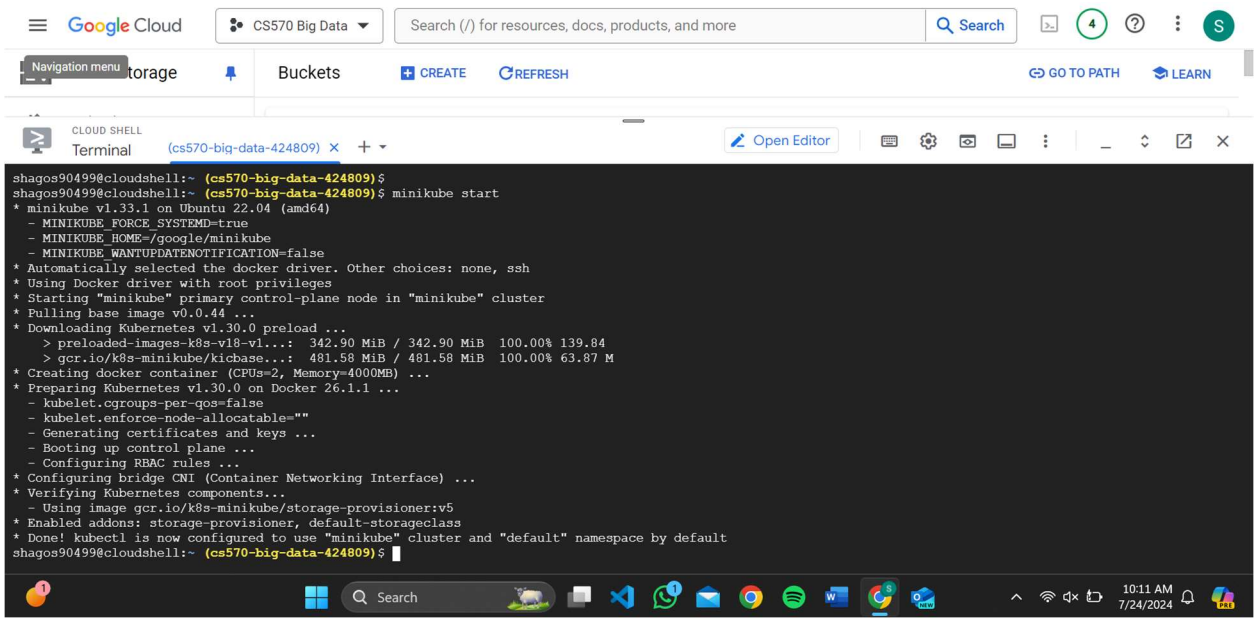
Machine Learning on Kubernetes

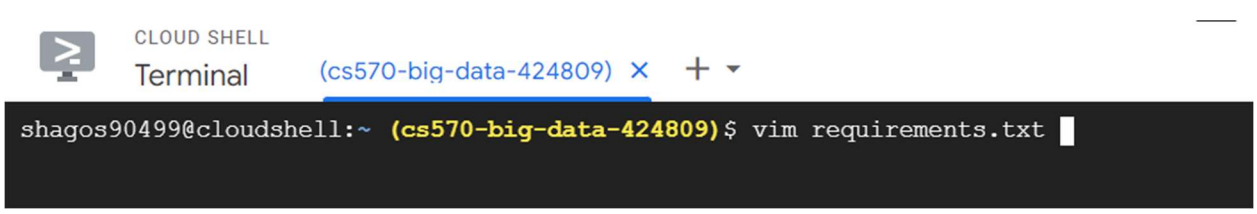
Step 1: Setting Up the Environment

1. **Start Minikube in GCP**
   - Open Cloud Shell and since minikube can be accessed and is already configured, you can start minikube immediately.
   - Begin by starting Minikube on Google Cloud Platform.

