Web Application and API deployment on Cloud: Google Cloud (GCP)

For this activity, we will be using the same toy data available on sklearn to build the model for a classification problem. We will use wine data and classify them into wine categories. In this article, we will create a pretrained classifier and then will save our classification using Pickle. Using the same pickle file, we will convert our classifier into an API and a web application. We will deploy both the applications, web application and the API based application in Google Cloud (GCP).

Let's create a machine learning model now.

Create a machine learning model:

We will load the wine recognition data from sklearn. The dataset contains 13 numeric features and target class. Target column has 3 different values 0, 1, 2. Based on 13 features of the wine data we will classify the wine into these three categories.

```
#All necessary imports
from sklearn import datasets
import pandas as pd # Import pandas

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

```
toy_data = datasets.load_wine()
```

```
print(toy_data.keys())
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'])
```

```
# Add a target column, and fill it with the target data
df_toydata['target'] = toy_data.target
# Show the first five rows
df_toydata.head()
```

y_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target
15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	0
11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	0
18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	0
16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	0
21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	0

```
df_toydata['target'].unique()
```

```
array([0, 1, 2])
```

We can see, we have to classify the wine into three categories: 0, 1, 2.

```
df_toydata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
 7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
 11  od280/od315_of_diluted_wines  178 non-null    float64
 12  proline                       178 non-null    float64
 13  target                        178 non-null    int32
dtypes: float64(13), int32(1)
memory usage: 18.9 KB
```

We will use in-built model K-Neighbor Classifier. From the data available, we will split the data into train and test dataset as 80:20 ratio.

```
y = toy_data.target
# split the data using Scikit-Learn's train_test_split
from sklearn.model_selection import train_test_split
#We will split into 80:20 train - test ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

```
# training a KNN classifier

from sklearn.neighbors import KNeighborsClassifier
KNClassifier = KNeighborsClassifier(n_neighbors=5)
KNClassifier.fit(X_train, y_train)
KNClassifier.score(X_test, y_test)
```

```
0.75
```

```
# make predictions on the testing data
y_predict = KNClassifier.predict(X_test)
```

```
# check results
print(confusion_matrix(y_test, y_predict))
print(classification_report(y_test, y_predict))
```

```
[[11  0  0]
 [ 3 12  1]
 [ 1  4  4]]
              precision    recall  f1-score   support

           0       0.73      1.00      0.85        11
           1       0.75      0.75      0.75        16
           2       0.80      0.44      0.57         9

    accuracy                           0.75        36
   macro avg       0.76      0.73      0.72        36
weighted avg       0.76      0.75      0.73        36
```

We will now save the model for future use, using a pickle file.

Save the model using Pickle file

We need to save the model to deploy it and to be able to use it later with some other inputs. We will save our pretrained model using pickle using the following code:

```
import pickle

# save the breast cancer classification model as a pickle file
model_pkl_file = "wine_class_prediction.pkl"

with open(model_pkl_file, 'wb') as file:
    pickle.dump(KNClassifier, file)
```

We will write in binary mode (wb) from the pretrained model called KNClassifier and store it in a pickle file called, “wine_class_prediction.pkl”. The dump() method stores the model in the given pickle file. Now, we will open the file in rb (read binary) mode to load the saved model.
We will now load the model from the pickle file and make predictions. Below is the scores.

```
with open(model_pkl_file, 'rb') as file:
    model = pickle.load(file)

# evaluate model
y_predict = model.predict(X_test)

# check results
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.73      1.00      0.85        11
           1       0.75      0.75      0.75        16
           2       0.80      0.44      0.57         9

    accuracy                           0.75        36
   macro avg       0.76      0.73      0.72        36
weighted avg       0.76      0.75      0.73        36
```

We would test one data from the input data we have, so to see if the model is predicting any right value.

```
# Loading model to compare the results
model = pickle.load(open('wine_class_prediction.pkl','rb'))
print(model.predict([[14.23,2,2.43,15.6,127.0,2.80,3.06,0.28,2.29,5.64,1.04,3.92,1065.0]]))
```

```
[0]
```

Since our model has now been built and it can now predict the wine category. Now, we will convert pretrained Machine Learning model into two forms: one as web application and other one would be API based application. We will then see how then can be deployed on Google cloud (GCP). These can be summarized into following:

- Develop a Web application and deploy it on Google cloud.
- Create an API and deploy it on Google cloud.

Develop a web application with Flask and deploy it on Google cloud:

Before going further, we would create the following files. The code could be found on Github on the webapp link.

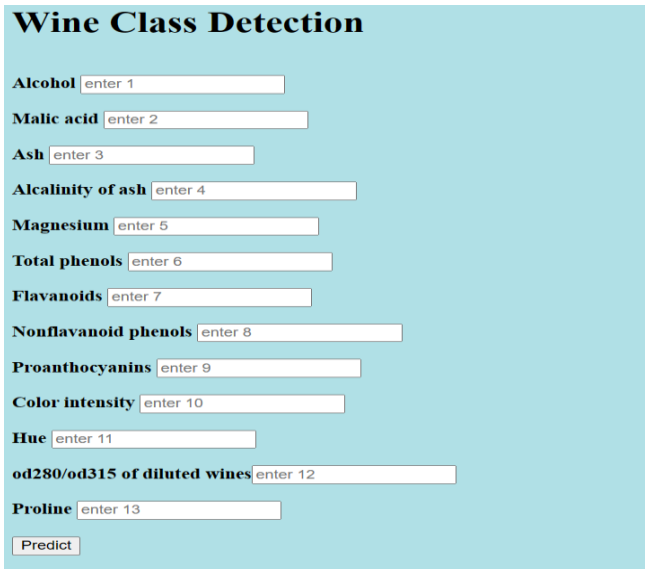
a. An HTML file (home.html)
b. Create a python file (app.py).

a. An HTML file:

We will create a webpage that will ask the user to provide all those 13 input features as input and will display the target, i.e. category of the wine, based on the feature values provided.

```
<!DOCTYPE html>
<html>
<body style="background-color:powderblue">
        <h1>Wine Class Detection</h1><br>
<form action="{{url_for('predict')}}", method="POST">
<b>
        Alcohol <input type="text", name='a', placeholder="enter 1"><br><br>
        Malic acid <input type="text", name='b', placeholder="enter 2"><br><br>
        Ash <input type="text", name='c', placeholder="enter 3"><br><br>
        Alcalinity of ash <input type="text", name='d', placeholder="enter 4"><br><br>
        Magnesium <input type="text", name='e', placeholder="enter 5"><br><br>
        Total phenols <input type="text", name='f', placeholder="enter 6"><br><br>
        Flavanoids <input type="text", name='g', placeholder="enter 7"><br><br>
        Nonflavanoid phenols <input type="text", name='h', placeholder="enter 8"><br><br>
        Proanthocyanins <input type="text", name='i', placeholder="enter 9"><br><br>
        Color intensity <input type="text", name='j', placeholder="enter 10"><br><br>
        Hue <input type="text", name='k', placeholder="enter 11"><br><br>
        od280/od315 of diluted wines<input type="text", name='l', placeholder="enter 12"><br><br>
        Proline  <input type="text", name='m', placeholder="enter 13"><br><br>

        <button type="submit", class="btn">Predict</button><br><br>

</form>
<!--Prediction Result-->
        <div id ="result">
                <strong style="color:red">{{prediction_text}}</strong>
        </div>
</body>
</html>
```

The webpage would look something like this.



b. Create a Python file (app.py)

In this python file, we will define the operations for execution of the model or application we have built. It will load the load pickle file created earlier during the model building stage and then we will run the code on Flask. This file will take all the data entered by the user on the webpage and apply the pretrained classifier on the data and will predict and display the category of the wine on screen.

