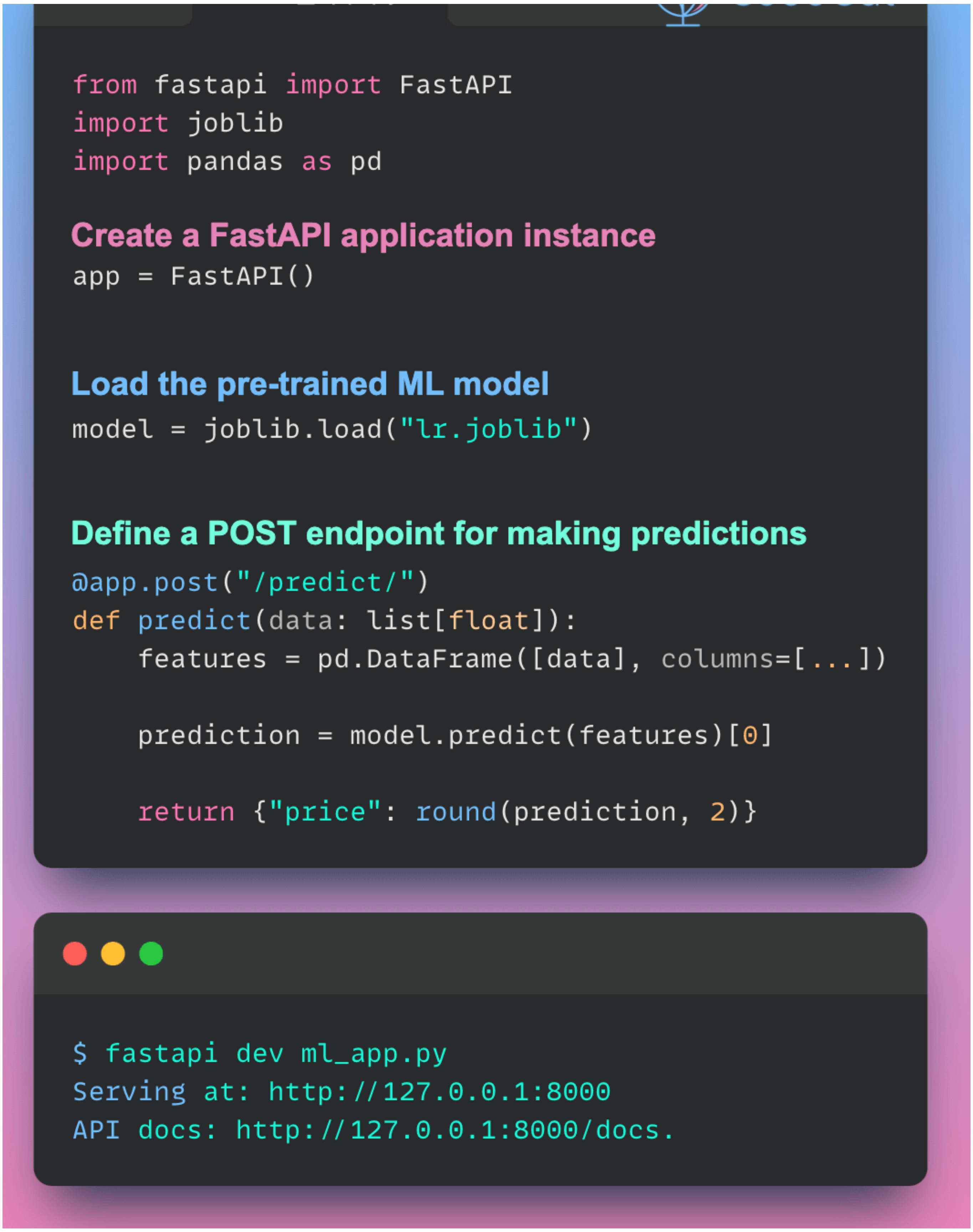

```python
from fastapi import FastAPI
import joblib
import pandas as pd
```

Create a FastAPI application instance

```python
app = FastAPI()
```

Load the pre-trained ML model

```python
model = joblib.load("lr.joblib")
```

Define a POST endpoint for making predictions

```python
@app.post("/predict/")
def predict(data: list[float]):
    features = pd.DataFrame([data], columns=[...])

    prediction = model.predict(features)[0]

    return {"price": round(prediction, 2)}
```

```
$ fastapi dev ml_app.py
Serving at: http://127.0.0.1:8000
API docs: http://127.0.0.1:8000/docs.
```

Simplifying ML Model Integration with FastAPI

Motivation

Imagine this scenario: You have just built an ML model with great performance, and you want to share this model with your team members so that they can develop a web application on top of your model.

One way to share the model with your team members is to save the model to a file (e.g., using pickle, joblib, or framework-specific methods) and share the file directly

Open in Colab

```
import joblib

model = ...

# Save model
joblib.dump(model, "model.joblib")

# Load model
model = joblib.load(model)
```

However, this approach requires the same environment and dependencies, and it can pose potential security risks.

An alternative is creating an API for your ML model. APIs define how software components interact, allowing:

1. Access from various programming languages and platforms

2. Easier integration for developers unfamiliar with ML or Python
3. Versatile use across different applications (web, mobile, etc.)

This approach simplifies model sharing and usage, making it more accessible for diverse development needs.

Create an ML API with FastAPI

Let's learn how to create an ML API with FastAPI, a modern and fast web framework for building APIs with Python.

Before we begin constructing an API for a machine learning model, let's first develop a basic model that our API will use. In this example, we'll create a model that predicts the median house price in California.

```python
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import joblib

# Load dataset
X, y = fetch_california_housing(as_frame=True, return_X_y=True)

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Initialize and train the logistic regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict and evaluate the model
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean squared error: {mse:.2f}")

# Save model
joblib.dump(model, "lr.joblib")
```

Once we have our model, we can create an API for it using FastAPI. We'll define a POST endpoint for making predictions and use the model to make predictions.

Here's an example of how to create an API for a machine learning model using FastAPI:

```
%%writefile ml_app.py
from fastapi import FastAPI
import joblib
import pandas as pd

# Create a FastAPI application instance
app = FastAPI()

# Load the pre-trained machine learning model
model = joblib.load("lr.joblib")

# Define a POST endpoint for making predictions
@app.post("/predict/")
def predict(data: list[float]):
    # Define the column names for the input features
    columns = [
        "MedInc",
        "HouseAge",
        "AveRooms",
        "AveBedrms",
        "Population",
        "AveOccup",
        "Latitude",
        "Longitude",
    ]

    # Create a pandas DataFrame from the input data
    features = pd.DataFrame([data], columns=columns)

    # Use the model to make a prediction
    prediction = model.predict(features)[0]

    # Return the prediction as a JSON object, rounding to 2 decimal places
    return {"price": round(prediction, 2)}
```

To run your FastAPI app for development, use the `fastapi dev` command:

```
$ fastapi dev ml_app.py
```

This will start the development server and open the API documentation in your default browser.

You can now use the API to make predictions by sending a POST request to the `/predict/` endpoint with the input data. For example:

Running this cURL command on your terminal:

```
curl -X 'POST' \
  'http://127.0.0.1:8000/predict/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '[
  1.68, 25, 4, 2, 1400, 3, 36.06, -119.01
]'
```

This will return the predicted price as a JSON object, rounded to 2 decimal places:

```
{"price":1.51}
```

Favorite ☆

Leave a Comment

Your email address will not be published. Required fields are marked *