Hotel Booking Prediction Docker Exercise

Scenario: Predicting Hotel Booking Cancellations

As a data scientist at Agoda, you have been tasked with developing a machine learning model to predict hotel booking cancellations. Your manager asked you to containerize your entire workflow using Docker to ensure reproducibility and easy deployment.

Exercise Overview

1. Set up a Docker environment for data analysis and model training
2. Load and preprocess hotel booking data
3. Train a machine learning model to predict booking cancellations
4. Create a simple frontend to interact with the model
5. Use Docker Compose to orchestrate multiple services

Why Use Docker for This Project?

Docker is used in this project to ensure consistency and reproducibility across different environments. It allows us to package our application, its dependencies, and its runtime environment into containers. This means that the project will run the same way on any machine with Docker installed, eliminating the “it works on my laptop” problem and making it easier to deploy and share the project.

File Explanations

- **Dockerfile:** This file contains instructions to build a Docker image for our application. It specifies the base image, sets up the working directory, and copies our code into the container.
- **docker-compose.yml:** This file defines and configures the services that make up our application. It allows us to run multi-container Docker applications easily.
- requirements.txt: This file lists all the Python packages our project depends on. Docker will use this to install the necessary libraries in the container.
- preprocess.py: This script handles data preprocessing tasks such as handling missing values and encoding categorical variables.
- train_model.py: This script trains the machine learning model using the preprocessed data and saves the trained model.
- app.py: This is the main application file that creates a web server to serve our predictions and analysis.
- index.html: This is a template for the webpage that displays our model's analysis results.

Prerequisites and Installation

Before starting the project, ensure you have the following installed on your system:

1. **Docker**: Download and install Docker from the official Docker website.
2. **Docker Compose**: This usually comes installed with Docker Desktop for Windows and Mac. For Linux, you might need to install it separately.
3. **Git**: If you want to clone the project repository.

To verify the installations:

```
docker --version
docker-compose —-version
```

Both commands should return version numbers if installed correctly.

Docker Cheat Sheet

Here's a quick reference for common Docker commands:

```
# Build and run your Docker containers
docker-compose up --build

# Stop and remove containers
docker-compose down

# List running containers
docker ps

# List all containers (including stopped ones)
docker ps -a

# Remove all stopped containers
docker container prune

# List Docker images
docker images

# Remove an image
docker rmi [image_name]

# View logs of a container
docker logs [container_name]

# Enter a running container
docker exec -it [container_name] /bin/bash

# Stop a running container
docker stop [container_name]
```

```
# Start a stopped container
docker start [container_name]
```

Step 1: Project Structure

- Create a folder “inn-hotels-prediction”, which will be your root directory for this project.
- Create the following project structure:

```
inn-hotels-prediction/
│
├── data/
│   └── Hotel.csv
├── src/
│   ├── preprocess.py
│   ├── train_model.py
│   └── app.py
├── templates/
│   └── index.html
├── Dockerfile
├── docker-compose.yml
└── requirements.txt
```

Step 2: Dockerfile

Create a `Dockerfile` in the root directory:

```
FROM python:3.8-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY src/ .
COPY data/ ./data/
COPY templates/ ./templates/

# Create a directory for the models
RUN mkdir -p /app/models

CMD ["python", "app.py"]
```

Step 3: requirements.txt

Create a requirements.txt file with the following content:

```
pandas
scikit-learn
flask
gunicorn
matplotlib
```

Step 4: Python Scripts

preprocess.py

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

def preprocess_data():
    df = pd.read_csv('data/Hotel.csv')

    # Separate numeric and categorical columns
    numeric_columns =
df.select_dtypes(include=[np.number]).columns
    categorical_columns =
df.select_dtypes(exclude=[np.number]).columns

    # Handle missing values for numeric columns
    numeric_imputer = SimpleImputer(strategy='mean')
    df[numeric_columns] =
numeric_imputer.fit_transform(df[numeric_columns])

    # Handle missing values for categorical columns
    categorical_imputer = SimpleImputer(strategy='most_frequent')
    df[categorical_columns] =
categorical_imputer.fit_transform(df[categorical_columns])

    # Encode categorical variables
    df_encoded = pd.get_dummies(df, columns=[col for col in
categorical_columns if col not in ['ID', 'status']])

    # Split features and target
    X = df_encoded.drop(['ID', 'status'], axis=1)
    y = df_encoded['status'].map({'Canceled': 1, 'Not_Canceled':
0})

