

DATA GLACIER INTERNSHIP

WEEK 4: CLOUD AND API DEPLOYMENT

Name: Nrusimha Saraswati Sai Teja Jampani

Batch: LISUM27

Submission Date: Dec-03-2023

Submitted To: Data Glacier

Abstract:

We have developed an ML application to predict the price of used cars across the US based on the location, year and number of miles run by the car. The model is then deployed on a cloud-based platform through flask API. In our case, we will be using Google Cloud Platform to host our model. As a part of this task, appropriate files for the model, pickle, requirements, HTML, CSS and Flask application will be created.

Dataset:

The dataset we have utilized is 'used car listing' available on Kaggle. The data is preprocessed by removing unwanted columns and the final dataset contains 5 predictors and a target column as given below.

Column	Description
Year	The year in which the car is manufactured
Mileage	The number of miles run by the car
City	The city to which the car belongs
State	The State to which the car belongs

Make	The Manufacturer of the car
Price	The Price of the car

The data intake report of the dataset is given below.

Tabular data details: Used_Cars

Total number of observations	852122
Total number of files	1
Total number of features	5
Base format of the file	csv
Size of the data	33 MB

Model:

We will be fitting a linear regression model to predict the price of the used car by using the other fields i.e., Year, Mileage, City, State and Make as predictor variables. Categorical variables City, State and Make are converted to numeric using label encoder from scikit learn.

```
le = LabelEncoder()
city_label = le.fit_transform(used_cars['City'])
state_label = le.fit_transform(used_cars['State'])
make_label = le.fit_transform(used_cars['Make'])

used_cars.drop("City", axis=1, inplace=True)
used_cars.drop("State", axis=1, inplace=True)
used_cars.drop("Make", axis=1, inplace=True)

used_cars['City'] = city_label
used_cars['State'] = state_label
used_cars['Make'] = make_label
```

```
X = used_cars.drop(columns = ['Price'])
y = used_cars['Price']
model = LinearRegression().fit(X, y)
```

We will then save our model as well as the label encoded data as a pickle file to be used in our flask program.

```
model = pickle.dump( model, open( "model.p", 'wb' ) )
```

```
city_label_encoder=pickle.dump( city_label, open( "city_label.p", 'wb' ) )
```

```
state_label_encoder=pickle.dump( state_label, open( "state_label.p", 'wb' ) )
```

```
make_label_encoder=pickle.dump( make_label, open( "make_label.p", 'wb' ) )
```

Deployment on Google Cloud Platform:

Initially, we will be creating a HTML script with 5 text boxes for inputting the predictor values i.e., Year, Mileage, State, City and Maker. We also use a submit button to output the prediction. A text field will be dynamically filled from the flask program to output the prediction value.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Input Form</title>
  <link rel="stylesheet" href="{{ url_for('static', filename = 'css/styles.css')}}">
</head>
<body>
  <div class = 'login'>
    <div class="header">
      
      <h1>Used Car Price Predictor</h1>
    </div>
    <form action="/predict" method="post">
      <label for="Year">Year:</label>
      <input type="text" id="Year" name="Year" required><br>

      <label for="Milesrun">Milesrun:</label>
      <input type="text" id="Milesrun" name="Milesrun" required><br>

      <label for="City">City:</label>
      <input type="text" id="City" name="City" required><br>

      <label for="State">State:</label>
      <input type="text" id="State" name="State" required><br>

      <label for="Maker">Maker:</label>
      <input type="text" id="Maker" name="Maker" required><br>

      <button type="submit">Predict Price</button>
    </form>
    <br>
    <br>
    <div id = 'pred_text'>{{ pred_text }}</div>
  </div>
</body>
</html>
```

We have also used a CSS script to further format our deployment front end application. I have added an image and enhanced the aesthetics. The script is as shown below. HTML script is placed in templates folder and CSS script is placed in static folder respectively.

