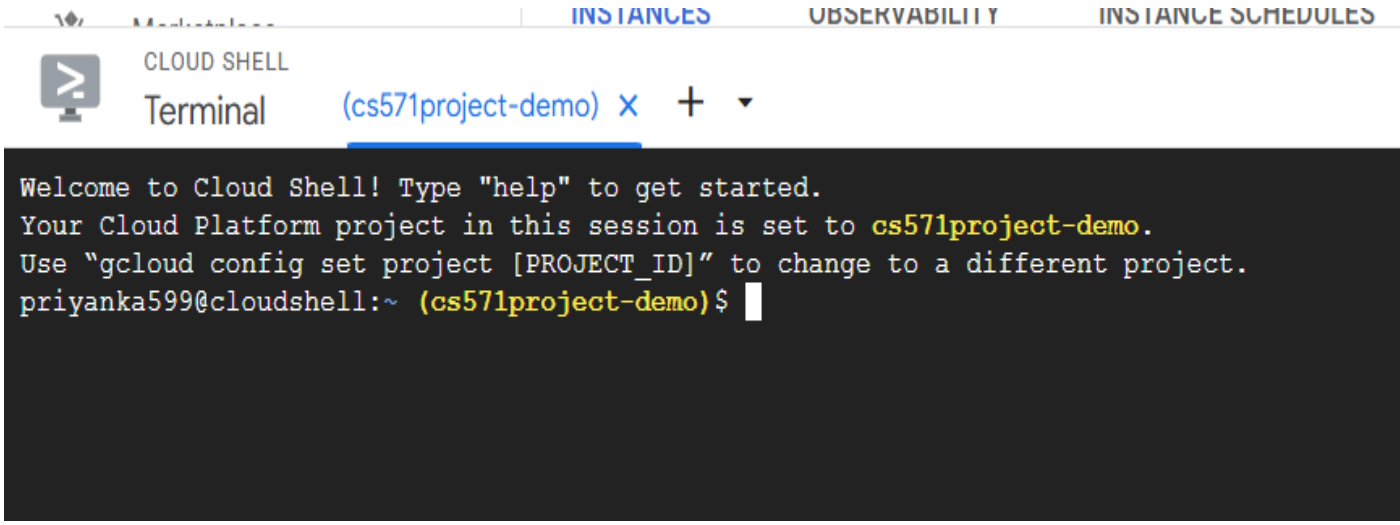
Week 10 Homework 1:

Project: Machine Learning on Kubernetes

Name: Priyanka, student-id: 20179

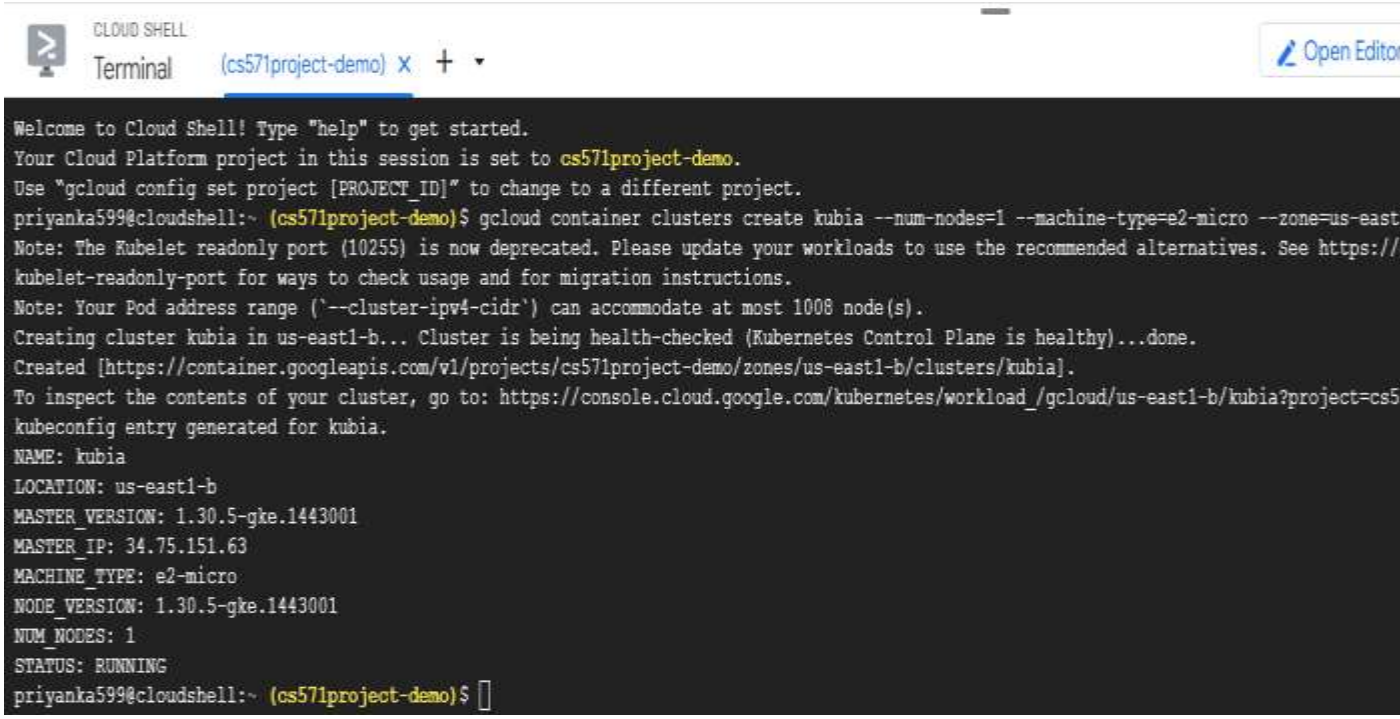
Step 1: Set up a functional Kubernetes cluster

1. Open GKE terminal



2. Now, lets create a kubenetes cluster with three nodes

gcloud container clusters create kubia --num-nodes=1 --machine-type=e2-micro --zone=us-east1-b



3. Double check if nodes are correctly created

kubectl get nodes

You should see three nodes being created:

```
priyanka599@cloudshell:~ (cs571project-demo)$ kubectl get nodes
NAME                                    STATUS   ROLES    AGE     VERSION
gke-kubia-default-pool-b40e64d7-k1c0    Ready    <none>   4m33s   v1.30.5-gke.1443001
priyanka599@cloudshell:~ (cs571project-demo)$
```