c



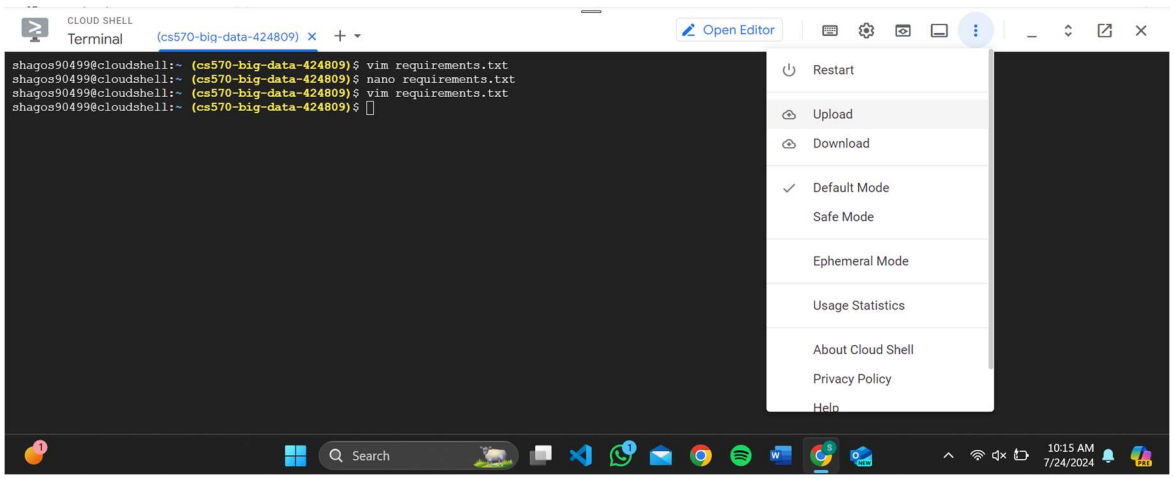
2. **Create `requirements.txt`**
   - Use the command `vim requirements.txt` to create the file and add the following dependencies:



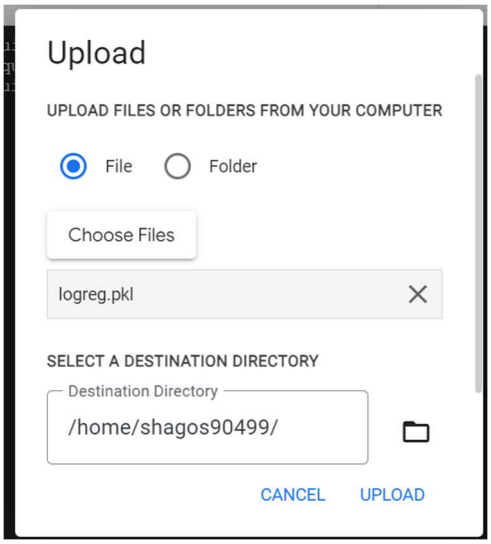

```
Flask==1.1.1
gunicorn==19.9.0
itsdangerous==1.1.0
Jinja2==2.10.1
MarkupSafe==1.1.1
Werkzeug==0.15.5
numpy==1.19.5  # Adjusted to a version before np.float
deprecation
scipy>=0.15.1
scikit-learn==0.24.2  # Ensure compatibility with numpy version
```

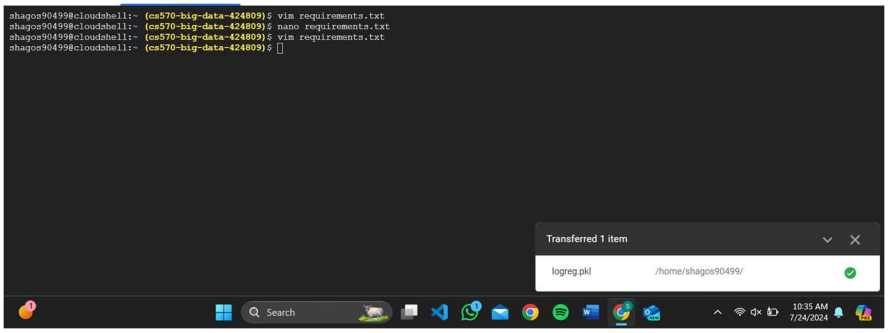
```
matplotlib>=1.4.3
pandas>=0.19
flasgger==0.9.4
```

3. **Upload `logreg.pkl` File**
   - Click the three dots in the top-right corner of the Cloud Shell Terminal, select "Upload", and upload `logreg.pkl`.

