = Sometimes, 3 = Always.	3
SMOKE	Indicates whether the individual smokes: 0 = No, 1 = Yes.	0

After entering the sample input data, click the "Execute" button to review the response section below. This section displays the predicted output along with the confidence score for each returned prediction.

III. Integrate with Client Application

After verifying that the FastAPI model endpoint functions correctly, the next step is to integrate it with a web or mobile application interface which allows users to allows users to interact with the model through an intuitive front-end form instead of directly using Swagger UI. A web or mobile application framework can be used to send requests to the web service and process the responses and display the model's predictions to the user in a user-friendly manner.