4. Start minikube in Google Cloud Platform

```
priyanka599@cloudshell:~ (cs571project-demo)$ minikube start
* minikube v1.34.0 on Ubuntu 24.04 (amd64)
  - MINIKUBE_FORCE_SYSTEMD=true
  - MINIKUBE_HOME=/google/minikube
  - MINIKUBE_WANTUPDATENOTIFICATION=false
* Automatically selected the docker driver. Other choices: ssh, none
* Using Docker driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.45 ...
* Downloading Kubernetes v1.31.0 preload ...
    > preloaded-images-k8s-v18-v1...:  326.69 MiB / 326.69 MiB  100.00% 254.93
    > gcr.io/k8s-minikube/kicbase...:  487.90 MiB / 487.90 MiB  100.00% 99.17 M
* Creating docker container (CPUs=2, Memory=4000MB) ...
* Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
  - kubelet.cgroups-per-qos=false
  - kubelet.enforce-node-allocatable=""
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring bridge CNI (Container Networking Interface) ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
priyanka599@cloudshell:~ (cs571project-demo)$
```

5. Create requirements.txt file using the following command –

nano requirements.txt

```
priyanka599@cloudshell:~ (cs571project-demo)$ nano requirements.txt
priyanka599@cloudshell:~ (cs571project-demo)$
```

Then enter the following contents

Flask==1.1.1

gunicorn==19.9.0

itsdangerous==1.1.0

Jinja2==2.10.1

MarkupSafe==1.1.1

Werkzeug==0.15.5

numpy==1.19.5 # Adjusted to a version before np.float deprecation

scipy>=0.15.1

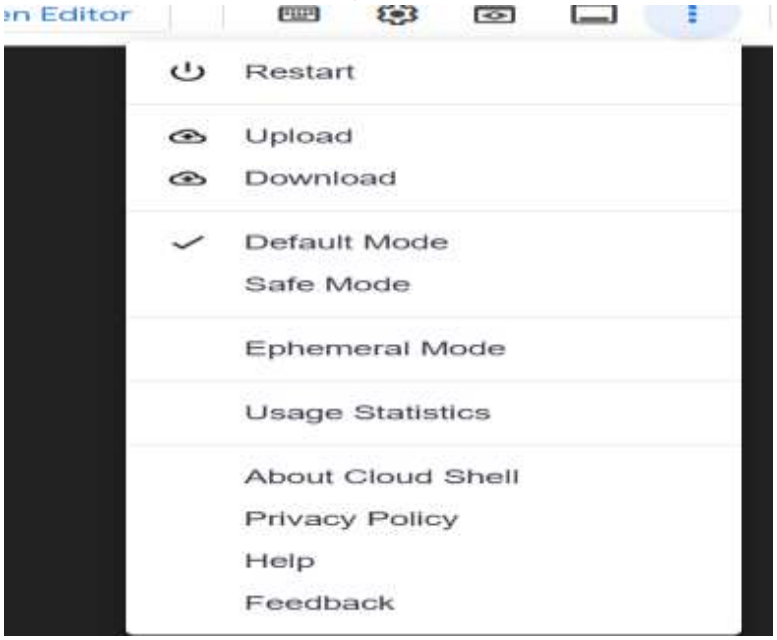
scikit-learn==0.24.2 # Ensure compatibility with numpy version

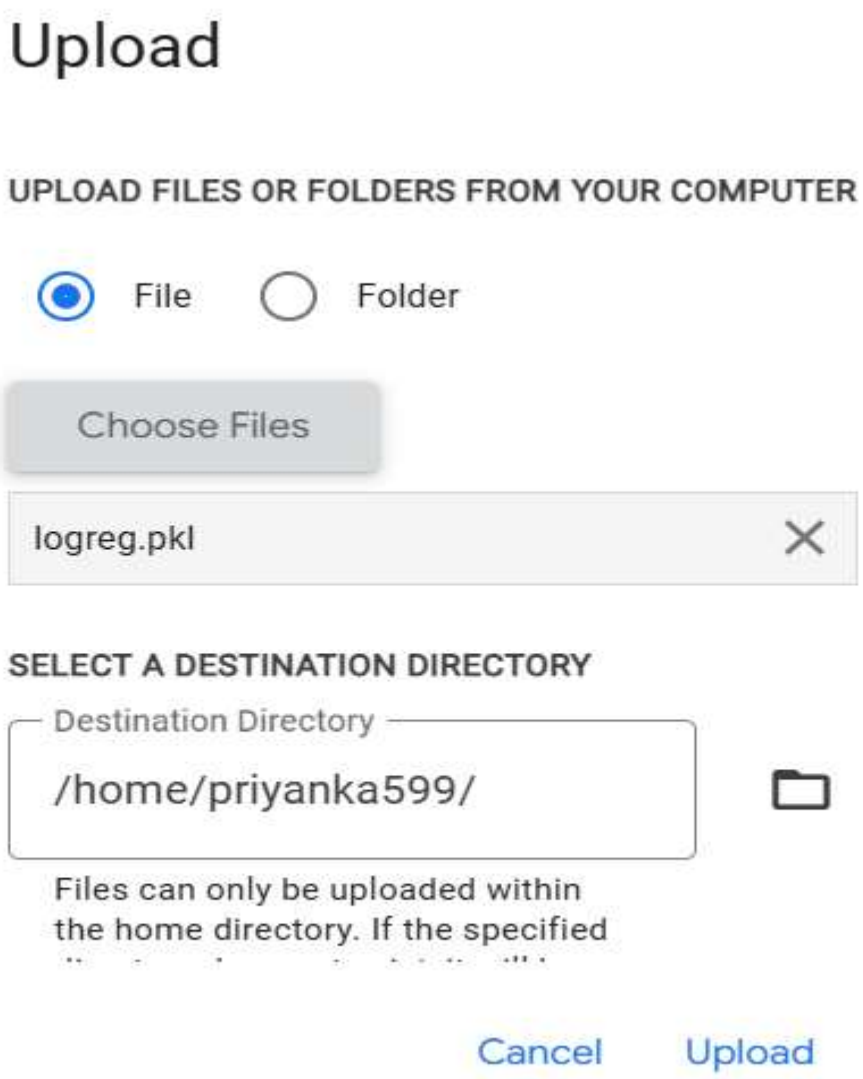
matplotlib>=1.4.3

pandas>=0.19

flasgger==0.9.4

6. Upload logreg.pkl file by clicking the three dots in the top-right part of the Cloud Shell Terminal and then choose upload