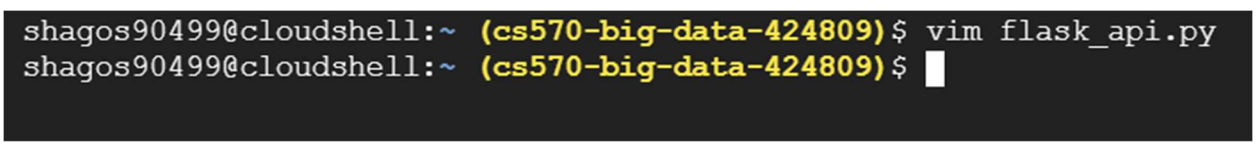


Then we can upload the file by selecting upload file:





4. **Create `flask_api.py`**
   - Use `vim flask_api.py` to create the file and enter the following code:




```
# -*- 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)
```

```python
# -*- 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
```

Step 2: Docker Configuration

1. **Create `Dockerfile`**
   - Use `vim Dockerfile` and add the following content:

```
shagos90499@cloudshell:~ (cs570-big-data-424809)$ vim Dockerfile
shagos90499@cloudshell:~ (cs570-big-data-424809)$
```

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

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

2. **Dockerfile Explanation**
    - **FROM python:3.8-slim**: Base image with Python 3.8.
    - **WORKDIR /app**: Sets the working directory.
    - **COPY . /app**: Copies the current directory's contents.
    - **EXPOSE 5000**: Specifies the port to expose.
    - **RUN pip install -r requirements.txt**: Installs dependencies.
    - **CMD ["python", "flask_api.py"]**: Starts the Flask application.

Step 3: Build and Run Docker Container

1. **Build Docker Image**
    - Run the command: `sudo docker build -t ml_app_docker .`

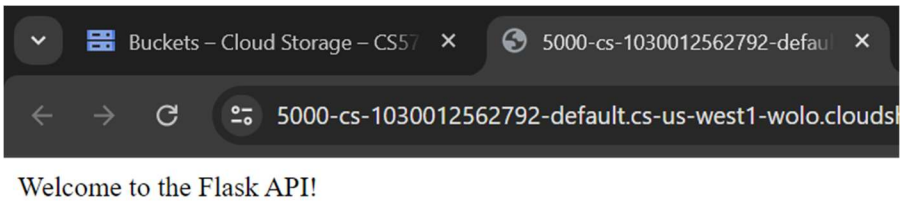
```
shagos90499@cloudshell:~ (cs570-big-data-424809)$ sudo docker build -t ml_app_docker .
[+] Building 54.3s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 163B
 => [internal] load metadata for docker.io/library/python:3.8-slim
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build context
 => => transferring context: 69.42MB
 => [1/4] FROM docker.io/library/python:3.8-slim@sha256:c3150277518bc546eec25a0c97469de06c047d68156a85a4db396d8bf1b58e5e
 => => resolve docker.io/library/python:3.8-slim@sha256:c3150277518bc546eec25a0c97469de06c047d68156a85a4db396d8bf1b58e5e
 => => sha256:b2d6266a63eafff4fcee6a9149b80686ff6c17b0e4556d0320198ce85e9aa41d 1.94kB / 1.94kB
 => => sha256:5208c64e1783e15b2d0ee357a3979688ae3faa21c6a43f4965e3530d68abbcfc 6.93kB / 6.93kB
 => => sha256:efc2b5ad9eec05befa54239d53feeae3569ccbef689aa5e5dbfc25da6c4df559 29.13MB / 29.13MB
 => => sha256:0d935f02ede5b557e3899b4161a3a7f7dd8461fd62d558451b3884172a710962 3.51MB / 3.51MB
 => => sha256:fea4f2170757fa9e87b6421086bf5d32880bd00955594164110 8ea89519b5742 11.67MB / 11.67MB
 => => sha256:c3150277518bc546eec25a0c97469de06c047d68156a85a4db396d8bf1b58e5e 10.41kB / 10.41kB
 => => sha256:e5635d0cdd4c514a4e76d97174785a452a1e81b87b6a8bd8ce09c5ac2d135df8 230B / 230B
 => => sha256:ebe530eb534fda723884e0b61fa4633bb792a9600f452a779301c8a7e4216d83 2.78MB / 2.78MB
 => => extracting sha256:efc2b5ad9eec05befa54239d53feeae3569ccbef689aa5e5dbfc25da6c4df559
 => => extracting sha256:0d935f02ede5b557e3899b4161a3a7f7dd8461fd62d558451b3884172a710962
 => => extracting sha256:fea4f2170757fa9e87b6421086bf5d32880bd00955594164110 8ea89519b5742
 => => extracting sha256:e5635d0cdd4c514a4e76d97174785a452a1e81b87b6a8bd8ce09c5ac2d135df8
 => => extracting sha256:ebe530eb534fda723884e0b61fa4633bb792a9600f452a779301c8a7e4216d83
 => [2/4] WORKDIR /app
 => [3/4] COPY . /app
 => [4/4] RUN pip install -r requirements.txt
 => exporting to image
 => => exporting layers
 => => writing image sha256:476c08e318dbf19dcf9a6af702009bf576ab6603e3157f45a223f0471bb4eb12
 => => naming to docker.io/library/ml_app_docker
shagos90499@cloudshell:~ (cs570-big-data-424809)$
```