```
body {
    margin: 0;
    padding: 0;
    font-family: 'Arial', sans-serif;
    background-color: #f4f4f4;
}

.login {
    max-width: 400px;
    margin: 50px auto;
    padding: 20px;
    background-color: #fff;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

.header {
    text-align: center;
    margin-bottom: 20px;
}

.car-photo {
    max-width: 100%;
    height: auto;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

form {
    text-align: center;
}

label {
    display: block;
    margin-bottom: 8px;
}
```

```

input {
    width: calc(100% - 16px);
    padding: 8px;
    margin-bottom: 16px;
    box-sizing: border-box;
    border: 1px solid #ddd;
    border-radius: 4px;
}

button {
    background-color: #4CAF50;
    color: white;
    padding: 10px 20px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}

button:hover {
    background-color: #45a049;
}

#pred_text {
    margin-top: 20px;
    text-align: center;
    font-weight: bold;
    color: #333;
}

```

We then create a flask API application to render the ML model on the cloud platform. For hosting the application on Google Cloud Platform, we name this file as main.py instead of app.py as done previously. In the logic part, we load the necessary pickle files and using the label encoder objects, input form data is converted from categorical variables to numerical before passing to the model. The predictor data is then passed to the model to make a price prediction. This output is passed back to the front-end application to show the predicted used car price.

```

from flask import Flask, render_template, request
import numpy as np
import pickle

app = Flask(__name__)

# Load pickle files for the model as well as the labelencoder objects for City, State and maker
model = pickle.load(open('model.p', 'rb'))
city_label = pickle.load(open('city_label.p', 'rb'))
state_label = pickle.load(open('state_label.p', 'rb'))
make_label = pickle.load(open('make_label.p', 'rb'))

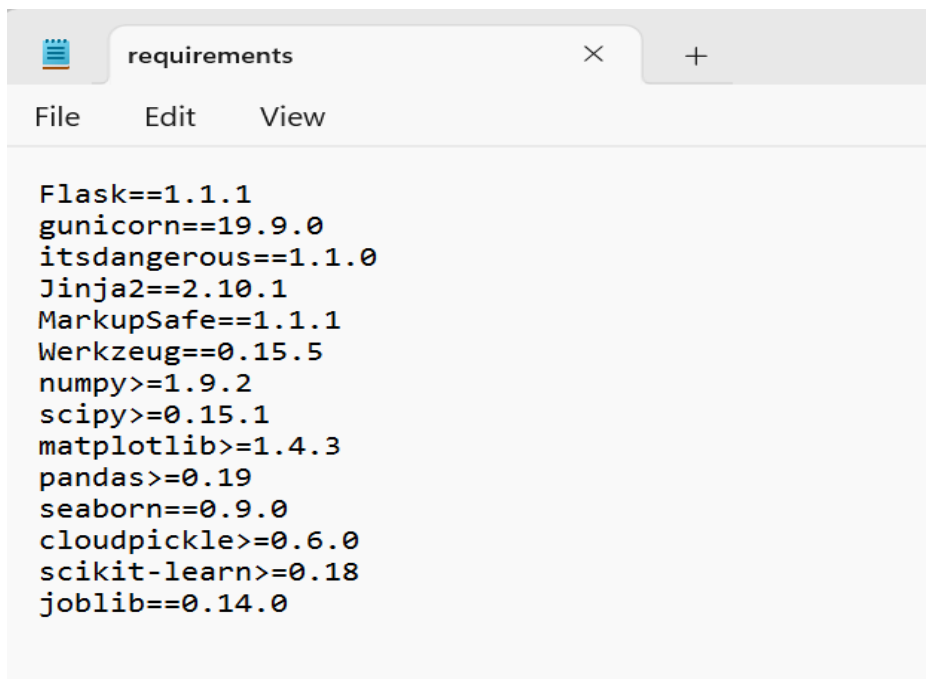
@app.route('/')
def index():
    return render_template('input.html') # render the input html template upon opening the url

@app.route('/predict', methods=['POST'])
def predict():
    input_data = [x for x in request.form.values()] # Get predictor variable values from the html input form
    city = city_label[input_data[2]] # Parse the categorical City data into numerical to pass to model
    state = state_label[f'{input_data[3]}'] # Parse the categorical State data into numerical to pass to model
    maker = make_label[input_data[4]] # Parse the categorical maker data into numerical to pass to model
    input_data[2] = city
    input_data[3] = state
    input_data[4] = maker
    input_data = [int(x) for x in input_data]
    input_data = np.array(input_data)
    prediction = model.predict(input_data) # Predict the price based on passed values
    prediction = round(prediction[0], 2)
    return render_template('input.html', pred_text = f'The Car should be priced: {prediction} $') # Output the price back to html form

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

```

We also create a requirements.txt file, which holds the necessary python libraries required to run our application on Google Cloud Platform.