7. Create flask_api.py file using the command - nano flask_api.py

```
priyanka599@cloudshell:~ (cs571project-demo)$ nano flask_api.py
priyanka599@cloudshell:~ (cs571project-demo)$
```

```
# -*- coding: utf-8 -*-
"""
Created on Mon May 25 12:50:04 2020
@author: pramod.singh
"""
from flask import Flask, request
import numpy as np
```

```
import pickle
import pandas as pd
from flasgger import Swagger

app = Flask(__name__)
Swagger(app)

pickle_in = open("logreg.pkl", "rb")
model = pickle.load(pickle_in)

@app.route('/')
def home():
    return "Welcome to the Flask API!"

@app.route('/predict', methods=["GET"])
def predict_class():
    """Predict if Customer would buy the product or not.
    ---
    parameters:
      - name: age
        in: query
        type: number
        required: true
      - name: new_user
        in: query
        type: number
        required: true
      - name: total_pages_visited
        in: query
        type: number
        required: true
```

```
    responses:
      200:
        description: Prediction
    """
    age = int(request.args.get("age"))
    new_user = int(request.args.get("new_user"))
    total_pages_visited = int(request.args.get("total_pages_visited"))
    prediction = model.predict([[age, new_user, total_pages_visited]])
    return "Model prediction is " + str(prediction)

@app.route('/predict_file', methods=["POST"])
def prediction_test_file():
    """Prediction on multiple input test file.
    ---
    parameters:
      - name: file
        in: formData
        type: file
        required: true
    responses:
      200:
        description: Test file Prediction
    """
    df_test = pd.read_csv(request.files.get("file"))
    prediction = model.predict(df_test)
    return str(list(prediction))

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

```
  GNU nano 7.2
# -*- coding: utf-8 -*-
"""
Created on Mon May 25 12:50:04 2020
@author: pramod.singh
"""
from flask import Flask, request
import numpy as np
import pickle
import pandas as pd
from flasgger import Swagger

app = Flask(__name__)
Swagger(app)

pickle_in = open("logreg.pkl", "rb")
model = pickle.load(pickle_in)

@app.route('/')
def home():
    return "Welcome to the Flask API!"

@app.route('/predict', methods=["GET"])
def predict_class():
    """Predict if Customer would buy the product or not.
    ---
    parameters:
      - name: age
        in: query
```

Step 2: Dockerfile

1. Create Dockerfile using command –
   nano Dockerfile

```
priyanka599@cloudshell:~ (cs571project-demo)$ nano Dockerfile
priyanka599@cloudshell:~ (cs571project-demo)$
```

```
  GNU nano 7.2
FROM python:3.8-slim
WORKDIR /app
COPY . /app
EXPOSE 5000
RUN pip install -r requirements.txt
CMD ["python", "flask_api.py"]
```

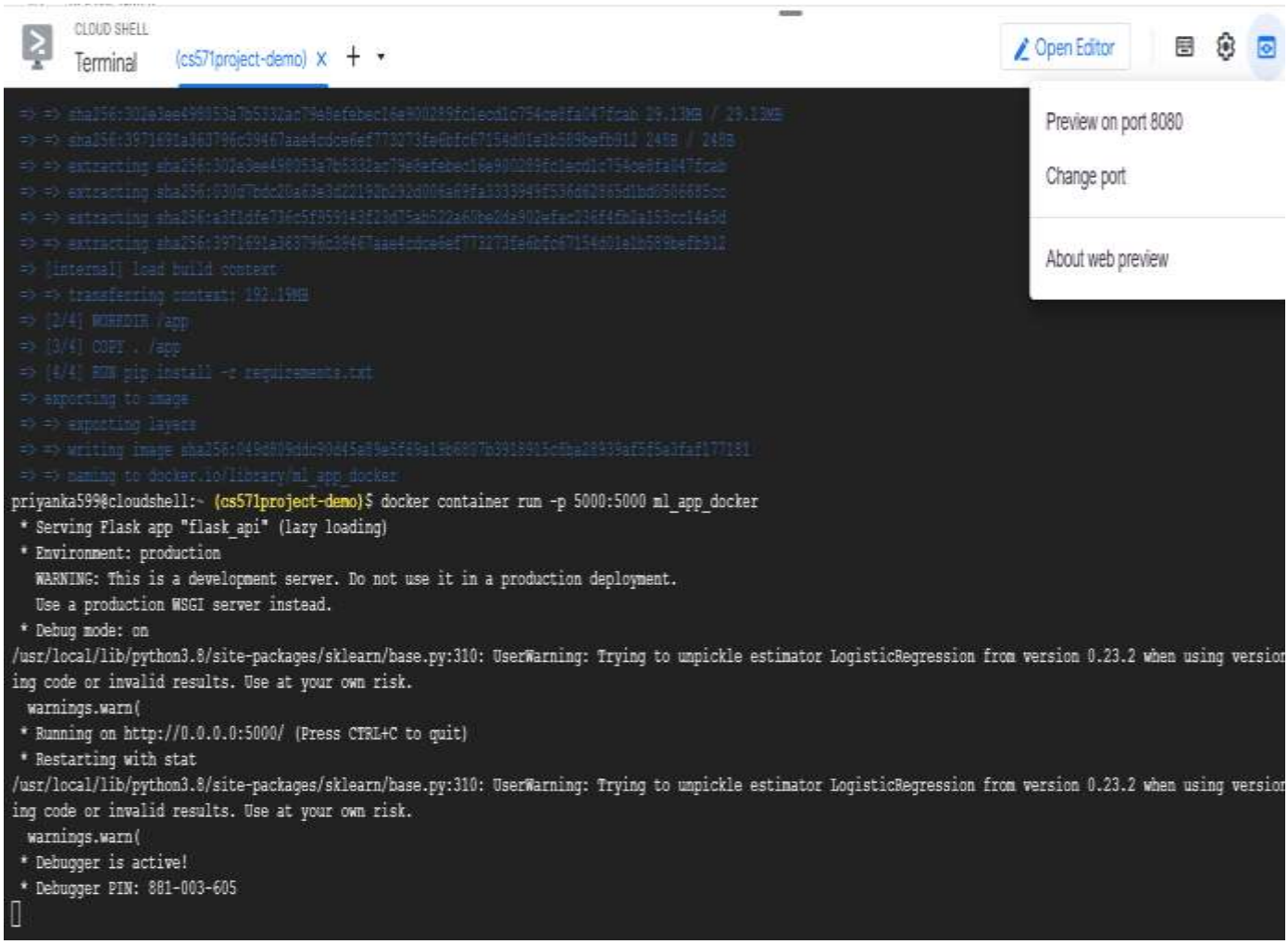
Step 3: Running the Docker Container

1. To build the docker image use the command –
   sudo docker build -t ml_app_docker .