2. **Run Docker Container**
    - Execute: `docker container run -p 5000:5000 ml_app_docker`

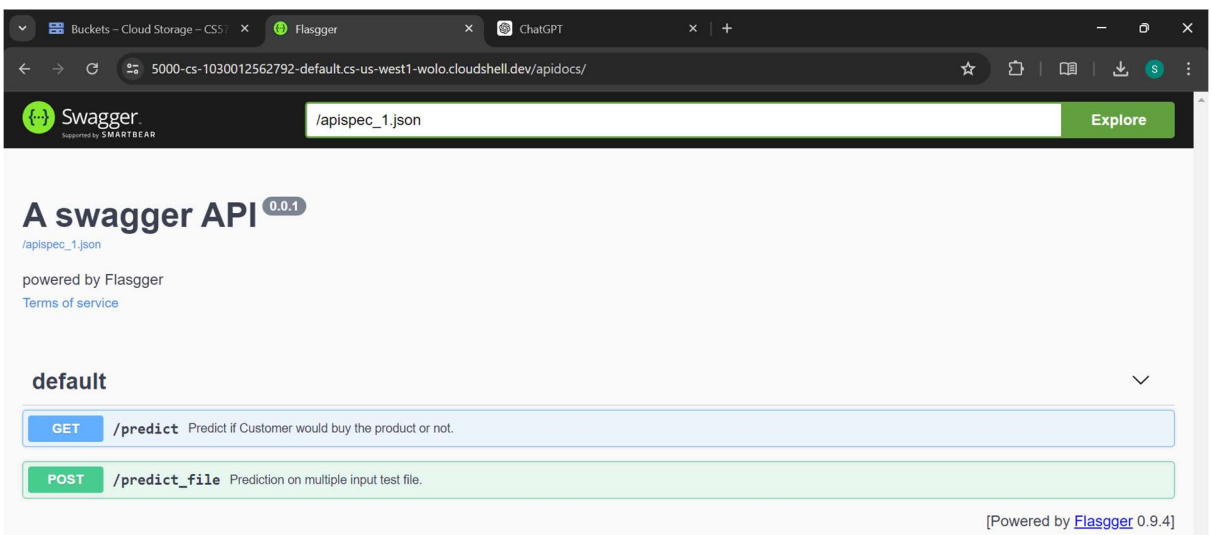
```
shagos90499@cloudshell:~ (cs570-big-data-424809)$ docker container run -p 5000:5000 ml_app_docker
 * Serving Flask app "flask_api" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
/usr/local/lib/python3.8/site-packages/sklearn/base.py:310: UserWarning: Trying to unpickle estimator Logisti
lead to breaking code or invalid results. Use at your own risk.
  warnings.warn(
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Restarting with stat
/usr/local/lib/python3.8/site-packages/sklearn/base.py:310: UserWarning: Trying to unpickle estimator Logisti
lead to breaking code or invalid results. Use at your own risk.
  warnings.warn(
 * Debugger is active!
 * Debugger PIN: 227-240-498
172.17.0.1 - - [24/Jul/2024 17:44:59] "GET /?authuser=0&redirectedPreviously=true HTTP/1.1" 200 -
172.17.0.1 - - [24/Jul/2024 17:44:59] "GET /favicon.ico HTTP/1.1" 404 -
```

3. **Access Application**
    - Click the eye icon in the terminal, select "Preview on port 5000", and verify the port. If the port number is not 5000, click on change port and change it to 5000.