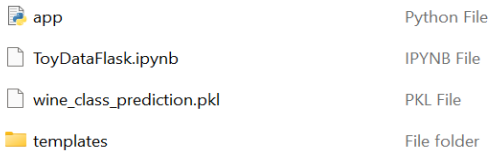
From the code below, we can see we have first installed all the necessary imports. As part of which is to import Flask library, render_template, and request. We then created a Flask instance and assigned it to a variable called app. Also, we created some URL routes using @app.route(), which would correspond to various web pages of our application.

app.run will start the server and will load the application on the web browser.

```
import pickle
import numpy as np
from flask import Flask, render_template, request
from sklearn.neighbors import KNeighborsClassifier

load_classifier = pickle.load(open('wine_class_prediction.pkl', 'rb'))
app = Flask(__name__)

#defining default route
@app.route('/')
def home():
    return render_template('home.html')

@app.route('/predict', methods=['POST'])
def predict():
    data1 = request.form['a']
    data2 = request.form['b']
    data3 = request.form['c']
    data4 = request.form['d']
    data5 = request.form['e']
    data6 = request.form['f']
    data7 = request.form['g']
    data8 = request.form['h']
    data9 = request.form['i']
    data10 = request.form['j']
    data11 = request.form['k']
    data12 = request.form['l']
    data13 = request.form['m']
    arr = np.array([[data1, data2, data3, data4, data5, data6, data7, data8, data9, data10, data11, data12, data13]])
    pred = load_classifier.predict(arr)
    return render_template('home.html', prediction_text='Class of the Wine is:{}'.format(pred))

if __name__ == "__main__":
    app.run(debug=True)
```

Deploy the model as Web Application.

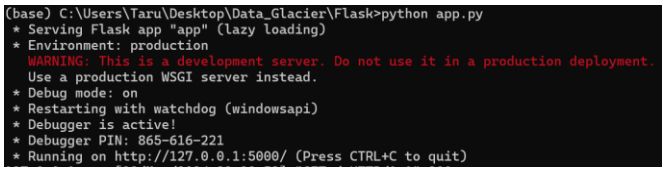
We would first create a folder structure. We need to make sure we have a separate directory for all the files related to the current project and will move or delete the files non-relevant to the project. We need to create a templates folder inside the working directory as we need to store the html file inside the templates folder. The working directory would now contain a pickle file (for the pretrained model), a python file (in this case, app.py) and a templates directory.



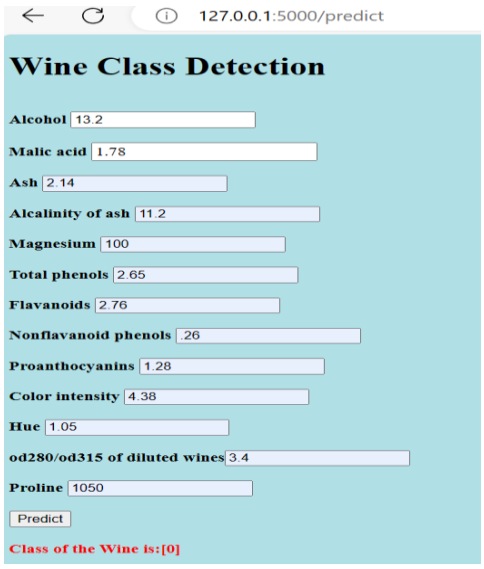
We will install flask using the command **pip install flask**

Being a Windows user, as I have anaconda installed on my computer, I used Anaconda shell to run and deploy the application on Flask. On the shell, we need to go to the directory where the Python file is present using “cd <path>”. Then, we run the command python <filename.py> (“python app.py”) on shell.

The output is as shown below:

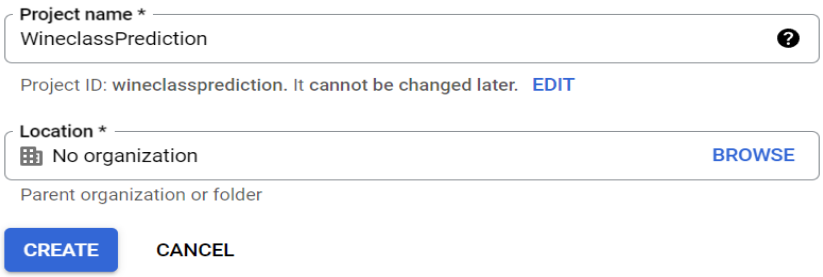


When we clicked on the url http://127.0.0.1:5000/, it displayed the webpage we created using HTML file. When we entered the values, it predicted the wine category as shown below:

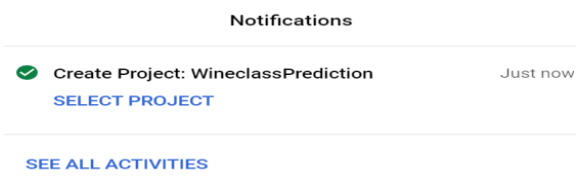


Deployment on Google Cloud

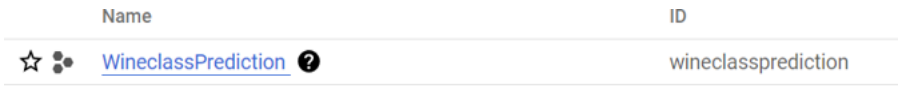
We will go to the URL for Google Cloud Platform (GCP) and then sign up. Then, will Create a NEW PROJECT.



The name of my project is ***WineclassPrediction***.

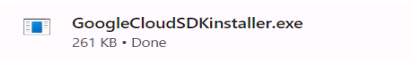


Then, I will select the project as below:



The next step would be to rename my app.py file to main.py as GCP only accepts main.py.

Then I created app.yaml file. Then we will install Google Cloud SDK installer from the link. We will then run this .exe file.