```

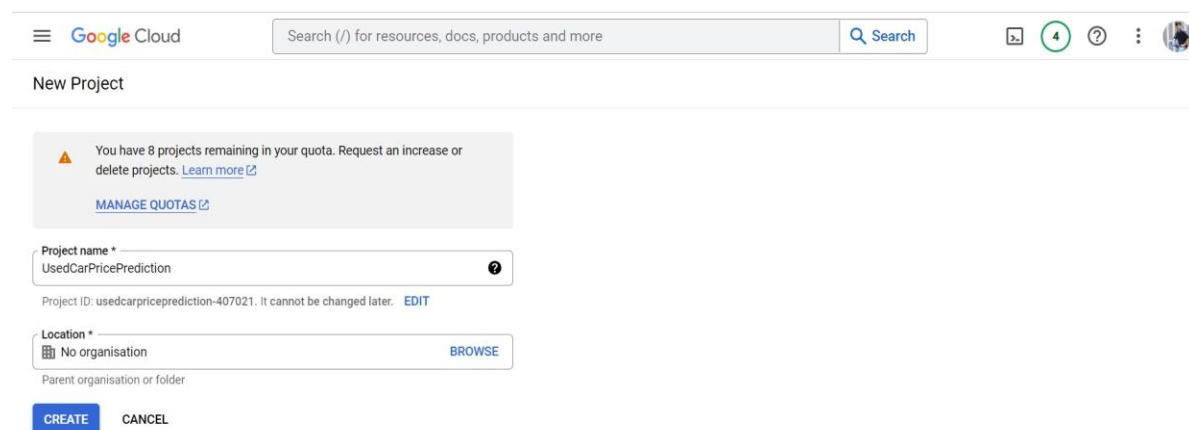
Flask==1.1.1
gunicorn==19.9.0
itsdangerous==1.1.0
Jinja2==2.10.1
MarkupSafe==1.1.1
Werkzeug==0.15.5
numpy>=1.9.2
scipy>=0.15.1
matplotlib>=1.4.3
pandas>=0.19
seaborn==0.9.0
cloudpickle>=0.6.0
scikit-learn>=0.18
joblib==0.14.0

```

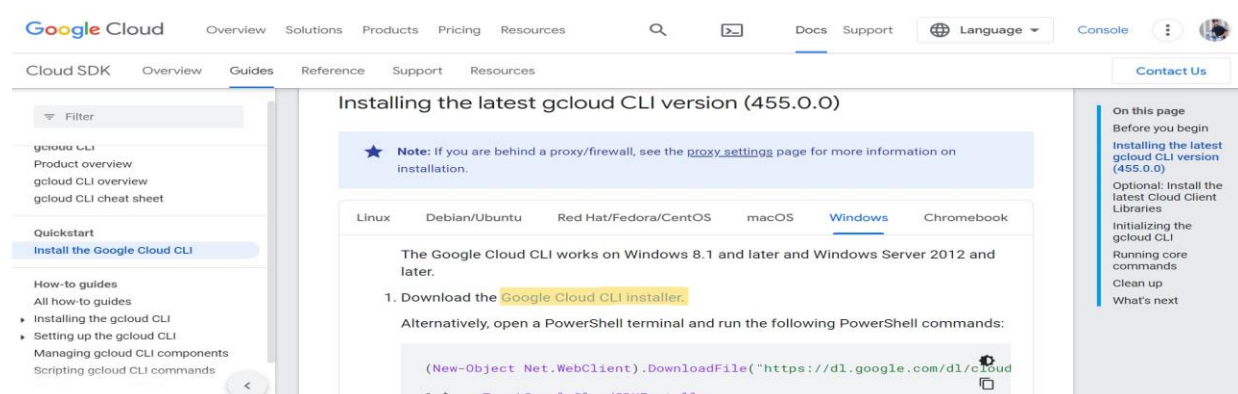
Along with that, GCP also requires an app.yaml file which specifies the python version to be used to run our application.

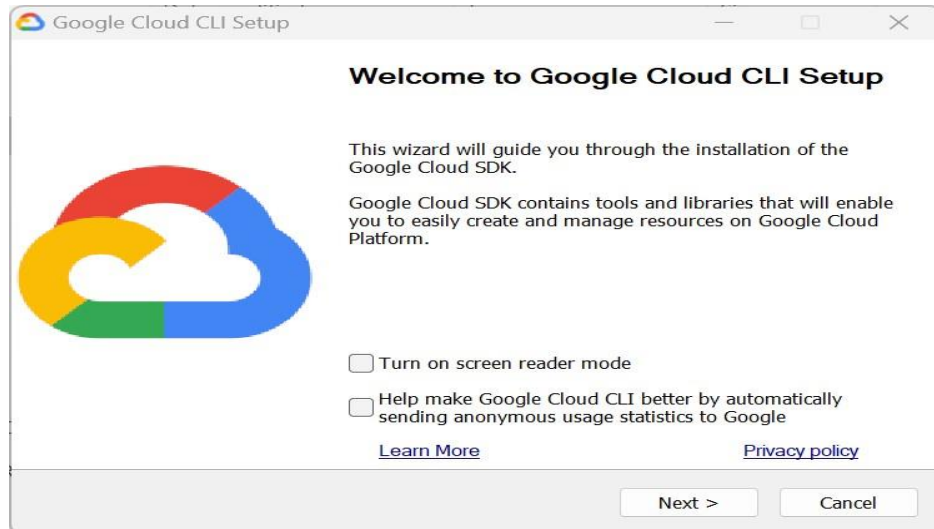
```
! app.yaml X
C: > Users > stjarn > OneDrive > Documents > Data Glacier > Week 5 > GCP > ! app.yaml
1 runtime: python39
```

Now after creating the pre-requisite front-end, API, requirement and app files, we now proceed to steps for hosting the application on GCP. At first, we will need to create a trail account on GCP. After creating the account, we create a new project on the GCP for our used car price application as shown below.



Next, we download the Google cloud SDK application from Google Cloud website and install it on our local machine. It is a tool that contains set of commands for developing and managing resources on Google Cloud Platform.





We now open the Google Cloud SDK Command Prompt, and a login window pops up on the browser. We then login into our Google Cloud Platform account.