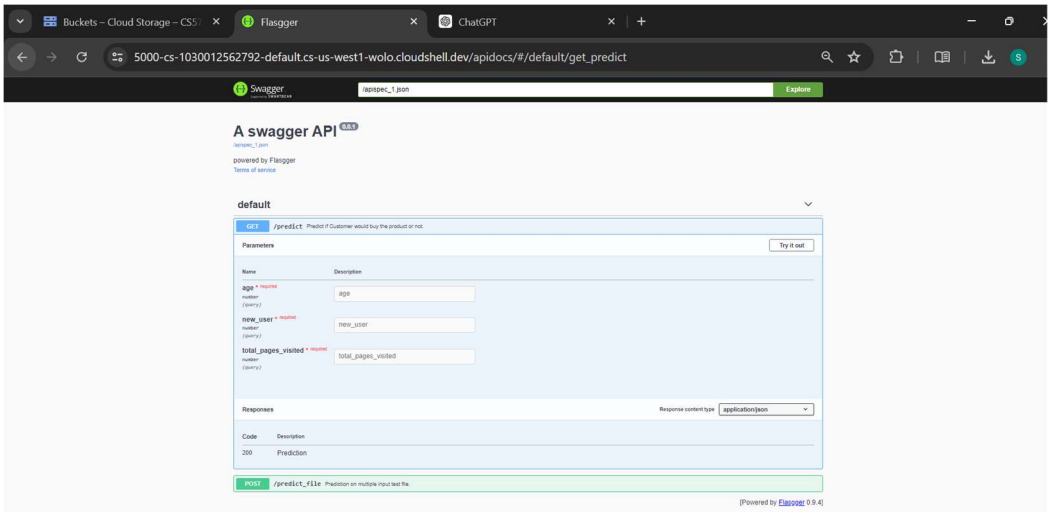
This is the preview:



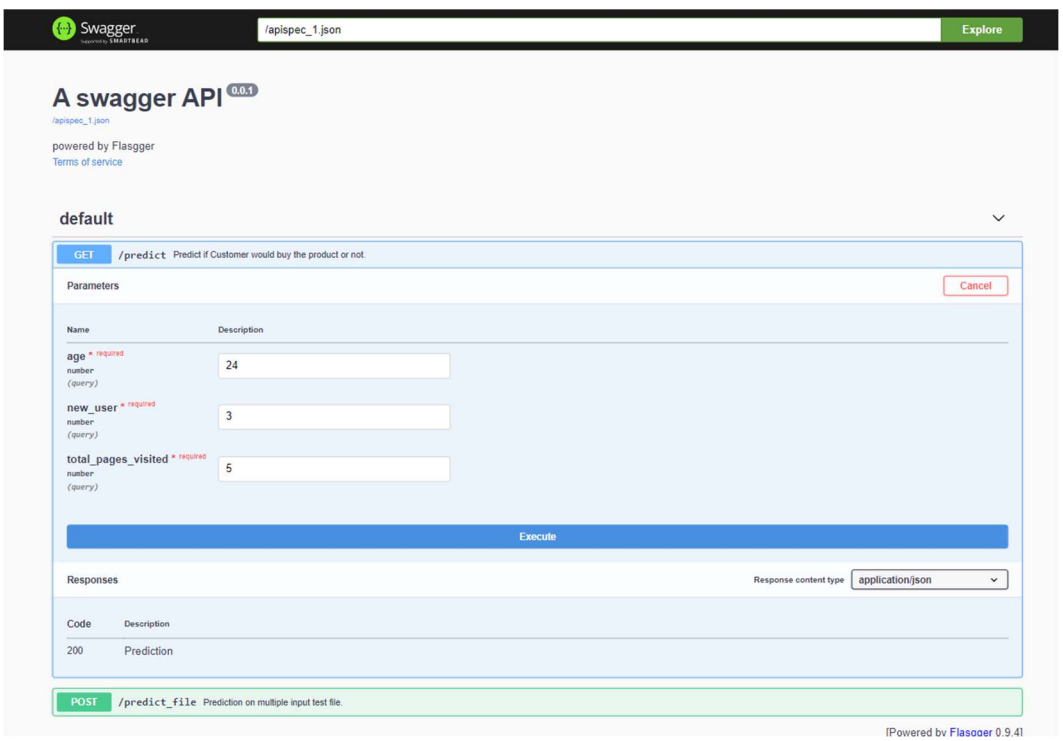
- Append `/apidocs/` to the URL to access the API documentation. Use the GET and POST tabs for testing.



