Exp No. 3	Containerize ML Model for Deployment	REG. NO: URK23CS1197
Date:		

Objective:

To create a Docker container for a machine learning model, enabling easy deployment, scalability, and reproducibility across different environments.

Tools/ Software Required:

- Docker Desktop or Docker Engine
- Python 3.8 or above
- Code Editor (e.g., VS Code)
- Internet connection for image pull and push (Docker Hub)
- Command-line interface (Terminal or PowerShell)

Job Role:

- DevOps Engineer
- Machine Learning Engineer

Skills Required:

- Containerization with Docker and understanding container life cycle
- Machine Learning frameworks such as TensorFlow, PyTorch, or Scikit-learn
- Proficiency in Python programming
- Familiarity with CI/CD tools (GitHub Actions, Jenkins)
- Knowledge of container orchestration tools like Kubernetes

Prerequisites:

- Understanding of the Software Development Life Cycle (SDLC) and Agile methodologies
- Basic understanding of ML model deployment concepts
- Installed Docker and Python environment
- Ability to write and document Dockerfiles and deployment steps
- Familiarity with Linux command-line operations

Description :

A container packages an application and all its dependencies so it can run uniformly across environments. Docker helps create, deploy, and manage these containers efficiently. Containerizing a machine learning model ensures consistency, scalability, and quick deployment — essential for cloud and production setups.When using Docker:

- The Dockerfile defines the environment and dependencies.
- The image built from the Dockerfile can be reused and shared.
- The container is the running instance of the image.

This lab demonstrates how to containerize a simple ML application.

Questions:

1.Demonstrate the process of creating a Dockerfile and building a Docker image to execute a machine learning algorithm, including the installation of required dependencies and packages.

Program:

```
from flask import Flask, request, jsonify, render_template
import joblib
import numpy as np
from flask_cors import CORS

app = Flask(__name__)
CORS(app)
model = joblib.load("diabetes_model.pkl")

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

@app.route('/predict', methods=['POST'])
def predict():
    try:
        data = request.get_json(force=True)
        features = [data[feature] for feature in [
            "Pregnancies", "Glucose", "BloodPressure", "SkinThickness",
            "Insulin", "BMI", "DiabetesPedigreeFunction", "Age"
        ]]
        input_array = np.array(features).reshape(1, -1)
        prediction = model.predict(input_array)[0]
        result = "Diabetic" if prediction == 1 else "Non-Diabetic"
        return jsonify({'prediction': result})
    except KeyError as e:
        return jsonify({'error': f'Missing feature: {e}'}), 400
    except Exception as ex:
        return jsonify({'error': str(ex)}), 500

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

```
FROM python:3.8-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir --upgrade pip && \
    pip install --no-cache-dir -r requirements.txt
```

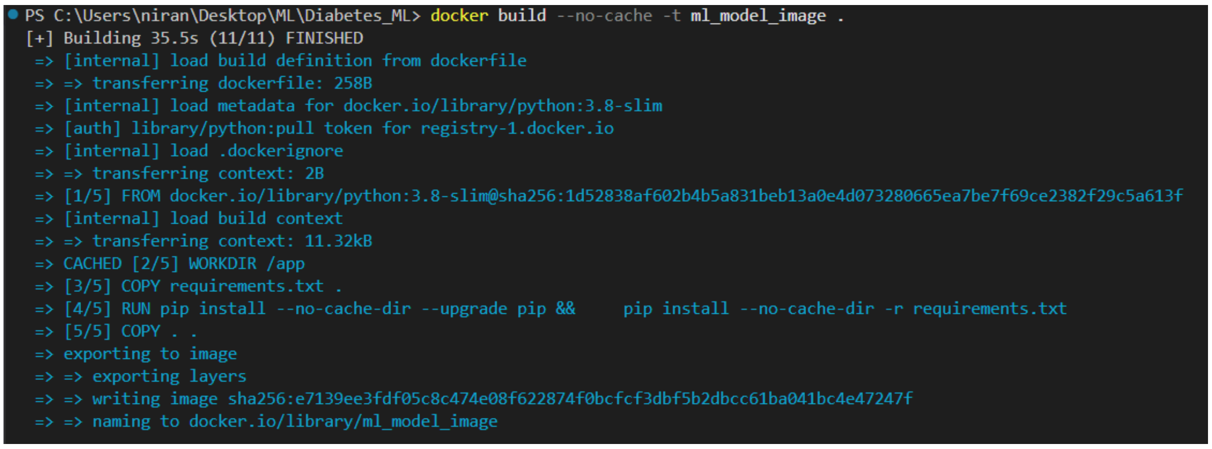
```
COPY . .
EXPOSE 80
CMD ["python", "app.py"]
```

