**[Opening Scene]**

**[Narrator]**: "Welcome to our comprehensive tutorial on training, tuning, and deploying a machine learning model! In this video, we'll guide you through each step of the process, from hyperparameter tuning to deploying your model using Flask and Docker. Let's dive in!"

**[Scene 1: Introduction to Hyperparameter Tuning]**

**[Narrator]**: "First, let's talk about hyperparameter tuning. Hyperparameter tuning is crucial for optimizing the performance of your machine learning model. It involves finding the best set of parameters that maximize the model's accuracy."

**[On-Screen Text]**: "Hyperparameter Tuning with Optuna and GridSearchCV"

**[Narrator]**: "We'll be using two powerful tools for this: Optuna and GridSearchCV. Let's start with Optuna."

**[Scene 2: Hyperparameter Tuning with Optuna]**

**[Narrator]**: "Optuna is an automatic hyperparameter optimization framework designed for machine learning. It uses a technique called Bayesian optimization to find the best parameters efficiently."

**[On-Screen Text]**: "Step 1: Install Optuna"

**[Narrator]**: "First, we need to install Optuna. Open your terminal and run the following command."

**[On-Screen Code]**:

pip install optuna

**[Narrator]**: "Next, we define an objective function. This function trains the model and returns its accuracy."

**[On-Screen Code]**:

import optuna

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

def objective(trial):

data = load\_iris()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target, test\_size=0.2)

n\_estimators = trial.suggest\_int('n\_estimators', 10, 100)

max\_depth = trial.suggest\_int('max\_depth', 2, 32)

clf = RandomForestClassifier(n\_estimators=n\_estimators, max\_depth=max\_depth)

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

return accuracy\_score(y\_test, y\_pred)

**[Narrator]**: "We then create a study and optimize it to find the best hyperparameters."

**[On-Screen Code]**:

study = optuna.create\_study(direction='maximize')

study.optimize(objective, n\_trials=100)

**[Narrator]**: "Finally, we train the model with the best parameters and save it using joblib."

**[On-Screen Code]**:

best\_model\_optuna = RandomForestClassifier(\*\*study.best\_params)

best\_model\_optuna.fit(X\_train, y\_train)

joblib.dump(best\_model\_optuna, 'best\_model\_optuna.pkl')

**[Scene 3: Hyperparameter Tuning with GridSearchCV]**

**[Narrator]**: "Next, let's use GridSearchCV. GridSearchCV performs an exhaustive search over a specified parameter grid to find the best combination of hyperparameters."

**[On-Screen Text]**: "Step 1: Define Parameter Grid"

**[Narrator]**: "We start by defining the parameter grid. This grid specifies the hyperparameters and their possible values."

**[On-Screen Code]**:

param\_grid = {'n\_estimators': [10, 50, 100], 'max\_depth': [None, 10, 20, 30]}

**[Narrator]**: "Next, we perform the grid search using GridSearchCV."

**[On-Screen Code]**:

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

import joblib

data = load\_iris()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target, test\_size=0.2)

clf = RandomForestClassifier()

grid\_search = GridSearchCV(estimator=clf, param\_grid=param\_grid, cv=5, scoring='accuracy')

grid\_search.fit(X\_train, y\_train)

**[Narrator]**: "Finally, we save the best model found by GridSearchCV."

**[On-Screen Code]**:

best\_model\_gridsearch = grid\_search.best\_estimator\_

joblib.dump(best\_model\_gridsearch, 'best\_model\_gridsearch.pkl')

**[Scene 4: Packaging the Model with Flask and Docker]**

**[Narrator]**: "Now that we have our best models, let's package them using Flask and Docker. Flask will help us create a web API to serve our model, and Docker will containerize our application for easy deployment."

**[On-Screen Text]**: "Step 1: Create Flask Application"

**[Narrator]**: "First, we create a Flask application. Install Flask by running the following command."

**[On-Screen Code]**:

pip install flask

**[Narrator]**: "Next, create a file named app.py and add the following code to define the API endpoint for predictions."

**[On-Screen Code]**:

from flask import Flask, request, jsonify

import joblib

app = Flask(\_\_name\_\_)

model = joblib.load('best\_model.pkl')

@app.route('/predict', methods=['POST'])

def predict():

data = request.get\_json(force=True)

prediction = model.predict([data['features']])

return jsonify({'prediction': prediction.tolist()})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

**[Narrator]**: "This code sets up a Flask app with an endpoint /predict that takes a JSON payload and returns the model's prediction."

**[On-Screen Text]**: "Step 2: Create Dockerfile"

**[Narrator]**: "Next, we create a Dockerfile to containerize our application. Create a file named Dockerfile and add the following content."

**[On-Screen Code]**:

FROM python:3.8-slim

WORKDIR /app

COPY requirements.txt requirements.txt

RUN pip install -r requirements.txt

COPY . .

EXPOSE 5000

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

**[Narrator]**: "This Dockerfile sets up the environment, installs dependencies, and runs the Flask app."

**[On-Screen Text]**: "Step 3: Create requirements.txt"

**[Narrator]**: "Create a file named requirements.txt and list all the dependencies your application needs."

**[On-Screen Code]**:

flask

joblib

scikit-learn

**[Scene 5: Building and Running the Docker Container]**

**[Narrator]**: "To build and run the Docker container, navigate to your project directory and run the following commands."

**[On-Screen Code]**:

docker build -t my-flask-app .

docker run -p 5000:5000 my-flask-app

**[Narrator]**: "These commands build the Docker image and run the container, making the Flask app accessible on port 5000."

**[Scene 6: Testing the Flask Application]**

**[Narrator]**: "Finally, let's test our Flask application to ensure it's working correctly."

**[On-Screen Text]**: "Testing with curl"

**[Narrator]**: "You can use curl to send a POST request to the /predict endpoint. Open your terminal and run the following command."

**[On-Screen Code]**:

curl -X POST http://localhost:5000/predict -H "Content-Type: application/json" -d '{"features": [5.1, 3.5, 1.4, 0.2]}'

**[Narrator]**: "Or use Postman to send a POST request with the JSON payload."

**[On-Screen Text]**: "Automated Testing with unittest"

**[Narrator]**: "You can also write automated tests using unittest. Create a file named test\_app.py and add the following code."

**[On-Screen Code]**:

import unittest

import json

from app import app

class FlaskTestCase(unittest.TestCase):

def setUp(self):

self.app = app.test\_client()

self.app.testing = True

def test\_predict(self):

response = self.app.post('/predict',

data=json.dumps({"features": [5.1, 3.5, 1.4, 0.2]}),

content\_type='application/json')

data = json.loads(response.get\_data(as\_text=True))

self.assertEqual(response.status\_code, 200)

self.assertIn('prediction', data)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**[Narrator]**: "This script tests the /predict endpoint to ensure it returns the expected response."

**[Closing Scene]**

**[Narrator]**: "And that's it! You've successfully trained, tuned, packaged, and tested a machine learning model