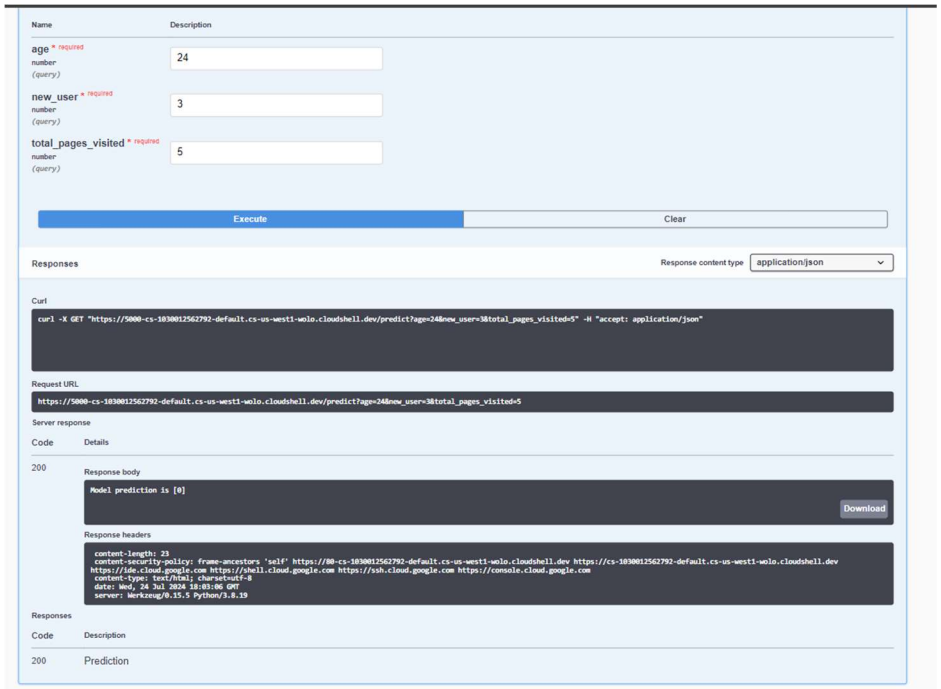
4. **Test API**
   - Use the GET endpoint, fill input parameters, and click "Execute".
     1. Click *GET* and then click *Try it out* in the top-right corner of the GET box.



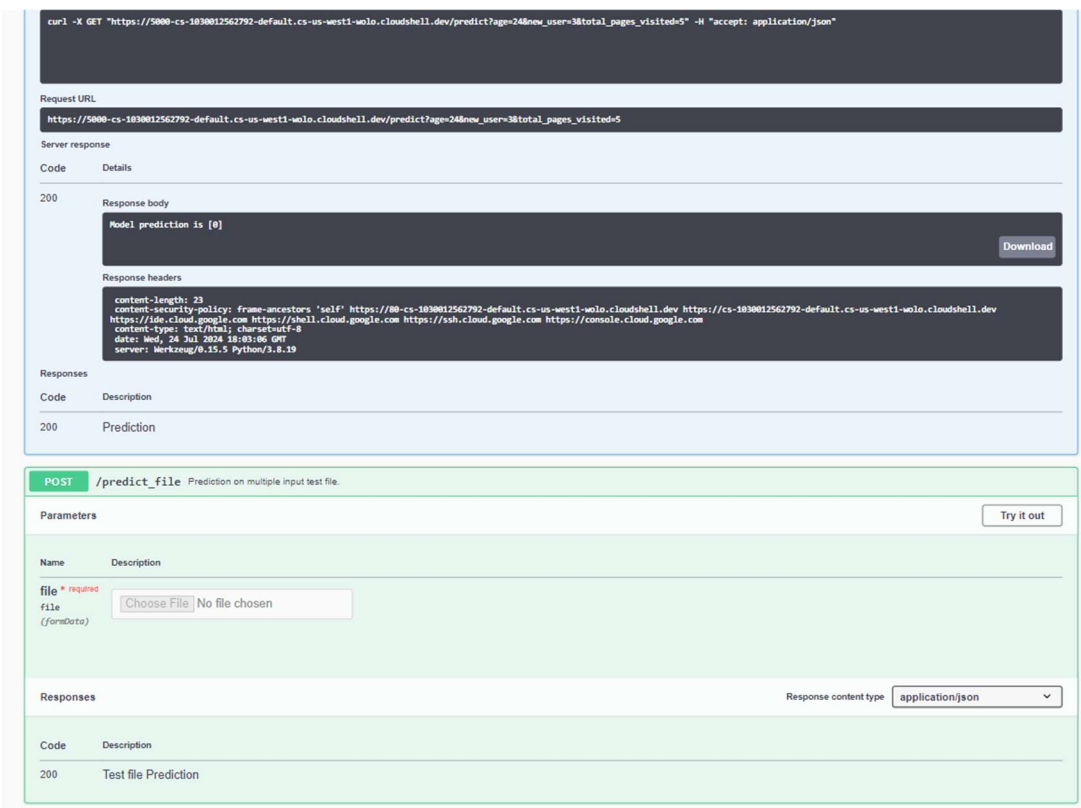
2. Fill values for the input parameters and then click Execute.