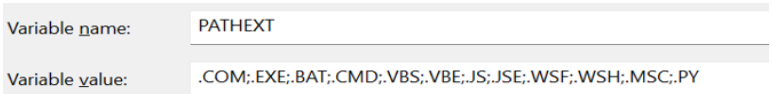


Go to the location of the folder with all the necessary files for WebApp development.

```
C:\Users\Taru>cd C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\WebApp
```

Set up environment variable for Google Cloud SDK installer:

- Under System variables in Environment variables, on PATHEXT, add ';.PY'

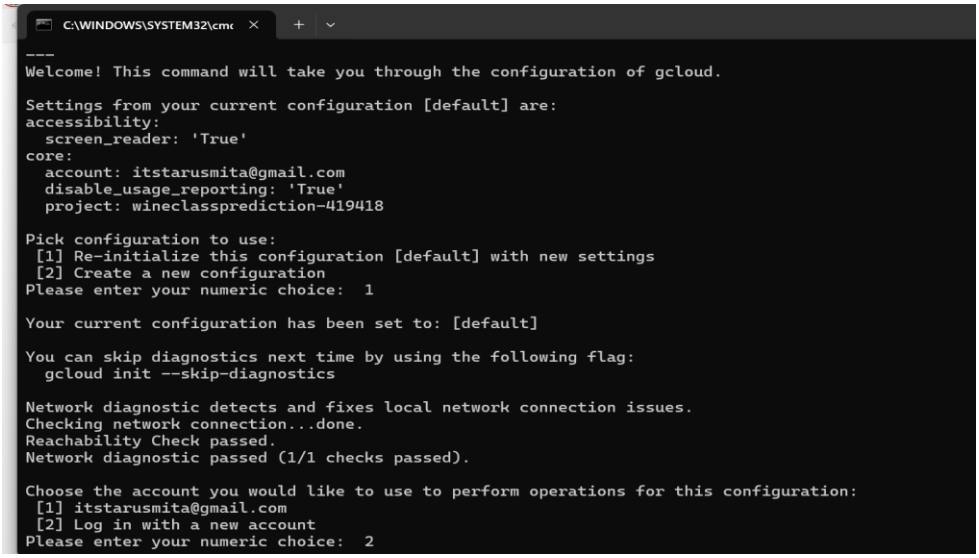


- Under System variables in Environment variables, on PATH, add "C:\Program Files\Google\Cloud SDK\google-cloud-sdk\bin".

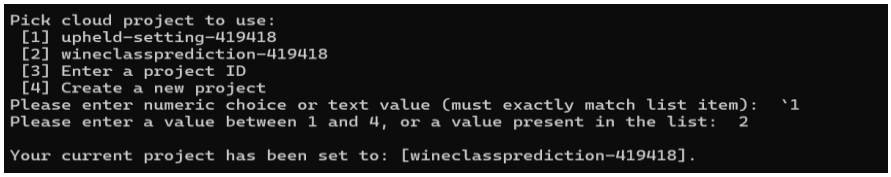
Now, restart the command shell and type the following:

gcloud init

The following appears on the screen. Choose the appropriate option.



Choose the cloud project we created as below:



Go to the path of the code directory and type the following:

gcloud app deploy app.yaml --project wineclassprediction-419418

wineclassprediction-419418 is the project ID of my Google cloud project.

```
C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\WebApp>gcloud app deploy app.yaml --project wineclassprediction-419418
You are creating an app for project [wineclassprediction-419418].
WARNING: Creating an App Engine application for a project is irreversible and the region
cannot be changed. More information about regions is at
<https://cloud.google.com/appengine/docs/locations>.

Please choose the region where you want your App Engine application located:

 [1] asia-east1    (supports standard and flexible)
 [2] asia-east2    (supports standard and flexible and search_api)
 [3] asia-northeast1 (supports standard and flexible and search_api)
 [4] asia-northeast2 (supports standard and flexible and search_api)
 [5] asia-northeast3 (supports standard and flexible and search_api)
 [6] asia-south1   (supports standard and flexible and search_api)
 [7] asia-southeast1 (supports standard and flexible)
 [8] asia-southeast2 (supports standard and flexible and search_api)
 [9] australia-southeast1 (supports standard and flexible and search_api)
 [10] europe-central2 (supports standard and flexible)
 [11] europe-west   (supports standard and flexible and search_api)
 [12] europe-west2  (supports standard and flexible and search_api)
 [13] europe-west3  (supports standard and flexible and search_api)
 [14] europe-west6  (supports standard and flexible and search_api)
 [15] northamerica-northeast1 (supports standard and flexible and search_api)
 [16] southamerica-east1 (supports standard and flexible and search_api)
 [17] us-central    (supports standard and flexible and search_api)
 [18] us-east1      (supports standard and flexible and search_api)
 [19] us-east4      (supports standard and flexible and search_api)
 [20] us-west1      (supports standard and flexible)
 [21] us-west2      (supports standard and flexible and search_api)
 [22] us-west3      (supports standard and flexible and search_api)
 [23] us-west4      (supports standard and flexible and search_api)
 [24] cancel
Please enter your numeric choice:  12
```

Choose the region. I chose 12, which is Europe-west2 for London. The following appears on the screen:

```
Creating App Engine application in project [wineclassprediction-419418] and region [europe-west2]....done.
Services to deploy:

descriptor:                  [C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\WebApp\app.yaml]
source:                      [C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\WebApp]
target project:              [wineclassprediction-419418]
target service:              [default]
target version:              [20240405t211131]
target url:                  [https://wineclassprediction-419418.nw.r.appspot.com]
target service account:      [wineclassprediction-419418@appspot.gserviceaccount.com]


Do you want to continue (Y/n)?  Y
```

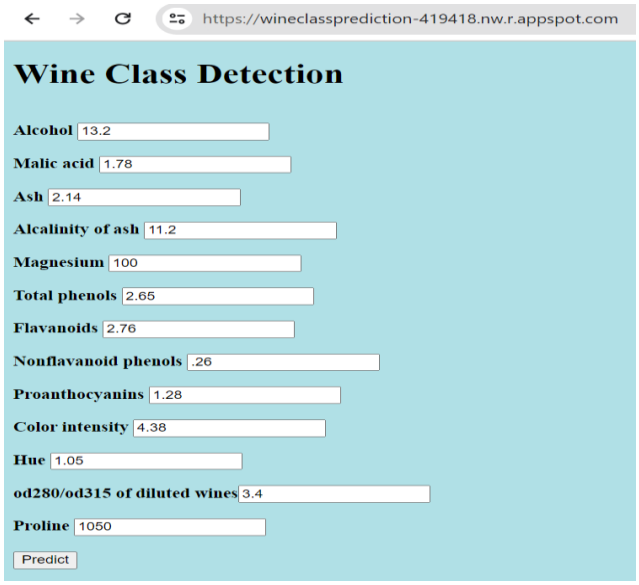
We can see the progress now:

```
Do you want to continue (Y/n)?  Y

Beginning deployment of service [default]...
Uploading 2 files to Google Cloud Storage
50%
100%
100%
File upload done.
Updating service [default]...done.
Setting traffic split for service [default]...done.
Deployed service [default] to [https://wineclassprediction-419418.nw.r.appspot.com]

You can stream logs from the command line by running:
  $ gcloud app logs tail -s default

To view your application in the web browser run:
  $ gcloud app browse
```

We will now go to the URL on screen:

```
https://wineclassprediction-419418.nw.r.appspot.com
```

When we opened the browser using the URL, it displayed the webpage I created.