```
C:\WINDOWS\SYSTEM32\cmd.exe
Welcome to the Google Cloud CLI! Run "gcloud -h" to get the list of available commands.
---
Welcome! This command will take you through the configuration of gcloud.
Your current configuration has been set to: [default]
You can skip diagnostics next time by using the following flag:
  gcloud init --skip-diagnostics
Network diagnostic detects and fixes local network connection issues.
Checking network connection...done.
Reachability Check passed.
Network diagnostic passed (1/1 checks passed).
You must log in to continue. Would you like to log in (Y/n)? Y
Your browser has been opened to visit:
https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=32555940559.apps.googleusercontent.com&redirect_uri=http%3A%2F%2Flocalhost%3A8085%2F&scope=openid+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.email+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fappengine.admin+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fsqlservice.login+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcompute+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Faccounts.reauth&state=ngYxjIa5glZe9h8CL3qvlkXd1zeVvx&access_type=offline&code_challenge=MscSvLqn0mhTchryOvxqpANK6P2Z7sI4s1az2w7PGFg&code_challenge_method=S256
```



We select the project we just created on Google Cloud Platform from the list of projects displayed on the command prompt.

```
You are logged in as: [st.jampani@gmail.com].

Pick cloud project to use:
[1] fine-effect-407006
[2] gentle-breaker-407018
[3] just-experience-407018
[4] profound-hydra-407018
[5] usedcarpriceprediction-407021
[6] Enter a project ID
[7] Create a new project
Please enter numeric choice or text value (must exactly match list item): 5

Your current project has been set to: [usedcarpriceprediction-407021].
```

Then, we navigate to the directory on the local machine that contains the necessary files required to deploy our model on the Google Cloud Platform and then type the following command to deploy our model on cloud as a part of the selected project.

```
C:\Users\stjam\AppData\Local\Google\Cloud SDK>cd c:\Users\stjam\OneDrive\Documents\Data Glacier\Week 5\GCP

c:\Users\stjam\OneDrive\Documents\Data Glacier\Week 5\GCP>gcloud app deploy app.yaml --project usedcarpriceprediction
```

We then need to specify the region in which the model is to be hosted.

```
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: 18

Creating App Engine application in project [usedcarpriceprediction-407021] and region [us-east1]...done.
```

The SDK command prompt uploads the files on the Google Cloud Platform and returns the URL on which the application is deployed and hosted.

