WEEK 5: CLOUD AND API DEPLOYMENT

Name: Thrinesh Duvvuru

Internship Batch: LISUM01

Submission date: 11.07.2021

FLASK:

Flask is a **web application framework written in python**, which helps end users interact with our python code (in this case our ML models) directly from their web browser without needing any libraries, code files, etc.

Files required for deployment of ML model on Flask:

1. **templates:** - This folder contains the html files (index.html, predict.html) that would be used by our main file (app.py) to generate the front end of our application
2. **app.py:** - This is the main application file, where all our code resides and it binds everything together.
3. **requirements.txt:** - This file contains all the dependencies/libraries that would be used in the project (whenever a virtual environment is created it can use this requirements file directly to download all the dependencies you need not to install all the libraries manually, you just need to put all of them in this file)
4. **model.pkl:** - This is pre trained model, that can be used directly on flask without training the model again.

Installing Dependencies:

The following commands are used to install **Flask** and **requirements.txt**

```
$ python3 -m pip install Flask==1.1.2
```

```
$ python3 -m pip freeze > requirements.txt
```

Procedure:

I have taken a dataset from Kaggle (car_price_prediction), built a model and saved it in the form of .pkl.

Step 1: Import all necessary libraries.

```
from flask import Flask,request,render_template
import jsonify
import requests
import pickle
import sklearn
import numpy as np
```

Step 2: Creating **app** object

```
app=Flask(__name__)
```

Just like any other python class object, at the end this app is nothing but just an object of class Flask which would do all the heavy lifting for us, like handling the incoming requests from the browser (in our case the question that user enters) and providing with appropriate responses (in this case our model prediction) in some nice form using html,css.

Step 3:

```
model = pickle.load(open('car_price.pkl', 'rb'))

@app.route('/',methods=["GET"])

def home():
    return render_template('index.html')

@app.route('/predict',methods=['POST'])
```

Load the pkl file, in my case “car_price.pkl”.

The line @app.route (‘/’,methods=[“GET”]) maps the method to the URL mentioned inside the decorator, i.e whenever user visits that URL ‘/’ (the complete address would have an ip address and a port number as well, for example http://127.0.0.1:5000/), index() method would be called automatically, and the index() method returns our main HTML page called index.html (in our case index.html provides a text box to the user where he could enter his question)

The flask.render_template() looks for the this index.html file in the templates folder that we created in our main directory and dynamically generates/renders a HTML page for the end user.

Now we have another decorator @app.route (‘/predict’), this one maps the predict() method with the /predict URL , this method takes the input given by the user, does all the pre-processing, generates the final feature vector, runs the model on it and gets the final prediction.