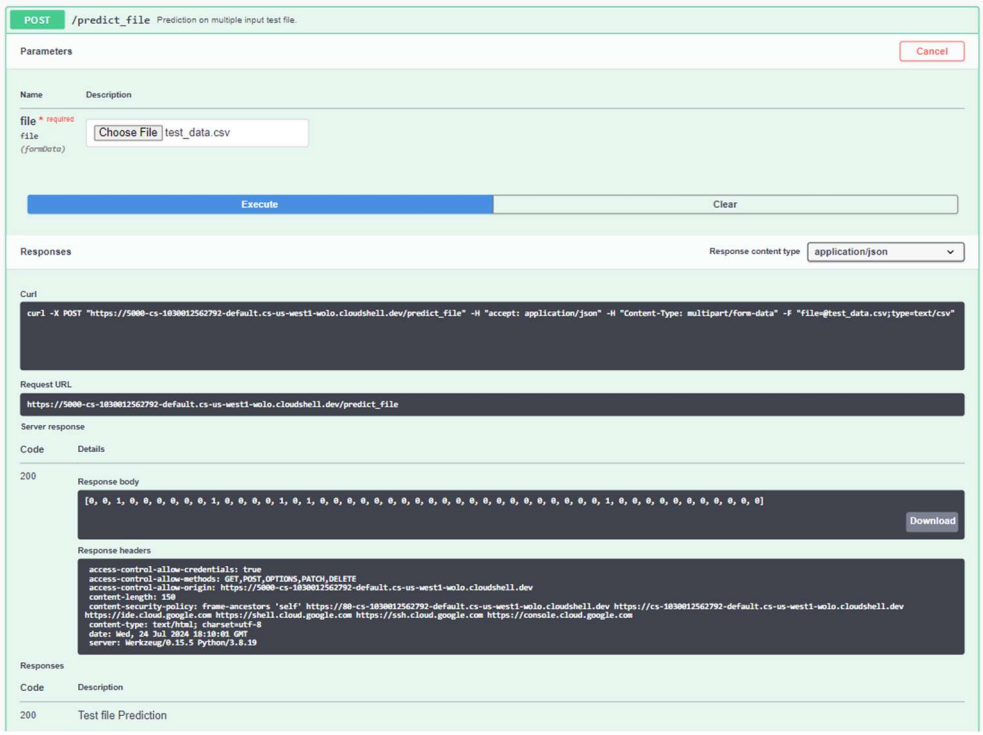
3. Upon the execution call, the request goes to the app, and predictions are made by the model. - The result of the model prediction is displayed in the Prediction section of the page as following



- For batch predictions, use the POST endpoint with a test data file.
    1. The next prediction that can be done is for a group of customers (test data) via a post request. Click on the POST button and Try it out.



2. 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: Manage Docker Container

1. **List Running Containers**
   - Use `docker ps` to find the `CONTAINER_ID`.

```
shagos90499@cloudshell:~ (cs570-big-data-424809)$ docker ps
CONTAINER ID   IMAGE                                  COMMAND                  CREATED             STATUS              PORTS
                                                     NAMES
038167161e8f   gcr.io/k8s-minikube/kicbase:v0.0.44    "/usr/local/bin/entr…"   About an hour ago   Up About an hour    127.0.0.1:32768->22/tcp, 127.0.0.1:32769->2376/tcp, 127.0.0.1:32770->5000/
0.1:32771->8443/tcp, 127.0.0.1:32772->32443/tcp   minikube
shagos90499@cloudshell:~ (cs570-big-data-424809)$
```

   My docker ID is 0 38167161e8f

2. **Stop the Container**
   - Execute: `docker kill <CONTAINER_ID>`

```
shagos90499@cloudshell:~ (cs570-big-data-424809)$ docker kill 038167161e8f
038167161e8f
shagos90499@cloudshell:~ (cs570-big-data-424809)$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS   PORTS   NAMES
shagos90499@cloudshell:~ (cs570-big-data-424809)$
```