We will test it using our training data, to see if the web application can predict wine class as 0.

```
# Add a target column, and fill it with the target data
df_toydata['target'] = toy_data.target
# Show the first five rows
df_toydata.head()
```

_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target
15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	0
11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	0

Now, we will create the same application on FastAPI and deploy it on Google Cloud.

Develop an API based application using FastAPI and deploy it on Google cloud Platform (GCP):

We need to create some files now.

- ***main.py*** – in this file, we have all the operations. We will also create a app object from FastAPI(). We will use the same pickle file created earlier for our pretrained model.

  ```
  #Create the app object
  app = FastAPI()
  wine_classifier = pickle.load(open('wine_class_prediction.pkl', 'rb'))
  ```

  We will create two API's here:
  One will display welcome message as below:

  ```
  @app.get('/')
  def index():
      return {'message':'Welcome to the Wine Prediction site'}
  ```

  Another one will take user's input for the features and will post the prediction as wine type:

```python
#Prediction functionality
@app.post('/predict')
def predictWineClass(data:WineFeature):
    data = data.dict()
    alcohol = data['alcohol']
    malic_acid = data['malic_acid']
    ash = data['ash']
    alcalinity_of_ash = data['alcalinity_of_ash']
    magnesium = data['magnesium']
    total_phenols = data['total_phenols']
    flavanoids = data['flavanoids']
    nonflavanoid_phenols = data['nonflavanoid_phenols']
    proanthocyanins = data['proanthocyanins']
    color_intensity = data['color_intensity']
    hue = data['hue']
    od280_od315_of_diluted_wines = data['od280_od315_of_diluted_wines']
    proline = data['proline']
    arr = np.array([[alcohol, malic_acid, ash, alcalinity_of_ash, magnesium, total_phenols, flavanoids, no
    pred = wine_classifier.predict(arr)
    #return {'Class of the Wine is:{}'.format(pred)}
    return 'Class of the Wine is:{}'.format(pred)
```

Then we will run the API with ***uvicorn***. API will run on http://127.0.0.1:8000, which we have defined on our main.py file.

```python
#Run the API with uvicorn
#API will run on http://127.0.0.1:8000
if __name__ == "__main__":
    uvicorn.run(app, host='127.0.0.1', port=8000)
#uvicorn <app/filename>:<app/object_name> --reload
```

- ***winefeatures.py*** – this file will contain a class file with all the input features and their datatype. For this we will import BaseModel from pydantic. Pydantic helps friendly errors and ensures user keys in the correct type of inputs. The contents of the file is as below:

```python
from pydantic import BaseModel

#class which will describe Wine features.

class WineFeature(BaseModel):
    alcohol : float
    malic_acid : float
    ash : float
    alcalinity_of_ash : float
    magnesium : float
    total_phenols : float
    flavanoids : float
    nonflavanoid_phenols : float
    proanthocyanins : float
    color_intensity : float
    hue : float
    od280_od315_of_diluted_wines : float
    proline : float
```

- ***Dockerfile*** and ***docker-compose.yaml***: Thess files will help us create a docker image which we need to deploy in the Google cloud platform.

Dockerfile

```
FROM python:3.10-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir --upgrade -r requirements.txt

CMD uvicorn main:app --reload --port=8000 --host=0.0.0.0
```

docker-compose.yaml

```yaml
version: '3'

services:
  web:
    build: .
    command: sh -c "uvicorn main:app --reload --port=8000 --host=0.0.0.0"
    ports:
      - 8000:8000
    volumes:
      - .:/app
```

Let's try to deploy the code and test it locally.

Deployment:

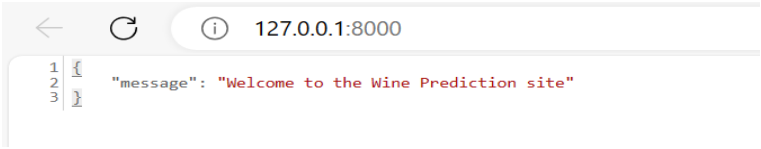
We will install fastapi uvicorn and run the code as below:

```
pip install fastapi uvicorn
```

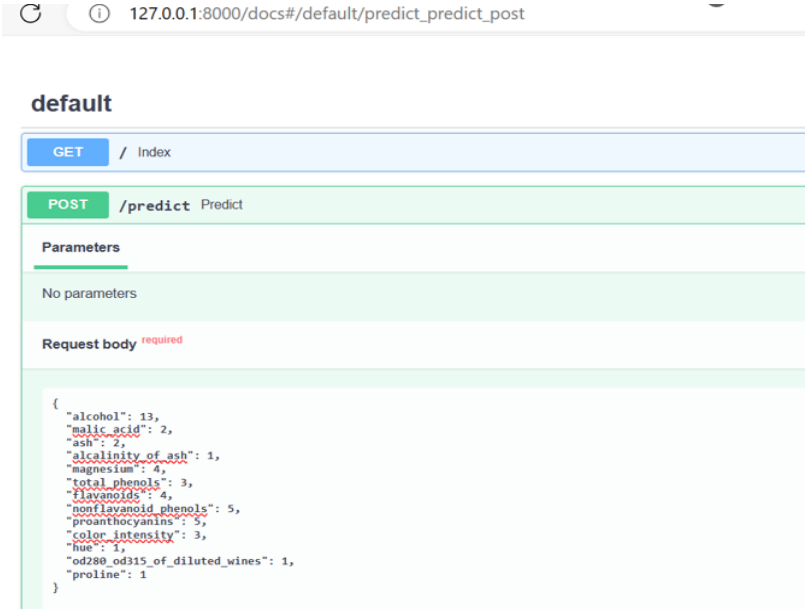
Then, we need to go the directory of the code base and run the code below:

```
(base) C:\Users\Taru>cd Desktop\Data_Glacier\CloudDeployment

(base) C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment>uvicorn app:app --reload
```

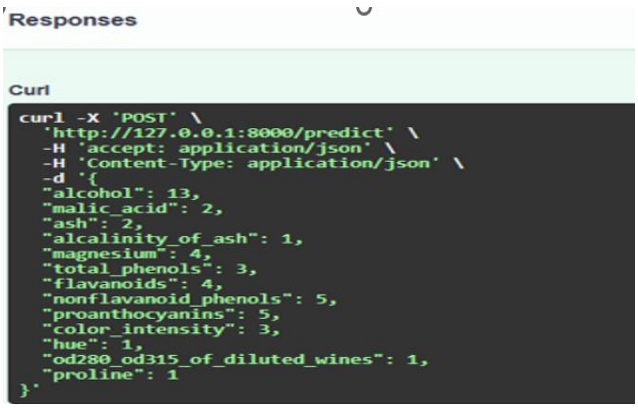
The following is our first API output.