Step 4: Creating predict function

```
def predict():
    if request.method=='POST':
        Year = int(request.form['Year'])
        car_age=2020-Year
        Present_Price=float(request.form['Present_Price'])
        Kms_Driven=int(request.form['Kms_Driven'])
        Owner=int(request.form['Owner'])
        Fuel_Type_Petrol=request.form['Fuel_Type_Petrol']
        if(Fuel_Type_Petrol=='Petrol'):
                Fuel_Type_Petrol=1
                Fuel_Type_Diesel=0
        elif(Fuel_Type_Petrol=='Diesel'):
            Fuel_Type_Petrol=0
            Fuel_Type_Diesel=1
        else:
             Fuel_Type_Petrol=0
             Fuel_Type_Diesel=0
        Seller_Type_Individual=request.form['Seller_Type_Individual']
        if(Seller_Type_Individual=='Individual'):
            Seller_Type_Individual=1
        else:
            Seller_Type_Individual=0
        Transmission_Mannual=request.form['Transmission_Mannual']
        if(Transmission_Mannual=='Mannual'):
            Transmission_Mannual=1
        else:
            Transmission_Mannual=0
        prediction=model.predict([[Present_Price,Kms_Driven,Owner,car_age,Fuel_Type_Diesel,Fuel_Type_Petrol,
                                   Seller_Type_Individual,Transmission_Mannual]])
        output=round(prediction[0],2)
        if output<0:
            return render_template('index.html',prediction_texts="please check the data again")
        else:
            return render_template('index.html',prediction_text="approximate selling price is Rs {} Lakhs".format(output))
    else:
        return render_template('index.html')

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

Now let's have a quick look at the predict() method in our main code, **request.form** it takes the user entered text, when clicked on the submit button in our form a json is returned containing key value pairs having responses, when the submit button is clicked this /predcit URL is called and responses are sent to the page corresponding to this URL as a json and we had seen it earlier as well, once this /predict URL is called, the associated predict() method is invoked automatically which takes this json and the output template (predict.html) and puts it together to generate our final output.

Step 5:

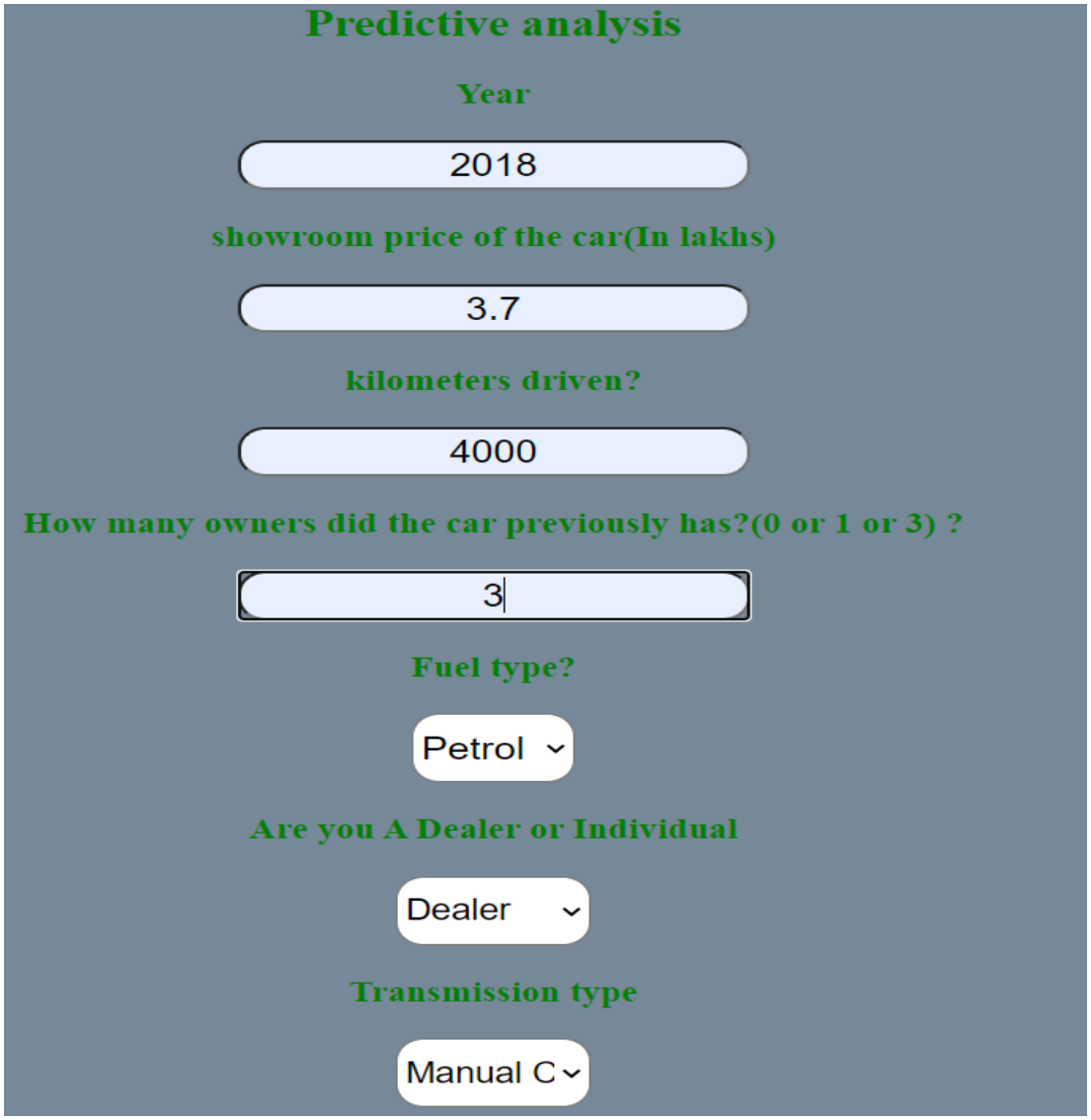
Run app.py on command prompt and copy the url (http://127.0.0.1.5000/) and paste it on the browser of local desktop and run it.

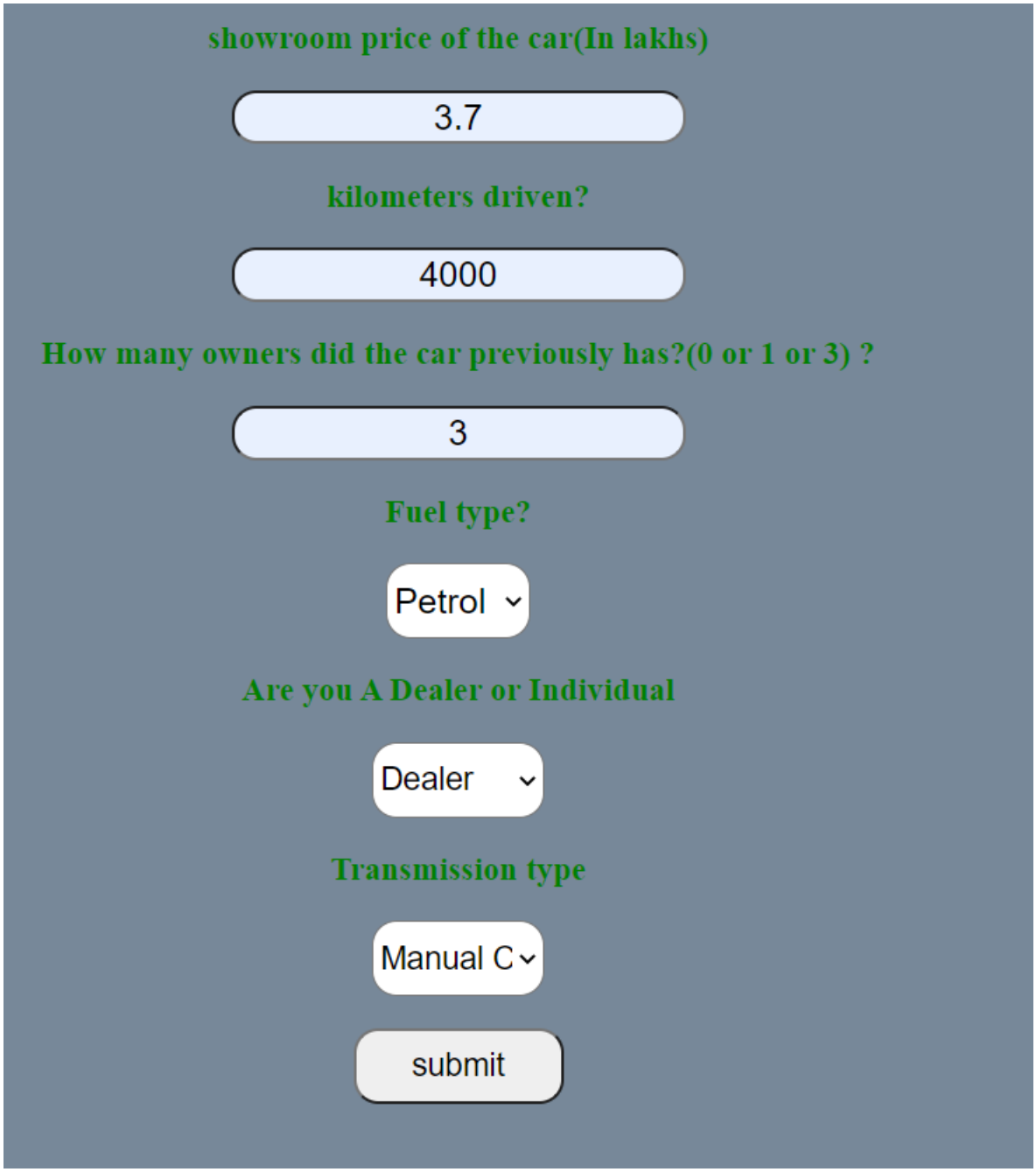
```
(base) C:\Users\Thrinesh Duvvuru>cd C:\Users\Thrinesh Duvvuru\Documents\My_projects\car_price