```
docker build --no-cache -t ml_model_image .
docker run -p 4000:80 --name ml_model_container_v2 ml_model_image
```

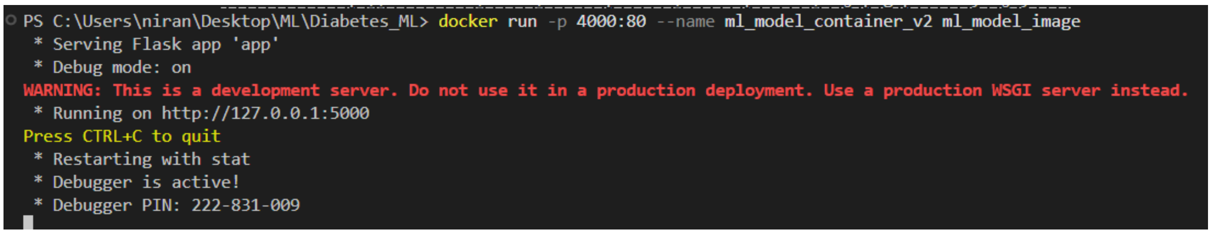
2.Analyze a provided Dockerfile used for executing a machine learning algorithm and identify any potential issues or improvements to optimize performance and resource utilization.

- The Dockerfile efficiently installs dependencies by copying requirements.txt first and upgrading pip before installing packages, which optimizes Docker layer caching and speeds up rebuilds when source code changes do not affect dependencies.
- To further improve performance, copying only requirements.txt before installing and then copying the rest of the source code reduces unnecessary reinstallation of packages when code changes occur, thus improving build times.
- Including a .dockerignore file to exclude unnecessary files such as local caches, Git metadata, or temporary files can significantly reduce the Docker build context size, leading to faster builds and smaller image sizes.
- It is important to pin dependency versions in the requirements.txt file to ensure reproducibility and avoid unexpected version conflicts or breaks caused by package updates.
- Adding a Docker HEALTHCHECK directive in the Dockerfile can facilitate automatic monitoring of container health, enabling orchestrators or container management platforms to manage container restarts or alerts, enhancing reliability during production deployments.

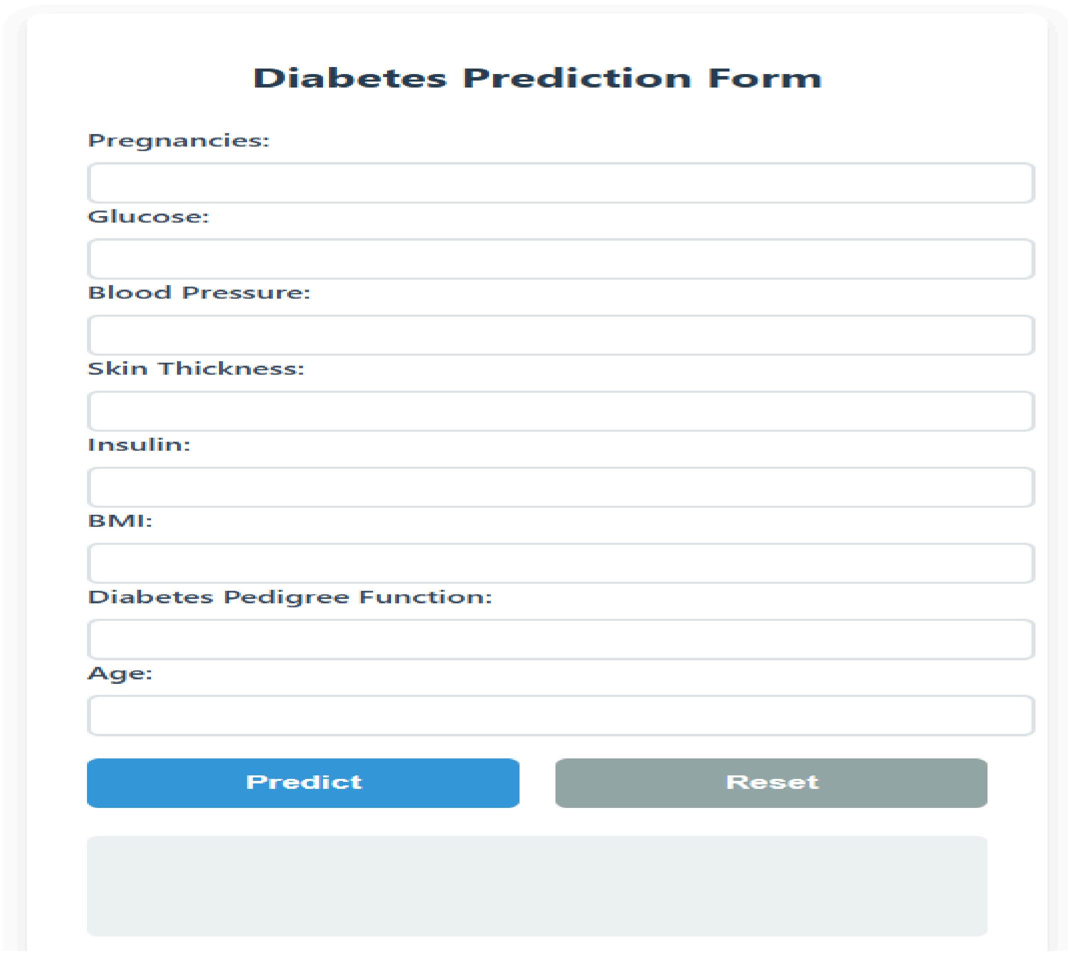
Expected Output :



