Table of Contents:

Introduction

- In this lecture, we're going to cover these many things:
    1. Building API for Loan Approval ML Model
    2. Export the trained model
    3. Develop APIs to serve the models' inferences.
- Up untill now, we have worked with Jupyter Machines on the local machines. We'll use similar notebook where we've trained a model to predict the loan approval.
- The notebook can be found here
- In notebook, we've saved the model using **`pickle`**.

- Now, let's imagine what a loan prediction app would look like.
- First of all, it will ask users to enter some details, such as salary, age, gender, etc., and at the bottom there would be some **`submit`** button, which when pressed would predict the results.
- This is the fron-end part of the app; which could be a web application; or an app running on android or iOS.
- When the **`submit`** button is pressed,what happens is somehow predicted answers is shown; this is the backend part; and this is where we jump in!
- In the backend, when a particular user clicks the submit button, there would be a **`url request`** generated for accessing the model; present in the backend.
- Now, in the backend, there will be many different types of code written to handle multiple requests of multiple tasks.
- So, using that code, our model will send a response back to the front end part; predicting the loan status.

Q.How would the backend know which response to send given that there are multiple url request coming to it?

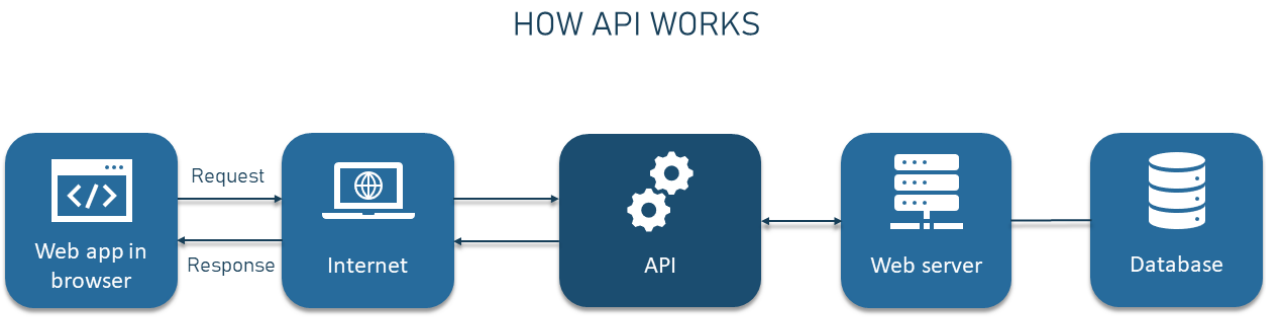
- The backend would be deployed somewhere on the cloud.
- There will be multiple code written for handling different types of request; and those each code is known as API endpoints.
- Each API endpoints will handle a particular type of request and send back the response

APIs

- API stands for **`Application Programming Interface`**
- Getting back to WWW (World Wide Web), you must have known that it sends the request and response from the client to server and vise versa.
- Those requests and responses must have some guidelines, and which is where HTTP comes into action.
- Now this comminucation does not always happen successfully. FOr that there are some **HTTP error codes**, which are as follows:
  - **`404`** - Page Not Found
  - **`200 series`** - Successful communcation
  - **`500 series`** - Internal Server Error

So, what is API, and where is it used?

- API works as an interface between the client and piece of code that is written on the servers; that would communicate with the requests and response by the clients



Types of Requests

- There can be multiple types of request that can be made from the client side.
- **`GET`** is used for fetching the data.

    - For example when you refresh your Instagram page for new content, you are fetching new posts from their servers
- **`POST`** is used for uploading/store the data on the server
    - For example, whenever you comment on a post, or like a post, that would be a POST request
- Another type of requests are: **`PUT`** and **`DELETE`**.

Let's now get started with Flask!

Flask

- Flask is a web framework written in Python
- To install flask, use the command: **`pip install flask`**

```
from flask import Flask
import pickle

app = Flask(__name__)
```

- Let us also collect the pre-trained classifier model, and for using it in our app, we'll use `pickle` library.
- We've imported `Flask` class, to which we'll pass the name of the app.
- We can now develop API endpoints as per our need.

Building API endpoints

- For a particular request we need to send it via some `url`, and corresponding to that `url`, we need some piece of code, that will respond when invoked.
- For this, we've written a function `ping()`, in which we will write a simple message written
- To define the `url`, we'll use a **decorator** `@app.route` to which you'll pass the url. You can also specify the type of url request such as **get**, **post**, etc.
- A decorator in python allows a use to add new functionality to an existing object without modifyign its structure.