(base) C:\Users\Thrinesh Duvvuru\Documents\My_projects\car_price>app.py
 * Serving Flask app "app" (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
 * Restarting with windowsapi reloader
 * Debugger is active!
 * Debugger PIN: 270-220-563
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Step 6:

After running the url you will see something like this on the webpage, enter all the required fields and hit submit button.





EXAMPLE OUTPUT:

Finally, our ML model is successfully deployed using flask.

Deployment on Heroku:

So far, we have deployed our ML model on our local desktop, now we will deploy our model on cloud platform (Heroku).

Requirements:

Procfile: This file guides the Heroku to execute the first file.

web: gunicorn app:app, this is the syntax of file and "app:app" is name of file:name of flask app. In our case both are "app"

Requirements.txt: This the file we already created before; we should make sure that all libraries are mentioned.

Create GitHub repository:

Next step is to create GitHub repository and upload following files:

1. Templates
2. App.py
3. Model.pkl
4. Procfile
5. Requirements.txt

Deployment on Heroku:

1. Next step is to create an account on Heroku.
2. Login to the account and create a new app.
3. Then click on deployment method using GitHub.
4. Connect to GitHub and enter the repository name that we created for deployment.
5. Then click on "enable automatic deployment".
6. Click on "Open app".

App url: heroku app