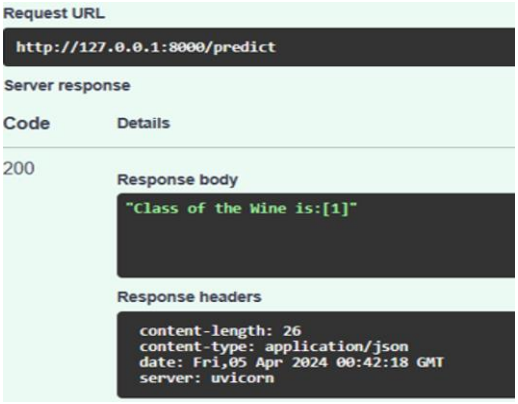
Then we will go to the URL, http:127.0.0.1:8000/docs, the following screen appears.



We will then try our post API and enter input feature values and execute, the following output appears.



This predicts the class of wine in json form as shown below:



Create virtual machine as venv. We will see a new folder in our working directory as **venv**. We will go to the **venv** directory and then type "***.\Scripts\activate***" to activate the virtual machine on windows computer as shown below:

```
\venv> .\Scripts\activate
```

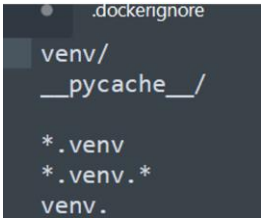
After installing all the necessary libraries, they can be viewed using pip list. We will now use the command below, to list the dependencies of our project. We will save those dependencies into **requirements.txt**.

```
pip freeze > requirements.txt
```

Let's understand, what's in our **Dockerfile**. "app" is our work directory, then we will copy everything from our current directory to the work directory app. Install all the dependencies into requirements.txt.

```
FROM python3.10-slim

WORKDIR /app

COPY . /app

RUN pip install -r requirements.txt
```

We will also create docker ignore file **".dockerignore**". The reason to create this file is that we do not want our virtual environment **venv** to be deployed anywhere like Github or GCP.



We have already discussed earlier about another docker file "docker-compose.yaml" .

We will also install Docker desktop for Windows and run the container.

We will now test it on our development environment, if it works.

We will add the following into environment variable.

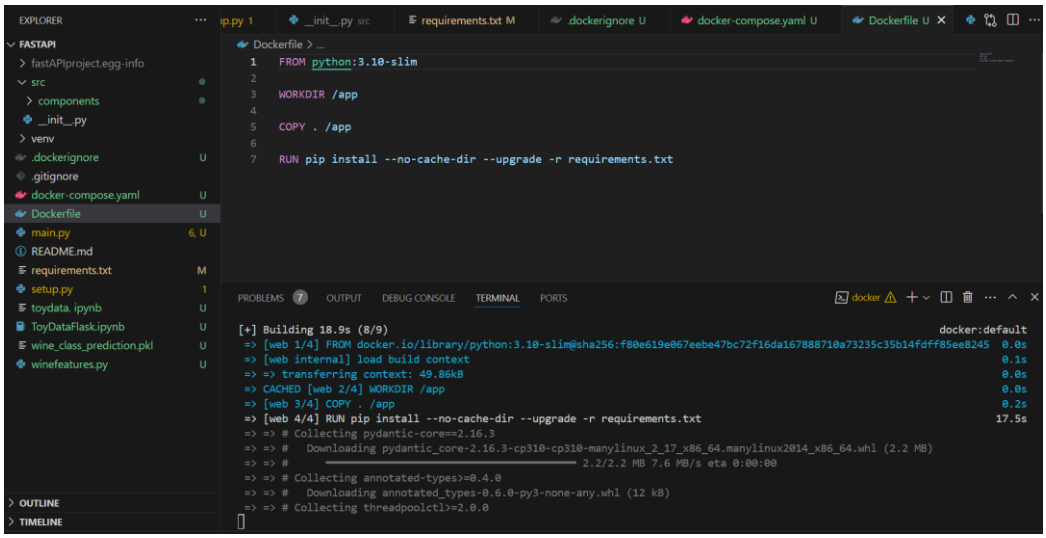
C:\Program Files\Docker\Docker\resources\bin

C:\ProgramData\DockerDesktop\version-bin

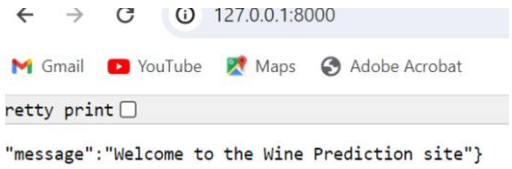
Now, we will see if it works.

```
PS C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\FastAPI> docker compose up --build
```

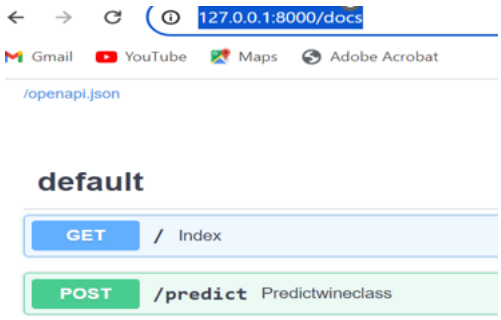
We can see our build progress.



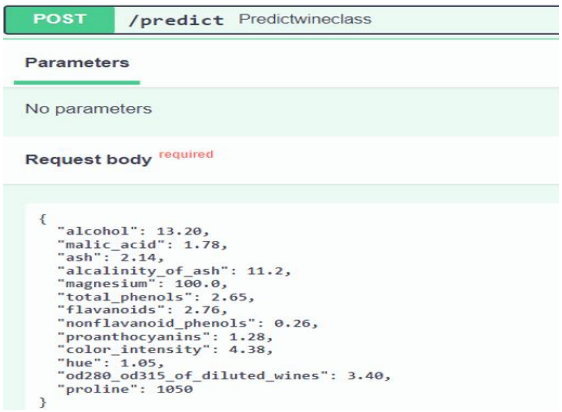
We will deploy that as API now. On the link http:127.0.0.1:8000, the following window appears.



Now, we will check on the URL: http://127.0.0.1:8000/docs



We will try it out and execute and it predicts the wine class.



Since, our project is working perfectly, we will prepare everything on GCP's side. We will first need a docker repository.

Create a Docker repository.