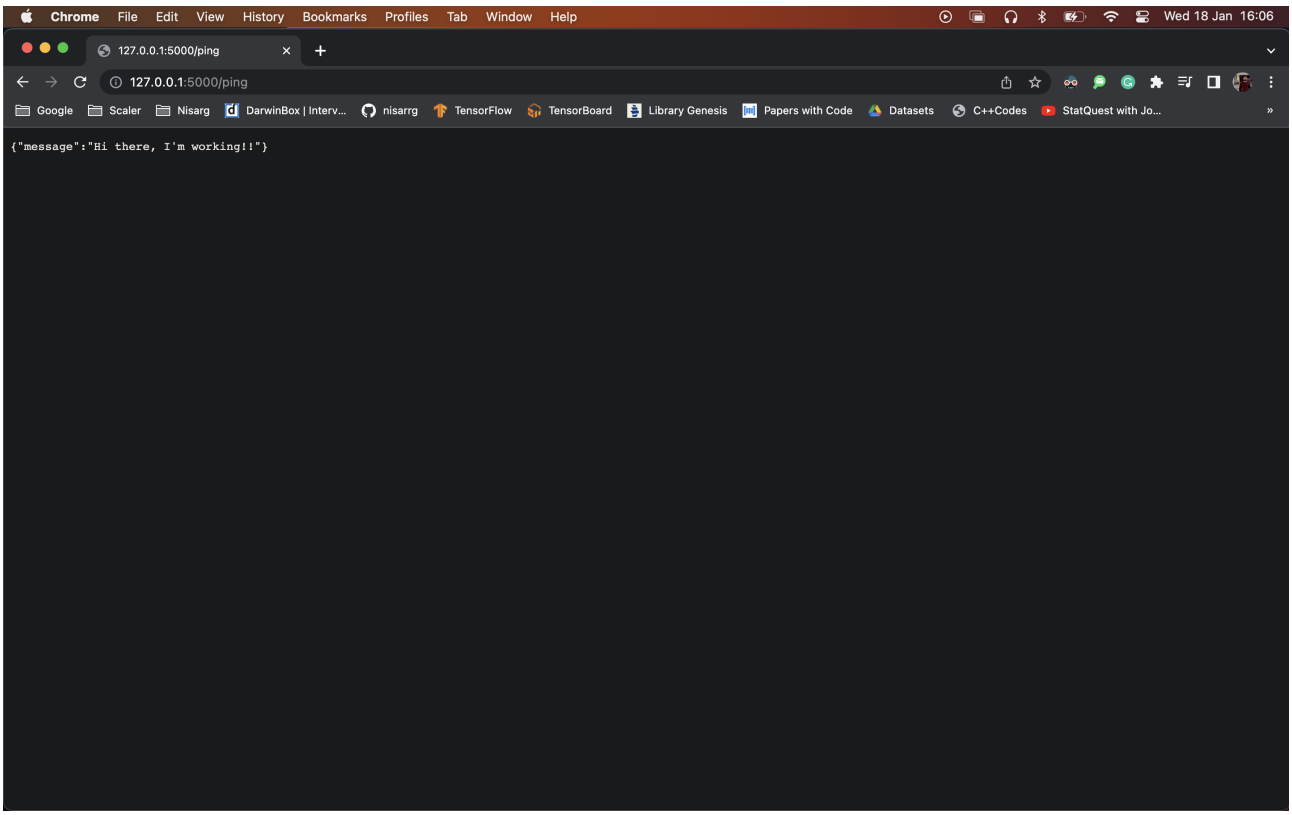
```
from flask import Flask
import pickle
```

```
app = Flask(__name__)

@app.route("/ping", methods=['GET'])
def ping():
    return {"message": "Hi there, I'm working!!"}
```

- To run this program; use the following command in the terminal of your IDE:

```
flask run
```

- Flask will give you an addreess. COpy that and paste it on your browser. After that add `/ping` to the address, because till now, we've only defined the backend for `ping` endpoint. The browser will show something like this:



- What flask does here is that it starts a Web Server Gateway Interface (WSGI)
- Let's now try to define an endpoint for using the pre-defined classifier

Prediction Endpoint

- For loan tap predictions, what we'll be asking from users is to fill up a form mentioning about their **salary, gender, marital status, credit history and loan amount**.
- We'll first laod the classifier model using pickle.
- We'll then create a new endpoint using a decorator with path as `/predict`. For this endpoint we'll use a `POST` request, becasue we'll be sending some information to the