    # Scale features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    return X_scaled, y, scaler, X.columns

if __name__ == "__main__":
    preprocess_data()
```

train_model.py

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
import joblib
from preprocess import preprocess_data
import matplotlib.pyplot as plt
import numpy as np
import os

def train_model():
    X, y, scaler, feature_names = preprocess_data()

    # Train a Logistic Regression model (simpler and faster than
Random Forest)
    model = LogisticRegression(random_state=42, max_iter=1000)
    model.fit(X, y)

    # Make predictions
    y_pred = model.predict(X)

    # Print model performance
    print("Model Accuracy:", accuracy_score(y, y_pred))
    print("\nClassification Report:")
    print(classification_report(y, y_pred))

    # Feature importance plot (for Logistic Regression, we'll use
the absolute values of coefficients)
    importances = np.abs(model.coef_[0])
    indices = np.argsort(importances)[::-1]

    plt.figure(figsize=(12,8))
    plt.title("Feature Importances")
    plt.bar(range(len(importances)), importances[indices])
    plt.xticks(range(len(importances)), [feature_names[i] for i in
indices], rotation=90)
    plt.tight_layout()

    # Ensure the templates directory exists
    os.makedirs('templates', exist_ok=True)
    plt.savefig('templates/feature_importance.png')

    # Ensure the models directory exists
    os.makedirs('models', exist_ok=True)

    # Save the model and scaler
    joblib.dump(model, 'models/lr_model.joblib')
    joblib.dump(scaler, 'models/scaler.joblib')

if __name__ == "__main__":
    train_model()
```

app.py

```
from flask import Flask, render_template
import pandas as pd
import joblib
import os
import time
import numpy as np

app = Flask(__name__)

def load_model(retries=5, delay=10):
    for _ in range(retries):
        if os.path.exists('models/lr_model.joblib') and
os.path.exists('models/scaler.joblib'):
            model = joblib.load('models/lr_model.joblib')
            scaler = joblib.load('models/scaler.joblib')
            return model, scaler
        print("Waiting for model and scaler files...")
        time.sleep(delay)
    raise FileNotFoundError("Model or scaler file not found after
multiple retries")

def load_data():
    return pd.read_csv('data/Hotel.csv')

model, scaler = None, None
df = load_data()

def analyze_model():
    global model, scaler, df

    if model is None or scaler is None:
        try:
            model, scaler = load_model()
        except FileNotFoundError:
            return "Model is still training. Please try again
later."

    # Get feature names (excluding 'ID' and 'status')
    feature_names = [col for col in df.columns if col not in
['ID', 'status']]

    # Get model coefficients
    coefficients = model.coef_[0]

    # Pair feature names with their coefficients
    feature_importance = list(zip(feature_names, coefficients))

    # Sort by absolute value of coefficient (descending)
    feature_importance.sort(key=lambda x: abs(x[1]), reverse=True)
```

```
    # Prepare the analysis results
    analysis = []
    analysis.append("Top 5 factors that predict cancellation:")
    for feature, coef in feature_importance[:5]:
        impact = "increases" if coef > 0 else "decreases"
        analysis.append(f"- {feature}: {impact} likelihood of
cancellation")

    analysis.append("\nRecommendations to avoid cancellations:")
    for feature, coef in feature_importance[:5]:
        if coef > 0:
            analysis.append(f"- Reduce {feature} if possible")
        else:
            analysis.append(f"- Increase {feature} if possible")

    return "\n".join(analysis)

@app.route('/')
def index():
    analysis_result = analyze_model()
    return render_template('index.html', analysis=analysis_result)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Step 5: HTML Template

Create `index.html` in the `templates` directory:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Hotel Booking Cancellation Analysis</title>
    <style>
        body { font-family: Arial, sans-serif; line-height: 1.6;
padding: 20px; max-width: 800px; margin: 0 auto; }
        h1 { color: #333; }
        pre { background-color: #f4f4f4; padding: 15px; border-
radius: 5px; white-space: pre-wrap; word-wrap: break-word; }
    </style>
</head>
<body>
    <h1>Hotel Booking Cancellation Analysis</h1>
    <pre>{{ analysis }}</pre>
</body>
</html>
```

Step 6: Docker Compose

Create a `docker-compose.yml` file in the root directory:

```
services:
  data_preprocessing:
    build: .
    command: python preprocess.py
    volumes:
      - ./data:/app/data
      - ./models:/app/models

  model_training:
    build: .
    command: python train_model.py
    volumes:
      - ./data:/app/data
      - ./models:/app/models
      - ./templates:/app/templates
    depends_on:
      - data_preprocessing

  web_app:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - ./models:/app/models
      - ./templates:/app/templates
    depends_on:
      - model_training
    command: >
      sh -c "python app.py"
```

Step 7: Running

- Place the Hotel.csv file in the data/ directory.
- Open a terminal or command prompt in the project root directory.
- Build and run the Docker containers:

```
docker-compose up --build
```

This command will:

  - Build the Docker image based on your Dockerfile
  - Start the containers defined in your docker-compose.yml
  - Run the data preprocessing, model training, and start the web application
- Now, you can access the web application in your browser http://localhost:5000.

Troubleshooting

If you encounter any issues:

1. Ensure all files are in the correct directories as specified in the project structure.
2. Check that the Hotel.csv file is present in the data/ directory.
3. Verify that Docker is running on your system.
4. If changes to your code are not reflected, try rebuilding the images:

   ```
   docker-compose build
   ```

   Then run the containers again.
5. Check the logs of your containers for any error messages:

   ```
   docker-compose logs
   ```

Remember, Docker isolates your application in containers, so any changes to files on your host system won't automatically appear inside the container unless you rebuild the image or use volume mounts (as we do for some directories in this project).