Week 5 assignment: Cloud and API deployment

Name: Roger Burek-Bors
Batch code: LICAN01
Submission date: March 28, 2021
Submitted to: Data Glacier

1. Introduction (common with assignment for week 4)
   I am fan of cars therefore I decided to use for this week assignment "FuelConsumptionCo2.csv" which is a 1000-point data set about modern cars. This data set contains such features like:
   - model of car,
   - year of introduction,
   - engine size,
   - cylinder,
   - fuel consumption,
   - CO2 emissions.

   My aim was to create a prediction model that would help to determine CO2 emissions when it comes to engine size.

2. Building the model (common with assignment for week 4)
   https://github.com/rodbergerrone/VC/blob/rbb/Model_deploy/FuelConsumptionCo2_Regr.ipynb
   a) I used Jupyter Notebook for "FuelConsumptionCo2.csv" analysis. I determined 4 most promising features for CO2 emissions predictions:

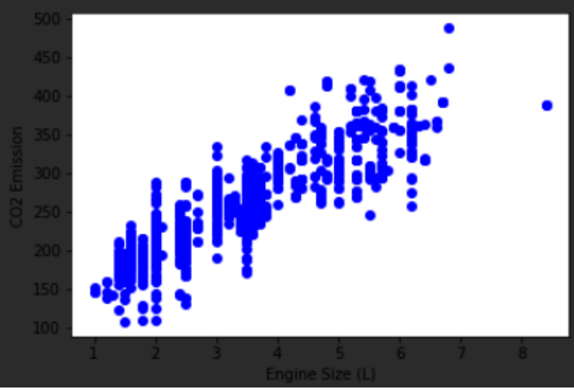
   Choosing data for modelling

```
cdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB','CO2EMISSIONS']]
cdf.head(5)
```

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_COMB	CO2EMISSIONS
0	2.0	4	8.5	196
1	2.4	4	9.6	221
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244

   b) Analysis led to conclusions that features like engine size, cylinders and fuel consumption have almost linear correlation with CO2 emissions feature therefore a linear regression model could be built. To keep this assignment simple I decided that model will be predicting CO2 emission based on only one feature – engine size.

```python
plt.scatter(cdf.ENGINESIZE, cdf.CO2EMISSIONS,  color='blue')
plt.xlabel("Engine Size (L)")
plt.ylabel("CO2 Emission")
plt.show()
```



c) Building the model was achieved as on following pictures with help of Scikit-learn library:

```python
msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]
```

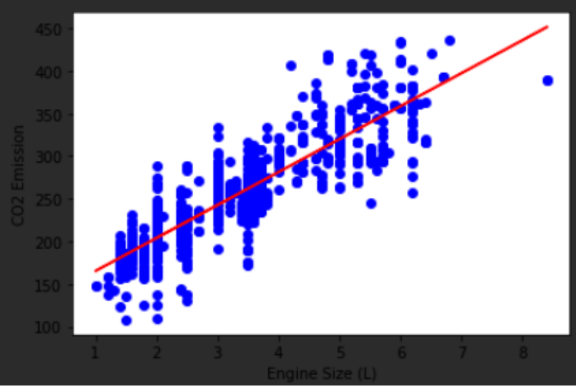
```python
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS,  color='blue')
plt.xlabel("Engine Size (L)")
plt.ylabel("CO2 Emission")
plt.show()
```

```python
from sklearn import linear_model
regr = linear_model.LinearRegression()
train_x = np.asanyarray(train[['ENGINESIZE']])
train_y = np.asanyarray(train[['CO2EMISSIONS']])
regr.fit (train_x, train_y)
print('Coefficients: ', regr.coef_)
print('Intercept: ',regr.intercept_)
```

```
Coefficients:  [[38.70010101]]
Intercept:  [126.4800893]
```

```python
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS,  color='blue')
plt.plot(train_x, regr.coef_[0][0]*train_x + regr.intercept_[0], '-r')
plt.xlabel("Engine Size (L)")
plt.ylabel("CO2 Emission")
```

```
Text(0, 0.5, 'CO2 Emission')
```