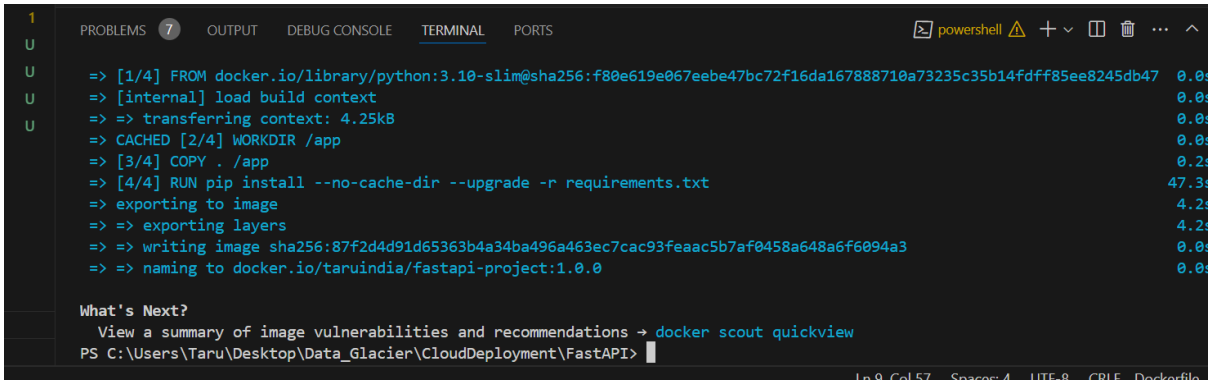
We will login to docker hub. Click on repository and then create a repository. I created the following:

Now, back to our shell, we will type the following command:

```
PS C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\FastAPI> docker build -t taruindia/fastapi-project:1.0.0 .
```

We can give any tag (in this case, it is 1.0.0) after **taruindia/fastapi-project:<tagname>.**

Our docker image is built now.



We will now type following command to authenticate our credentials and will get success message.

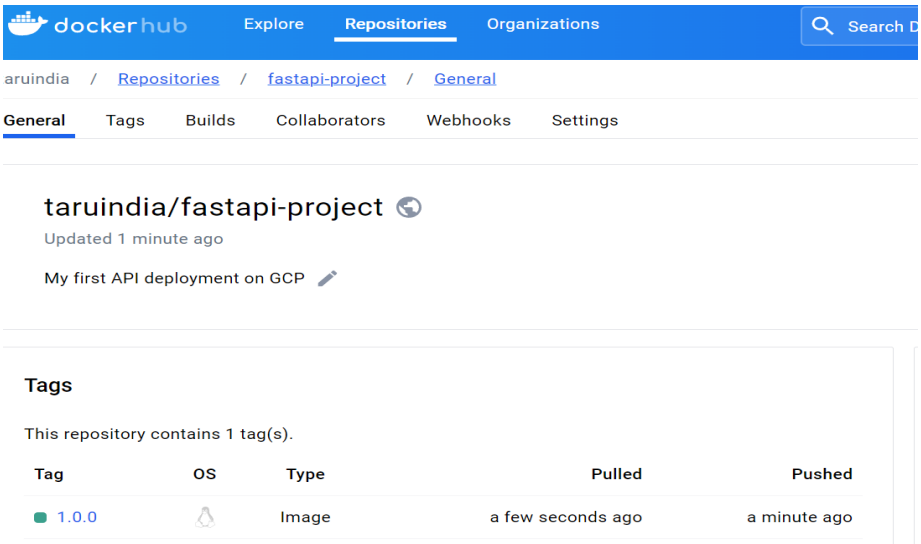
```
PS C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\FastAPI> docker login
Authenticating with existing credentials...
Login Succeeded
```

Now, we need to **push the image to docker hub**. We will use the command below:

docker push taruindia/fastapi-project:1.0.0

```
PS C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\FastAPI> docker push taruindia/fastapi-project:1.0.0
The push refers to repository [docker.io/taruindia/fastapi-project]
8e62dcb9f4f3: Pushed
7f0c5c4975b4: Pushed
4600495fb6ce: Pushed
9362be696331: Mounted from library/python
23cef6a8f8d1: Mounted from library/python
884af60a304e: Mounted from library/python
c8f253aef560: Mounted from library/python
a483da8ab3e9: Mounted from library/python
1.0.0: digest: sha256:3e994e8a74ec4f15c9a8dc55e8a33fe36d4047b57efa1704ddb2ff735714184c size: 1998
PS C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\FastAPI>
```

Once the image is pushed, we can refresh the repository in docker hub and we can now see the image.



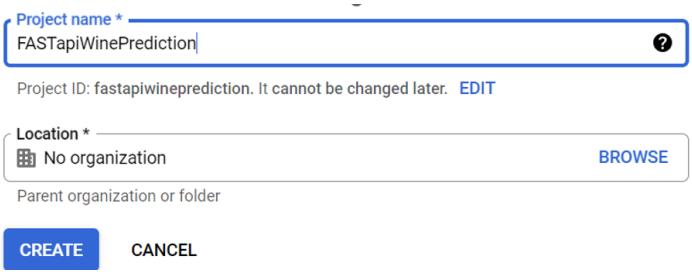
Deactivate the virtual machine.

We will now de-activate the virtual machine using **deactivate** command**.**

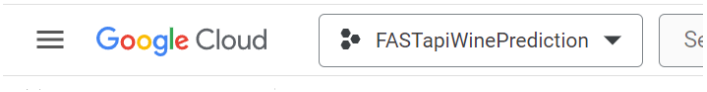
```
canceled
PS C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\FastAPI> deactivate
```

Create a new project on GCP.

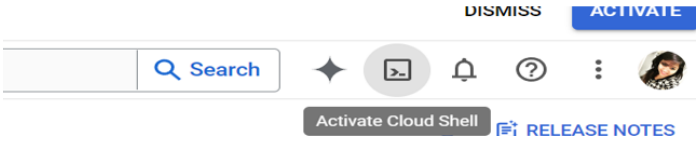
Now, we will open the browser and login to Google Cloud Platform. Then we will create a project. Name of my project is FASTapiWinePrediction



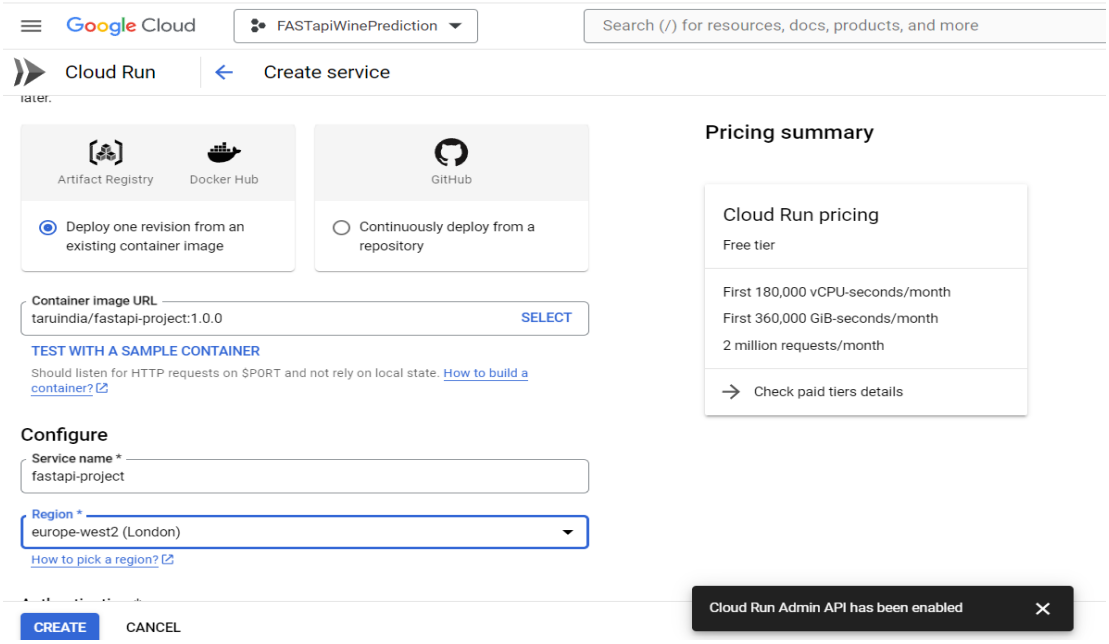
Once the project created, we will click on cloudrun and select the newly created project.