```
priyanka599@cloudshell:~ (cs571project-demo)$ sudo docker build -t ml_app_docker
[+] Building 55.7s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 315B
 => [internal] load metadata for docker.io/library/python:3.8-slim
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/4] FROM docker.io/library/python:3.8-slim@sha256:1d52838af602b4b5a831beb13
 => => resolve docker.io/library/python:3.8-slim@sha256:1d52838af602b4b5a831beb13
 => => sha256:030d7bdc20a63e3d22192b292d006a69fa3333949f536d62865d1bd0506685cc 3.
 => => sha256:a3f1dfe736c5f959143f23d75ab522a60be2da902efac236f4fb2a153cc14a5d 14
 => => sha256:1d52838af602b4b5a831beb13a0e4d073280665ea7be7f69ce2382f29c5a613f 10
 => => sha256:314bc2fb0714b7807bf5699c98f0c73817e579799f2d91567ab7e9510f5601a5 1.
 => => sha256:b5f62925bd0f63f48cc8acd5e87d0c3a07e2f229cd2fb0a9586e68ed17f45ee3 5.
 => => sha256:302e3ee498053a7b5332ac79e8efebec16e900289fc1ecd1c754ce8fa047fcab 29
 => => sha256:3971691a363796c39467aae4cdce6ef773273fe6bfc67154d01e1b589befb912 24
 => => extracting sha256:302e3ee498053a7b5332ac79e8efebec16e900289fc1ecd1c754ce8f
 => => extracting sha256:030d7bdc20a63e3d22192b292d006a69fa3333949f536d62865d1bd0
 => => extracting sha256:a3f1dfe736c5f959143f23d75ab522a60be2da902efac236f4fb2a15
 => => extracting sha256:3971691a363796c39467aae4cdce6ef773273fe6bfc67154d01e1b58
 => [internal] load build context
 => => transferring context: 192.19MB
 => [2/4] WORKDIR /app
 => [3/4] COPY . /app
 => [4/4] RUN pip install -r requirements.txt
 => exporting to image
 => => exporting layers