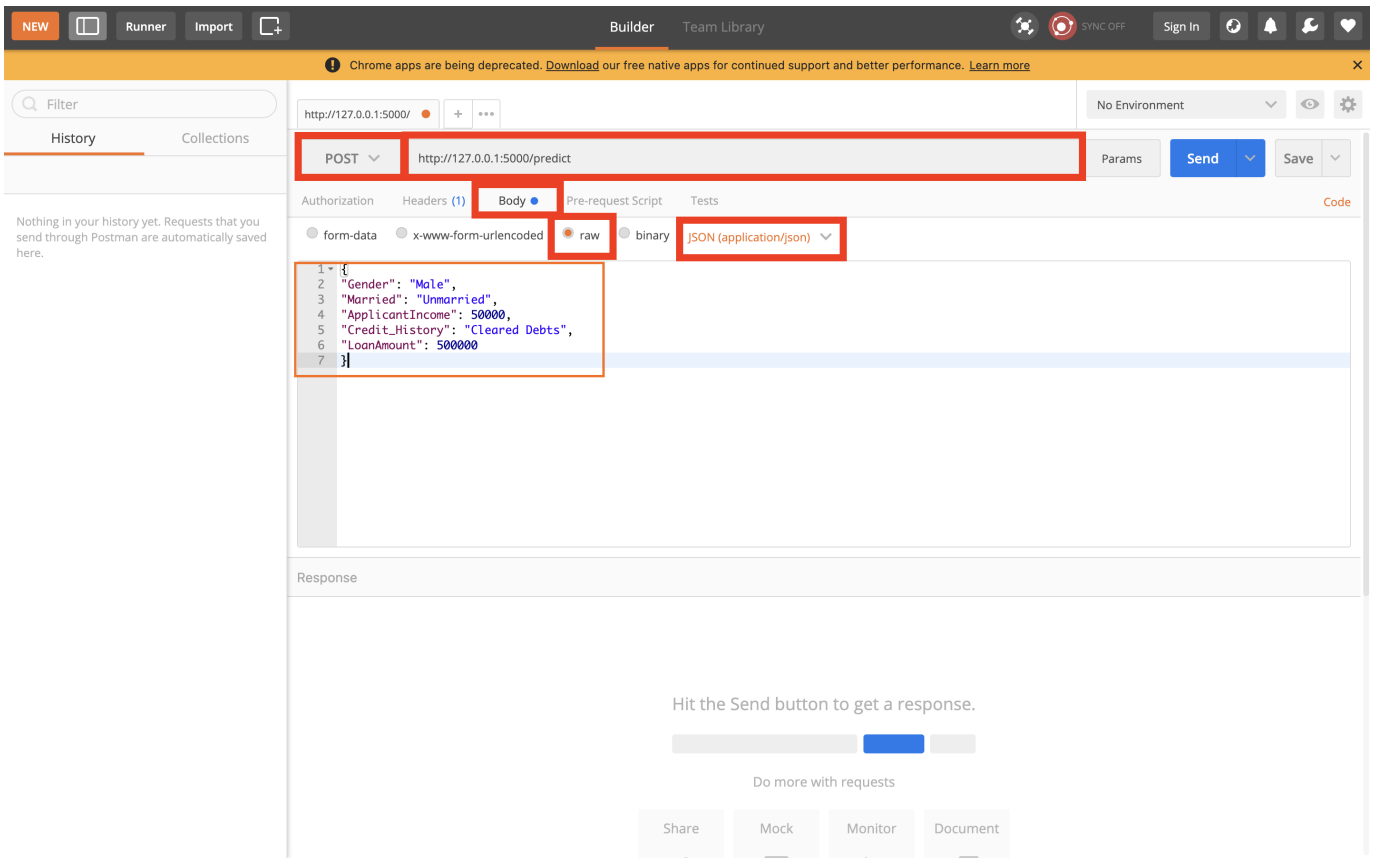
model for prediction.

- So, whenever a client is going to fill up the details and send a request through the url, we'll be sending the data in **JSON** format.
- JSON stands for Javascript Object Notation, which is similar to Python Dictionaries. All this is handled by Software engineers.
- This will invoke your `predcition` endpoint, and we'll have to decide how to read data from that JSON file.
- For that we'll import another class of flask : **`request`**

```
from flask import Flask, request
import pickle

app = Flask(__name__)


model_pickle = open("./artefacts/classifier.pkl", "rb")
clf = pickle.load(model_pickle)

@app.route("/ping", methods=['GET'])
def ping():
    return {"message": "Hi there, I'm working!!"}


#defining the endpoint which will make the prediction
@app.route("/predict", methods=['POST'])
def prediction():
    """ Returns loan application status using ML model
    """
    loan_req = request.get_json()
    print(loan_req)
    if loan_req['Gender'] == "Male":
        Gender = 0
    else:
        Gender = 1
    if loan_req['Married'] == "Unmarried":
        Married = 0
    else:
        Married = 1
    if loan_req['Credit_History'] == "Unclear Debts":
        Credit_History = 0
    else:
        Credit_History = 1

    ApplicantIncome = loan_req['ApplicantIncome']
    LoanAmount = loan_req['LoanAmount']

    result = clf.predict([[Gender, Married, ApplicantIncome, LoanAmount, Credit_His

    if result == 0:
        pred = "Rejected"
```

```
    else:
        pred = "Approved"

    return {"loan_approval_status": pred}
```

- You can sse in the aboove code that in the `prediction()` function we collected the data, and we're returning the prediction.
- Now, for our lecture, we don not have any websites or html forms to test this, and in the industry too, you're not supposed to do that. That is the part of the front-end engineers.
- So, to test our work, we'll use a tool called Postman. You need to install its extension on to your chrome browser using this link here
- After you've done that, you need to follow multiple steps shown below:
    1. select request type to `POST`
    2. eneter the endpoint address
    3. select `body` right under the address bar
    4. select input format as `raw` and `JSON` for the body
    5. Then hit `send`



- You can play around with differnt values of the data in the JSON, and check your model's predictions.