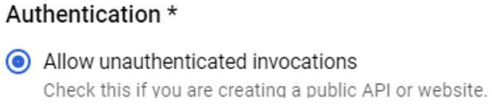
We will now click on activate cloud shell as below:

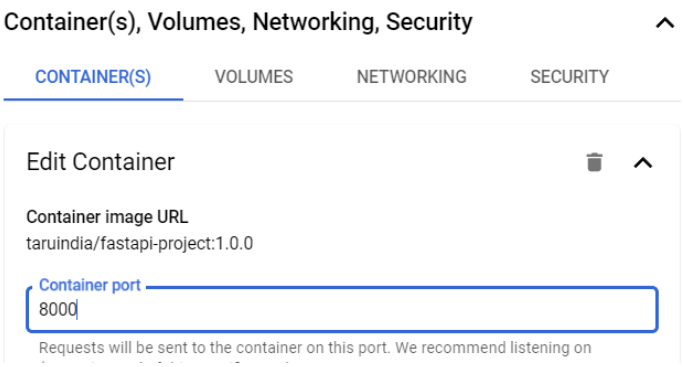


Then, we will navigate to Cloud Run and will Create Service.

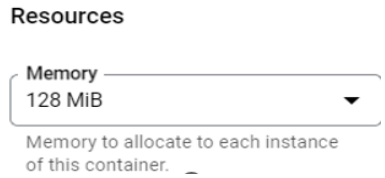


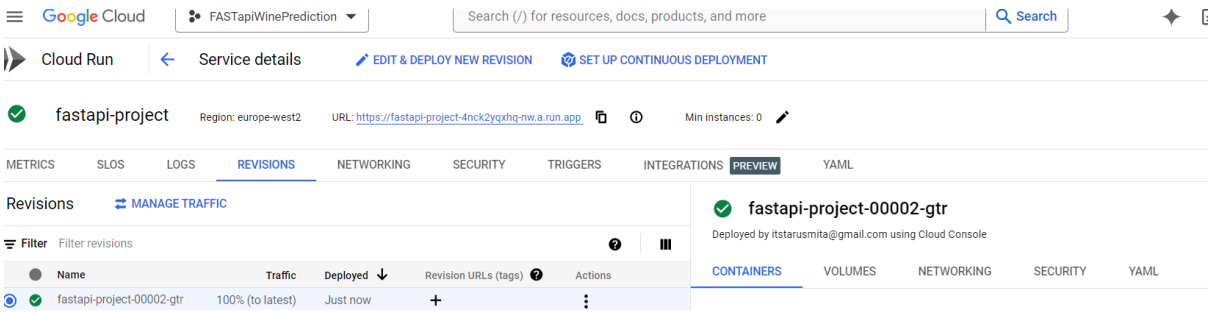
We will click on Allow unauthenticated invocations as below and type container port: 8000.



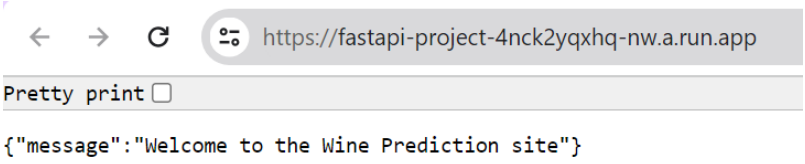


Resources memory, I will choose anything more than 128 MiB and then will click on create.

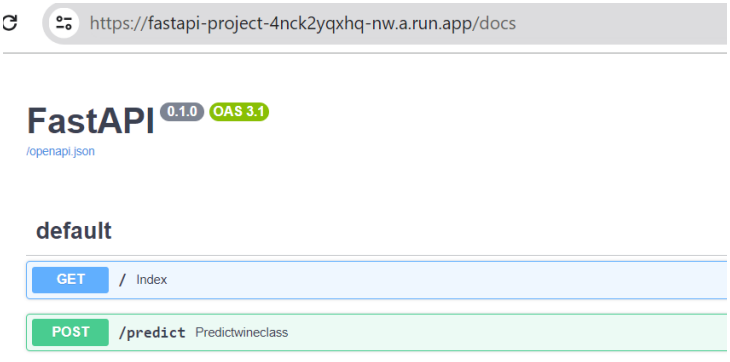




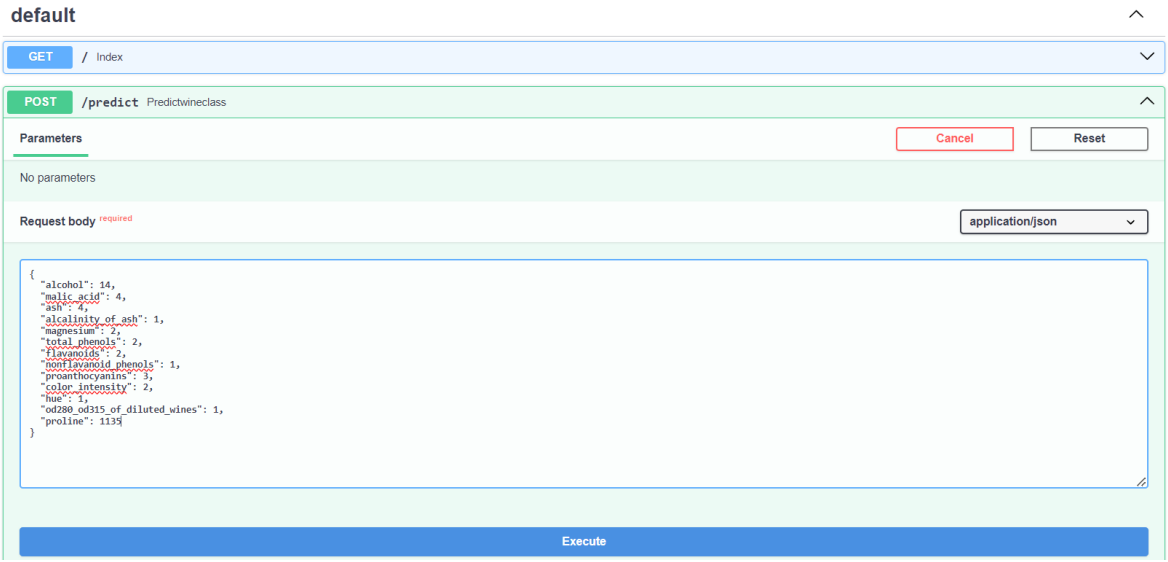
We can see that the deployment is now successful, and it has given us a URL to run our API online. We will now click on the URL.