 => => writing image sha256:049d809ddc90d45a89e5f69a19b6807b3918915c8ba28939af5f5
 => => naming to docker.io/library/ml_app_docker
priyanka599@cloudshell:~ (cs571project-demo)$
```

2. This command runs a Docker container from the ml_app_docker image:

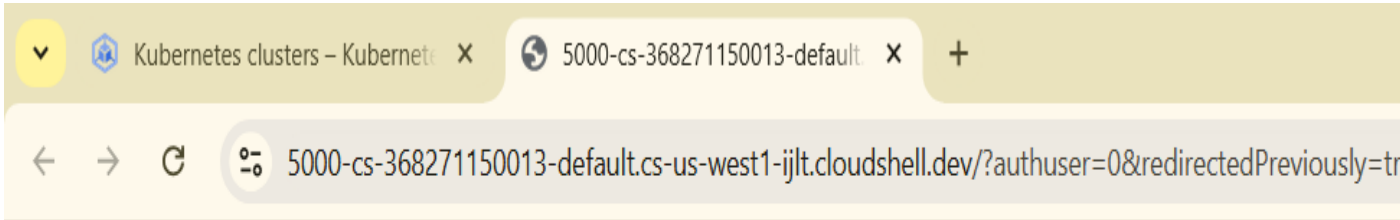
- docker container run -p 5000:5000 ml_app_docker



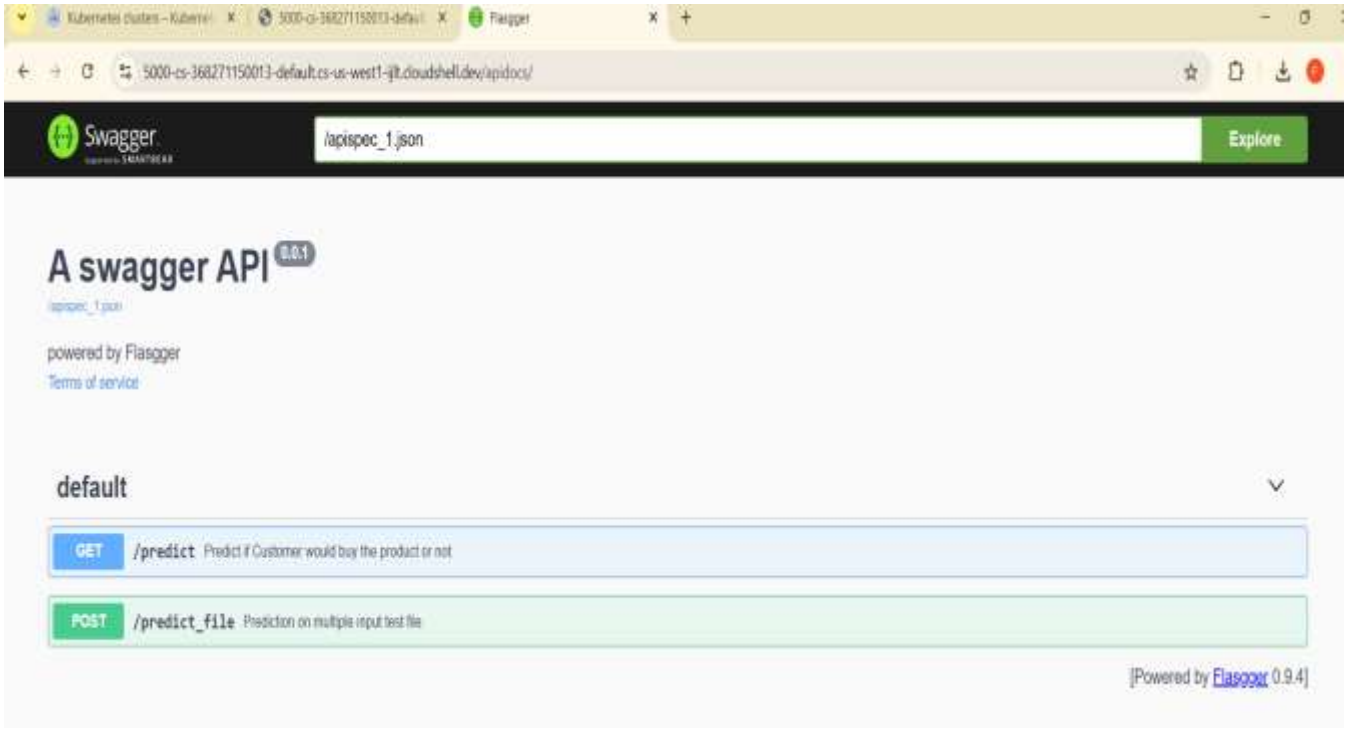
3.In the right-upper side of the terminal click the eye shaped button and then click Preview on port 5000.

Change port if it is not 5000 by default.

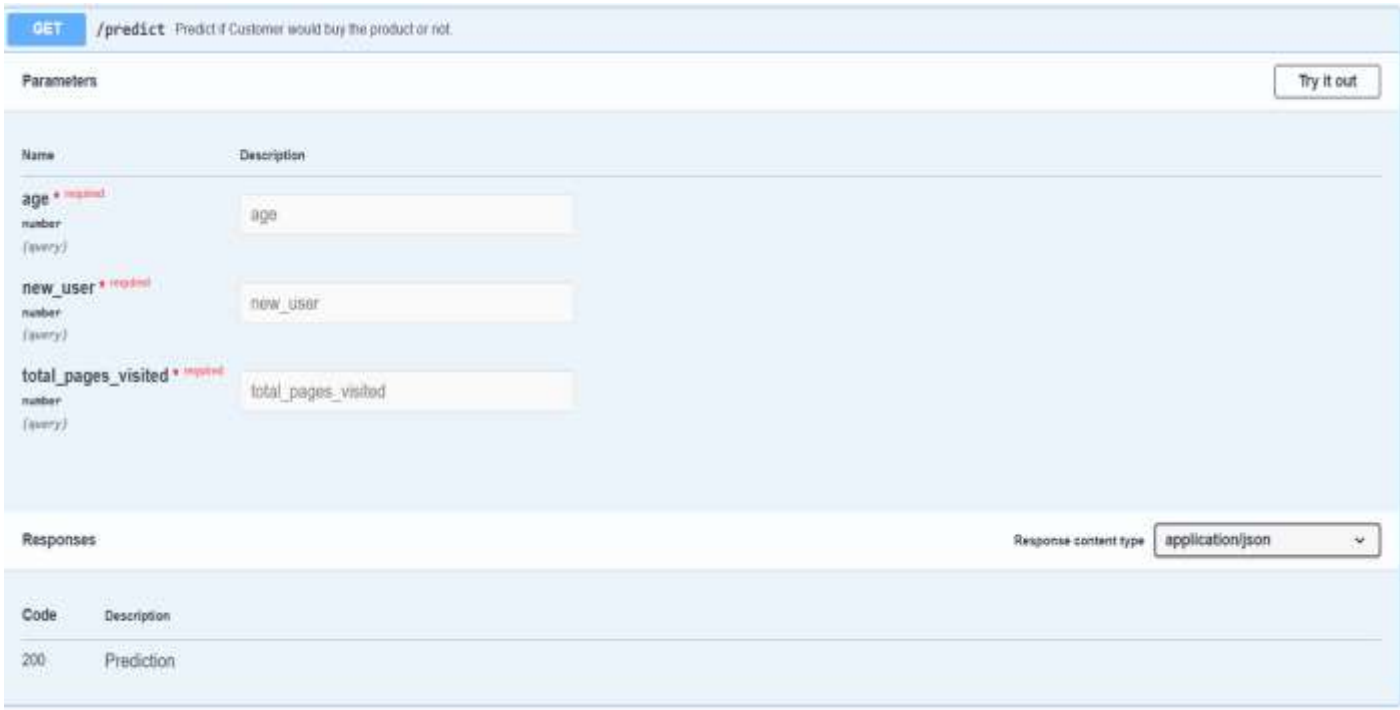
4. You will see this using the web preview.

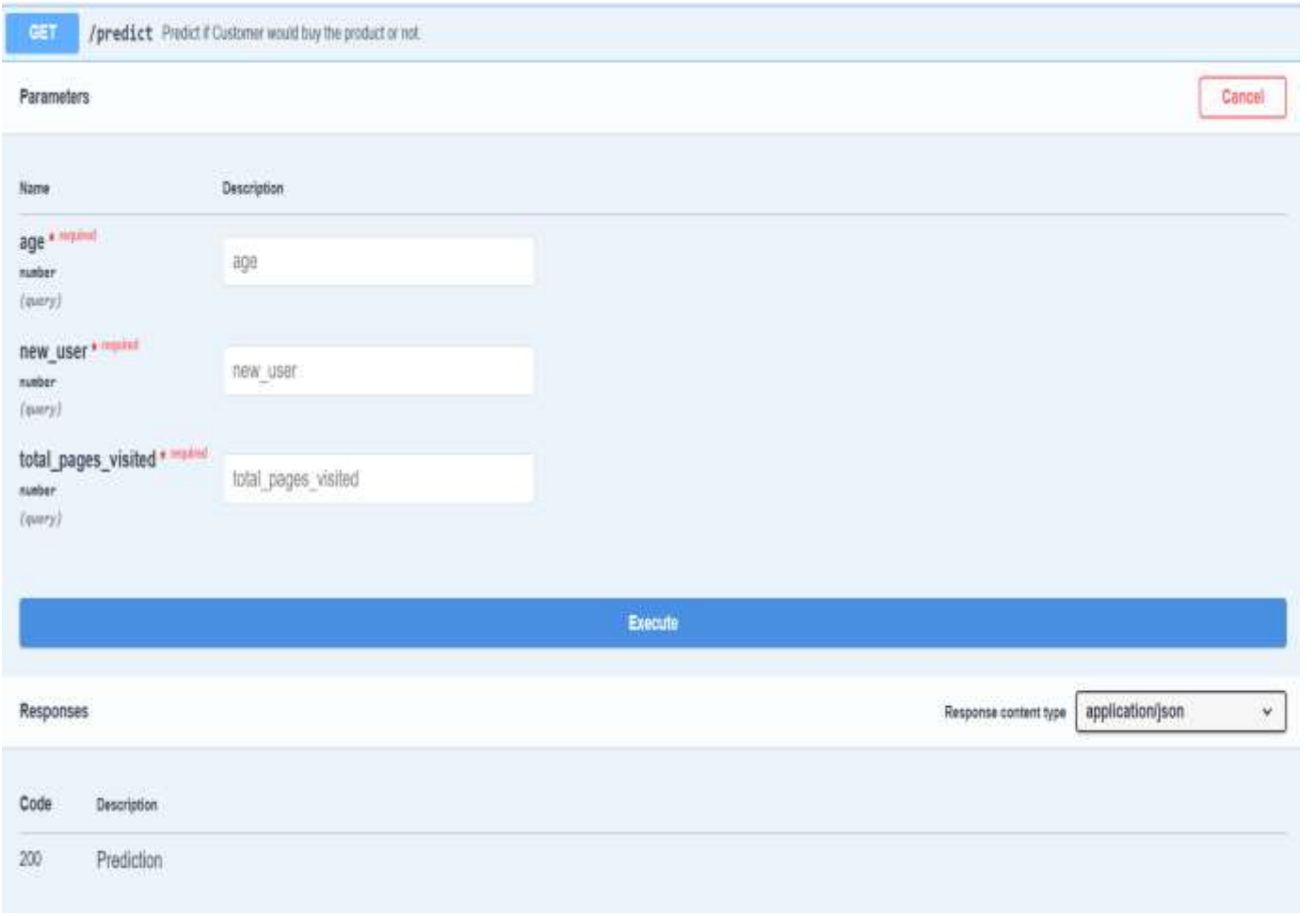


5. Add /apidocs/ at the end of the link to access the running ml- app as following - There are two tabs GET and POST.

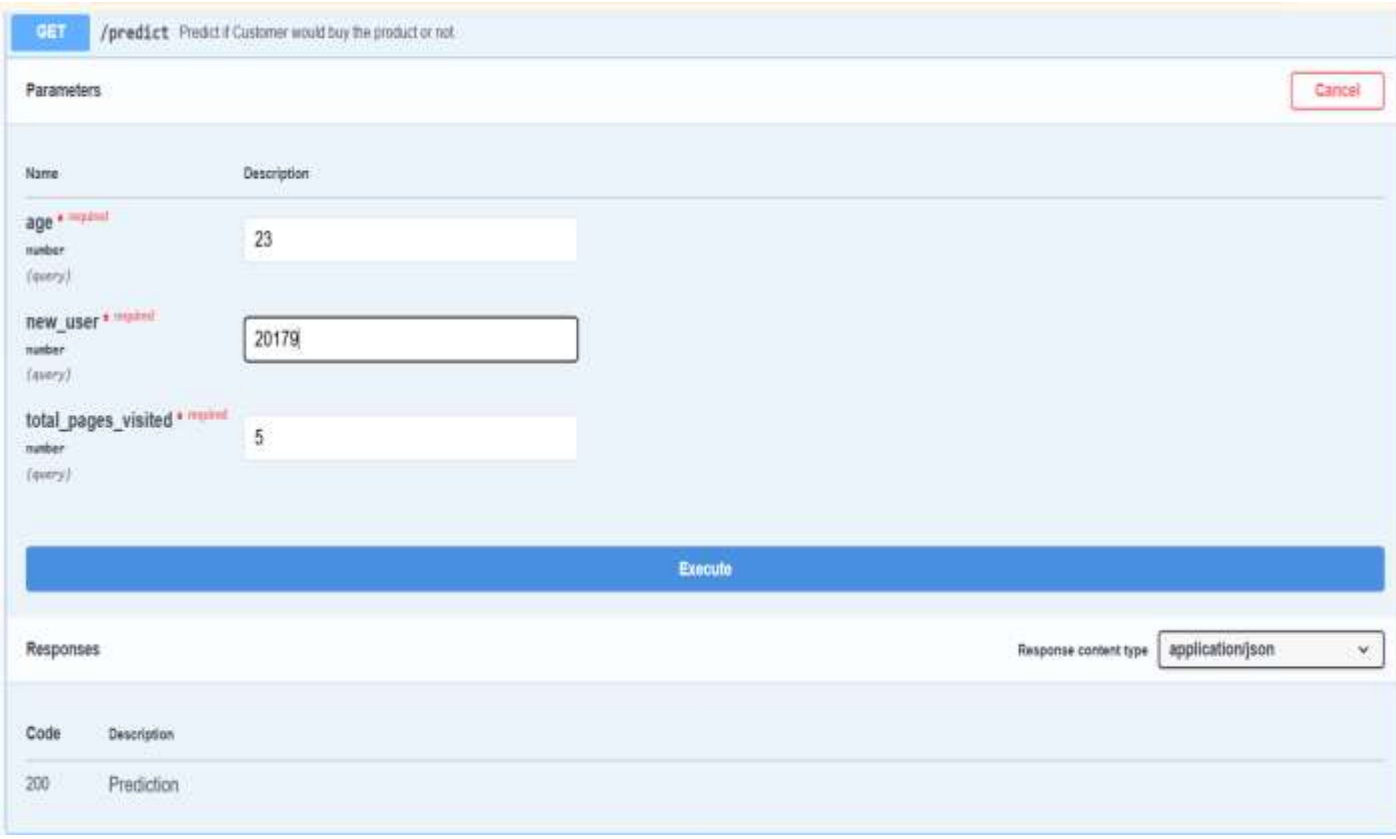