```
PS C:\Users\niran\Desktop\ML\Diabetes_ML> docker build --no-cache -t ml_model_image .
[+] Building 35.5s (11/11) FINISHED
 => [internal] load build definition from dockerfile
 => => transferring dockerfile: 258B
 => [internal] load metadata for docker.io/library/python:3.8-slim
 => [auth] library/python:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/5] FROM docker.io/library/python:3.8-slim@sha256:1d52838af602b4b5a831beb13a0e4d073280665ea7be7f69ce2382f29c5a613f
 => [internal] load build context
 => => transferring context: 11.32kB
 => CACHED [2/5] WORKDIR /app
 => [3/5] COPY requirements.txt .
 => [4/5] RUN pip install --no-cache-dir --upgrade pip &&     pip install --no-cache-dir -r requirements.txt
 => [5/5] COPY . .
 => exporting to image
 => => exporting layers
 => => writing image sha256:e7139ee3fdf05c8c474e08f622874f0bcfcf3dbf5b2dbcc61ba041bc4e47247f
 => => naming to docker.io/library/ml_model_image
```



```
PS C:\Users\niran\Desktop\ML\Diabetes_ML> docker run -p 4000:80 --name ml_model_container_v2 ml_model_image
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 222-831-009
```


Diabetes Prediction Form

Pregnancies:

Glucose:

Blood Pressure:

Skin Thickness:

Insulin:

BMI:

Diabetes Pedigree Function:

Age:

Result :

The experiment involved creating a Docker container for a machine learning model by writing a Dockerfile based on Python 3.8, copying the app and model files, and installing required packages like numpy, scikit-learn, and flask from a requirements.txt file. Issues with missing numpy components and dependencies were fixed by upgrading pip and explicitly listing modules. The built image runs the model successfully, providing a portable and reproducible deployment environment accessible via mapped ports, demonstrating effective ML model containerization best practices.