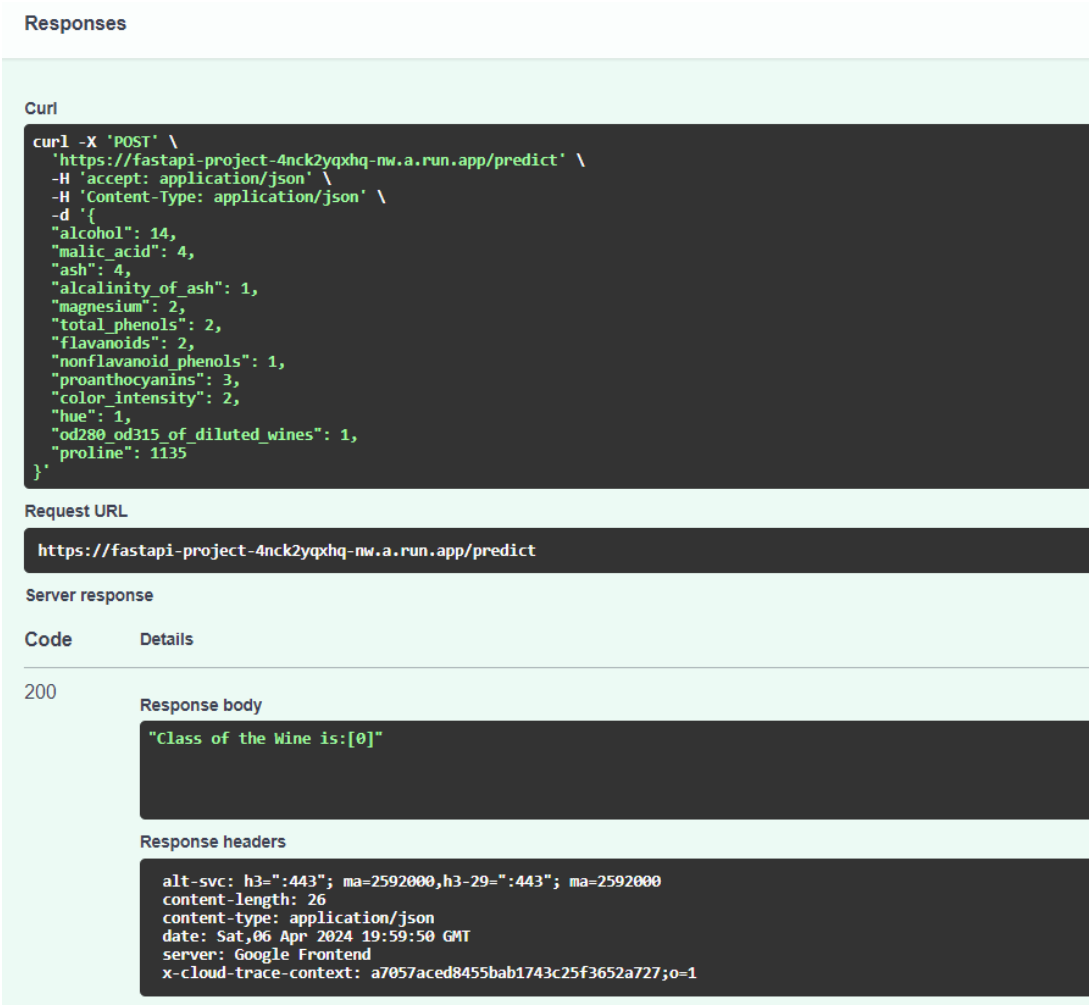
With the same URL, we will now check /docs as below:



We will try it out and execute now.



We got the expected response as shown below:



We have now successfully deployed both API based and Web based applications on Google Cloud Platform (GCP). We will now deploy our code on Github.

Deployment on Github:

The entire code for this implementation have been uploaded on Github on the github-link.

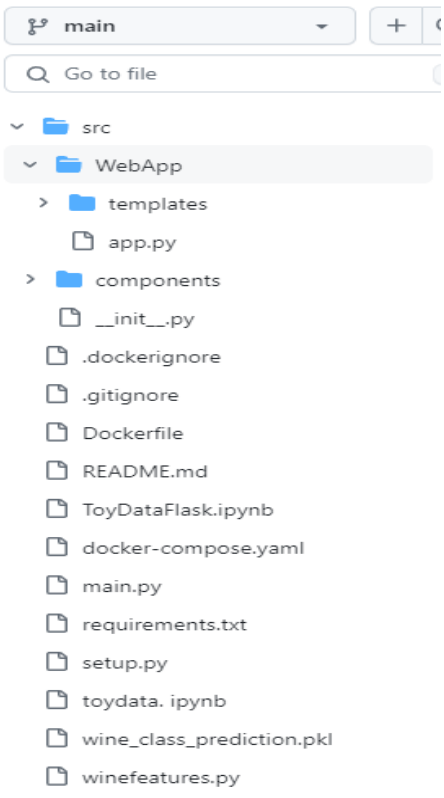
We will activate our environment variable again. Add, commit and push the changes to Github.

```
conda activate venv/
git add .
```

```
PS C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\FlaskWebApp> conda activate venv/
PS C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\FlaskWebApp> git init
Initialized empty Git repository in C:/Users/Taru/Desktop/Data_Glacier/CloudDeployment/FlaskWebApp/.git/
PS C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\FlaskWebApp> git branch -M mlwebapp
PS C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\FlaskWebApp> git remote add origin https://github.com/TaruIndia/CloudDeployment.git
PS C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\FlaskWebApp> git remote -v
origin  https://github.com/TaruIndia/CloudDeployment.git (fetch)
origin  https://github.com/TaruIndia/CloudDeployment.git (push)
PS C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\FlaskWebApp> git add .
```

```
PS C:\Users\Taru\Desktop\Data_Glacier\CloudDeployment\FlaskWebApp> git commit -m "My first commit on this branch"
[mlwebapp (root-commit) 8186cb2] My first commit on this branch
 9 files changed, 1030 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 README.md
 create mode 100644 ToyDataFlask.ipynb
 create mode 100644 main.py
 create mode 100644 requirements.txt
 create mode 100644 setup.py
 create mode 100644 src/__init__.py
 create mode 100644 templates/home.html
 create mode 100644 wine_class_prediction.pkl
```

The folder structure of my code on Github is as below:



Note:

All the files outside the folder WebApp are mainly for API based deployment on GCP and common files. However, the ones inside the folder WebApp are for Web application-based deployment on GCP and some common files share by both the applications are outside this folder. Though, GCP takes main.py as python file, I had to change the one inside WebApp folder to app.py as there was a conflict as both the applications had the same files name main.py and I wanted to store the code base in one location for both the applications.