6. Click GET and then click Try it out in the top-right corner of the GET box.



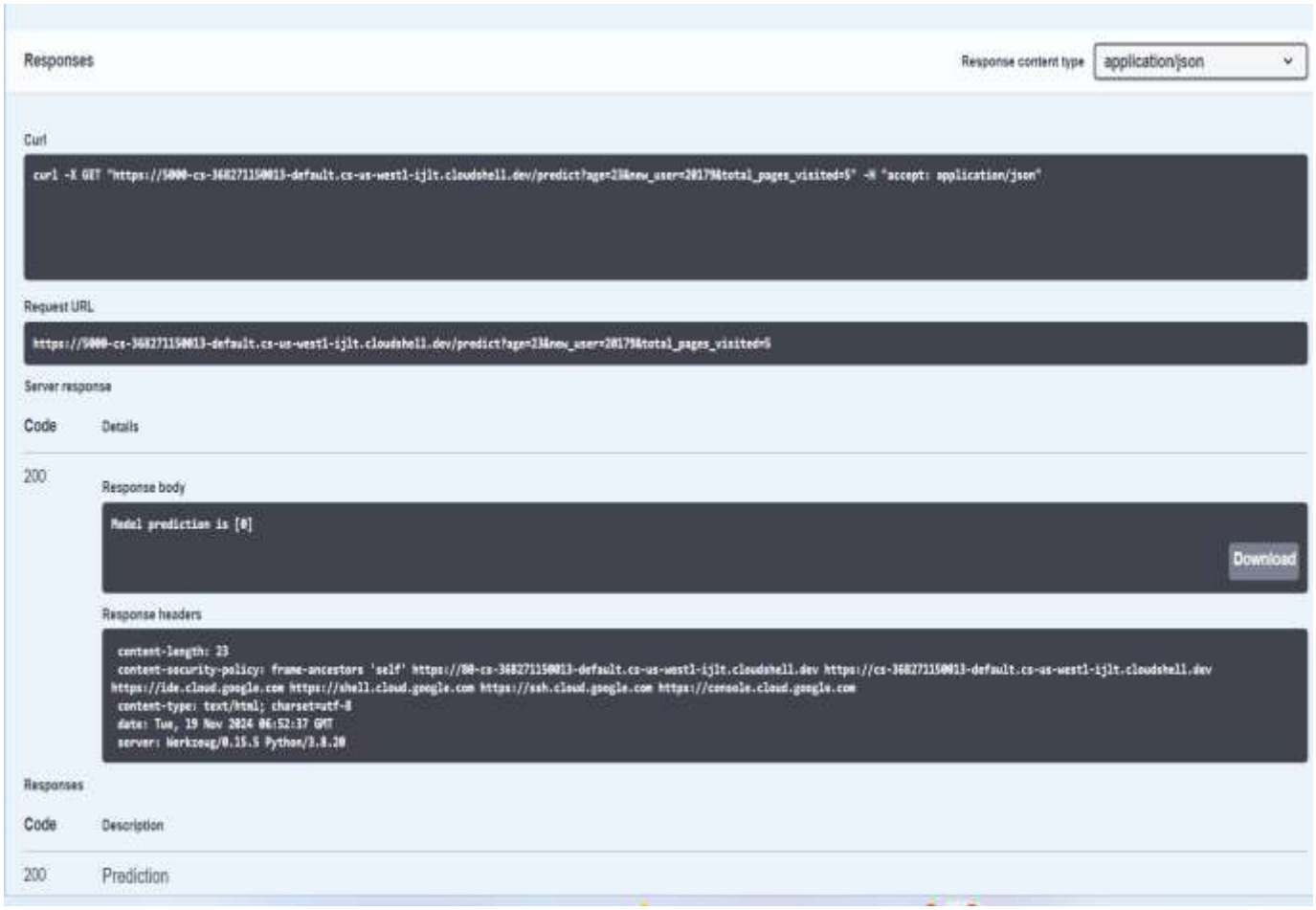


7. Fill values for the input parameters and then click Execute

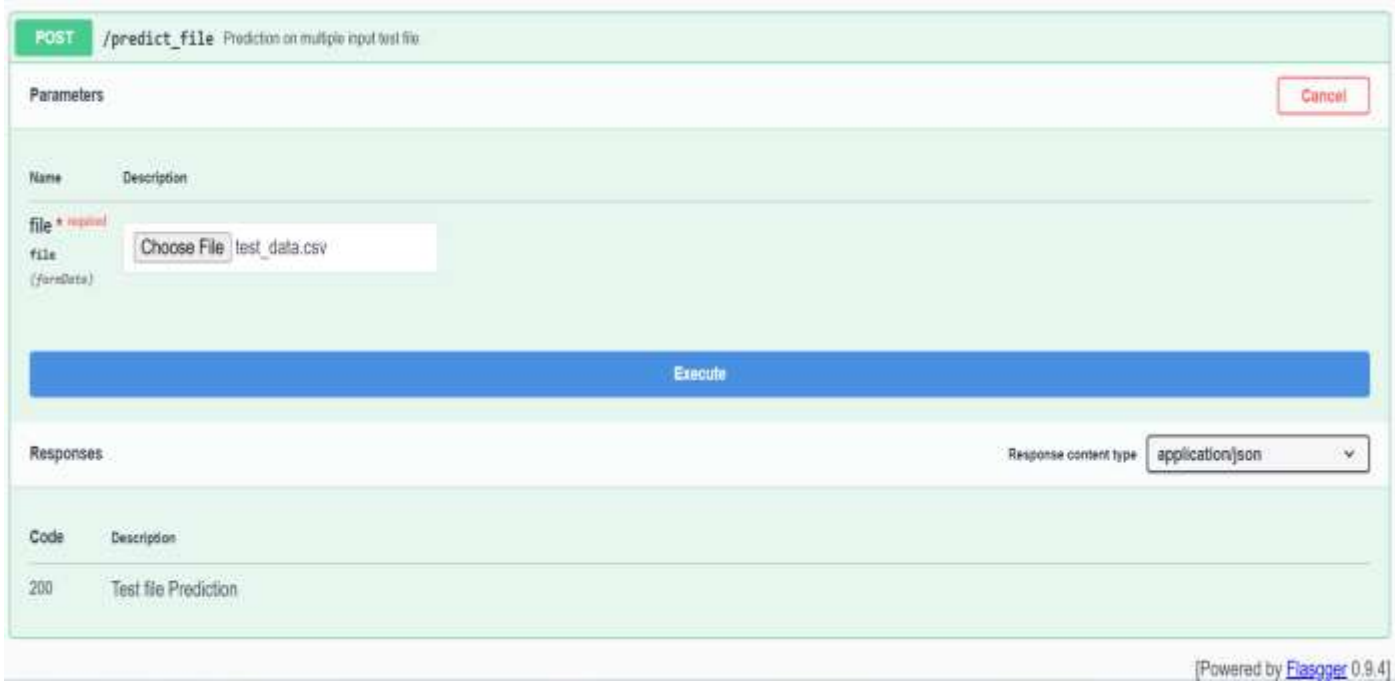


8. Upon the execution call, the request goes to the app, and predictions are made by the model.

- The result of the model prediction is displayed in the Prediction section of the page as following

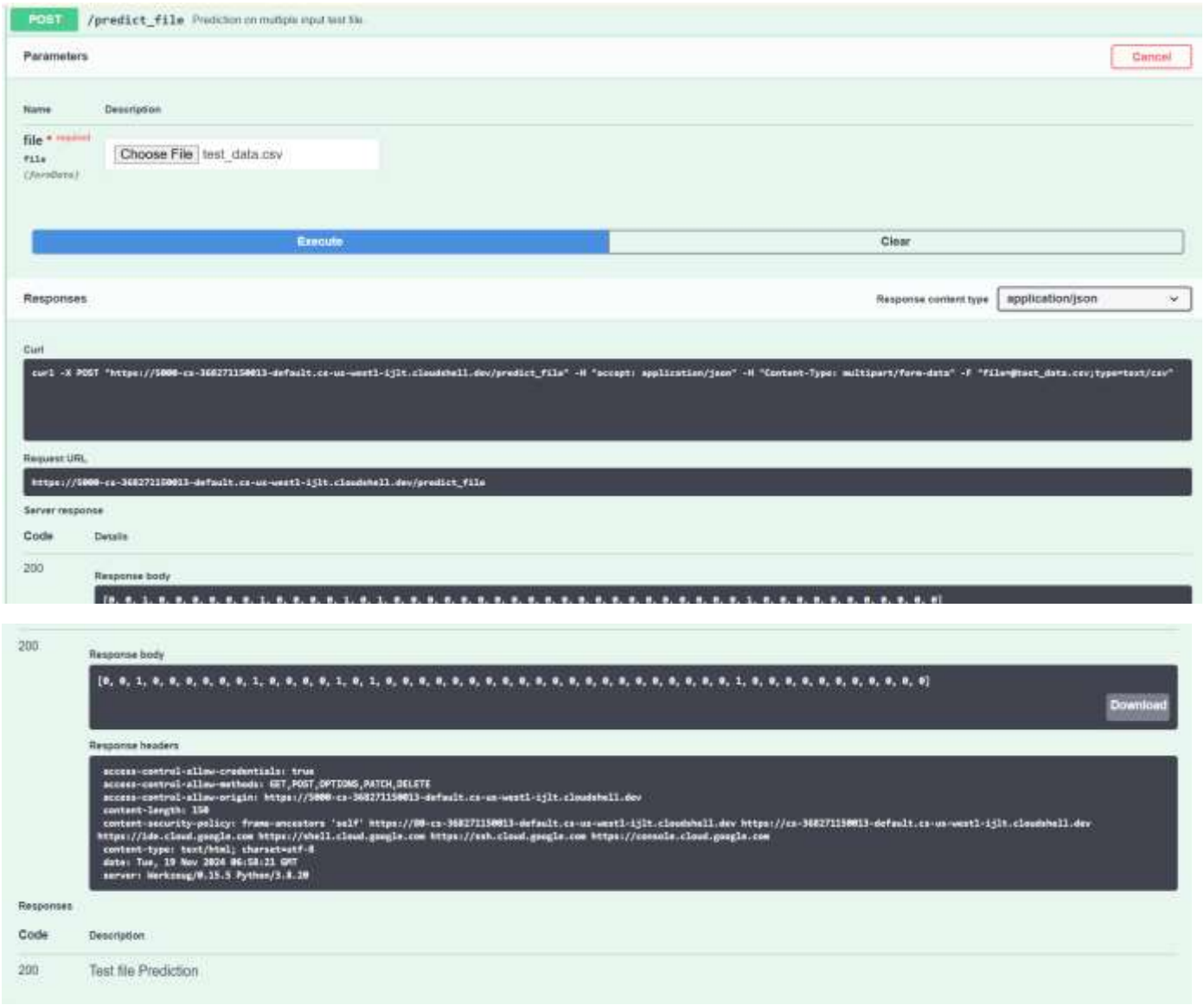


9.The next prediction that can be done is for a group of customers (test data) via a post request.



10. Upload the test data file containing the same parameters in a similar order.

The model would make the prediction, and the results would be displayed upon execute as following.



Step 4: Stopping/killing the running container

1. Use docker ps to list running Docker containers

```
priyanka599@cloudshell:~ (cs571project-demo)$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED         STATUS         PORTS
                                                      NAMES
4b6b975c6599   gcr.io/k8s-minikube/kicbase:v0.0.45   "/usr/local/bin/entr…"   58 minutes ago   Up 58 minutes   127.0.0.1:32768->22/tcp, 127.0.0.1:32769->2376/t
27.0.0.1:32771->8443/tcp, 127.0.0.1:32772->32443/tcp   minikube
priyanka599@cloudshell:~ (cs571project-demo)$
```

The CONTAINER_ID is given as 4b6b975c6599

2. Use the command

- docker kill to kill the running container as follows.

```
priyanka599@cloudshell:~ (cs571project-demo)$ docker kill 4b6b975c6599
4b6b975c6599
priyanka599@cloudshell:~ (cs571project-demo)$
```