```
Services to deploy:

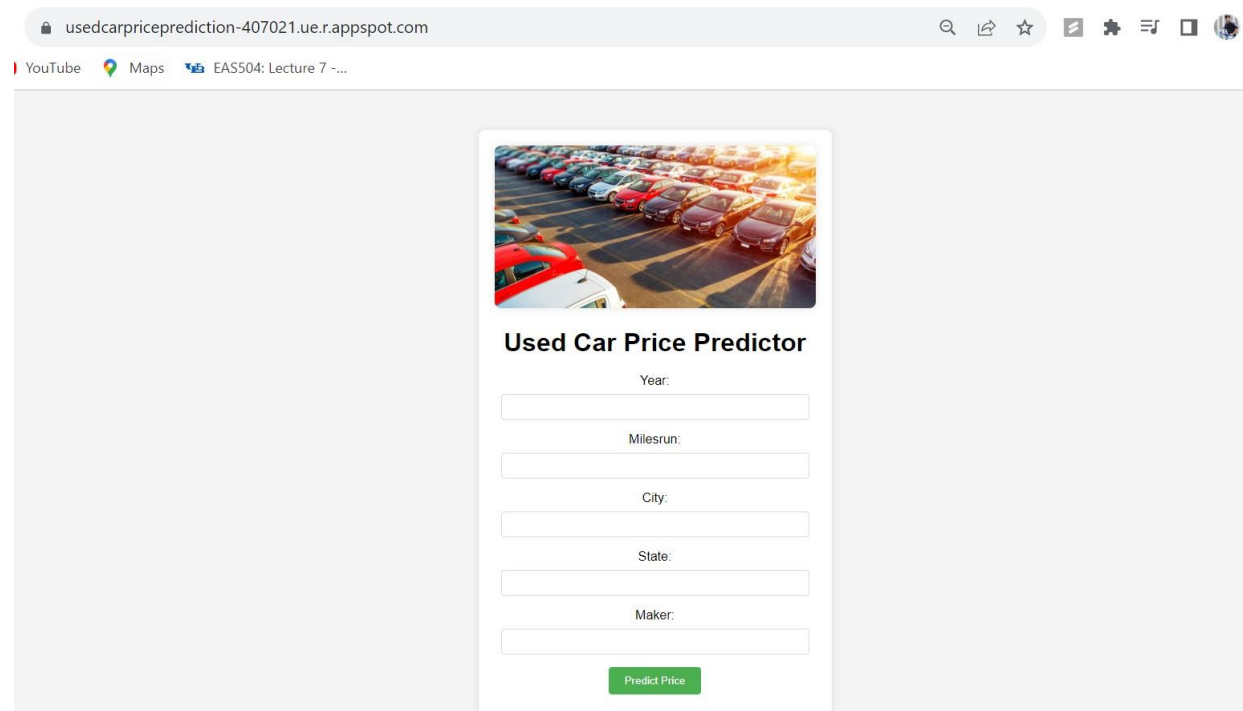
descriptor:      [c:\Users\stjam\OneDrive\Documents\Data Glacier\Week 5\GCP\app.yaml]
source:          [c:\Users\stjam\OneDrive\Documents\Data Glacier\Week 5\GCP]
target project:  [usedcarpriceprediction-407021]
target service:  [default]
target version:  [20231203t174320]
target url:      [https://usedcarpriceprediction-407021.ue.r.appspot.com]
target service account: [usedcarpriceprediction-407021@appspot.gserviceaccount.com]

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

Beginning deployment of service [default]...
#=====#
#= Uploading 1 file to Google Cloud Storage           =#
#=====#
File upload done.
Updating service [default]...done.
Setting traffic split for service [default]...done.
Deployed service [default] to [https://usedcarpriceprediction-407021.ue.r.appspot.com]
```

Results:

We can see that the model is deployed on Google Cloud once we open the url. The working of the model can be verified from the below images.



usedcarpriceprediction-407021.ue.r.appspot.com

YouTube Maps EAS504: Lecture 7 -...

Used Car Price Predictor

Year:

Milesrun:

City:

State:

Maker:



Used Car Price Predictor

Year:

2015

Milesrun:

35000

City:

Albany

State:

NY

Maker:

Ford

Predict Price



Used Car Price Predictor

Year:

Milesrun:

City:

State:

Maker:

Predict Price

The Car should be priced: 25004.09 \$