d) I performed model evaluation to see how prediction behave in correlation to data in “FuelConsumptionCo2.csv” and on leaflets of car dealerships. It worked really fine. I also checked metrics available in Scikit-learn library:

```
13  print("For 2 L engine:", regr.predict(np.array([[2]])))
    print("For 5 L engine:", regr.predict(np.array([[5]])))

    For 2 L engine: [[203.88029131]]
    For 5 L engine: [[319.98059433]]
```

```
14  from sklearn.metrics import r2_score

    test_x = np.asanyarray(test[['ENGINESIZE']])
    test_y = np.asanyarray(test[['CO2EMISSIONS']])
    test_y_ = regr.predict(test_x)

    print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y_ - test_y)))
    print("Residual sum of squares (MSE): %.2f" % np.mean((test_y_ - test_y) ** 2))
    print("R2-score: %.2f" % r2_score(test_y , test_y_) )

    Mean absolute error: 22.11
    Residual sum of squares (MSE): 851.73
    R2-score: 0.80
```

Coefficient of determination is 0.8 which in my opinion is quite good.

e) Then I serialized the model with Pickle library as “pickle_model.pkt”:

```
15  with open('pickle_model.pkl', 'wb') as f:
        pickle.dump(regr, f)
```

3. Building Flask backend (common with assignment for week 4)
   https://github.com/rodbergerrone/VC/blob/rbb/Model_deploy/deploy_flask.py
   I used Pycharm to develop Python code for backend:

```
deploy_flask.py

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


app = Flask(__name__)
with open('pickle_model.pkl', 'rb') as f:
    model = pickle.load(f)


@app.route('/')
def home():
    return render_template('index.html')


@app.route('/predict', methods=['POST'])
def predict():
    int_features = [float(x) for x in request.form.values()]
    final_features = [np.array(int_features)]
    prediction = model.predict(final_features)

    prediction = round(prediction[0][0], 2)

    return render_template('index.html', prediction_text='CO2 emission for this size of engine should be {} G/KM'
                           .format(prediction))


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

4. Building Flask frontend (common with assignment for week 4)
   https://github.com/rodbergerrone/VC/blob/rbb/Model_deploy/templates/index.html
   I used Pycharm to develop Python code for frontend:

deploy_flask.py × index.html ×

```html
<!DOCTYPE html>
<html >
<head>
  <meta charset="UTF-8">
  <title>CO2 Emission Prediction</title>
  <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">

</head>

<body style="background: #050;">
 <div class="login">
    <h1>CO2 Emission Prediction Based On Car Engine Size</h1>

     <!-- Main Input For Receiving Query to our ML -->
    <form action="{{ url_for('predict')}}"method="post">
        <input type="text" name="car engine size" placeholder="car engine size" required="required" />
        <button type="submit" class="btn btn-primary btn-block btn-large">Predict CO2 Emission</button>
    </form>

   <br>
   <br>
   {{ prediction_text }}

 </div>
</body>
</html>
```

5. Performing predictions on local network with Flask (common with assignment for week 4)

Run: deploy_flask ×

```
C:\Users\RBB\miniconda3\envs\Glacier\python.exe C:/Users/RBB/OneDrive/Python/Github/VC/Model_deploy/deploy_flask.py
 * Serving Flask app "deploy_flask" (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 stat
 * Debugger is active!
 * Debugger PIN: 300-683-121
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Comparing model predictions with automotive specifications:

a) frontend of my predictor – the result: 188.4 G/KM is for 1.6 L engine

b) specification of Toyota Yaris GR – the result: 186 G/KM is for 1.6 L engine

6. Deployment on Cloud with Heroku after creating account on www.heroku.com

a) creating Procfile which will guide Heroku on how the application should run:

Procfile × requirements.txt ×

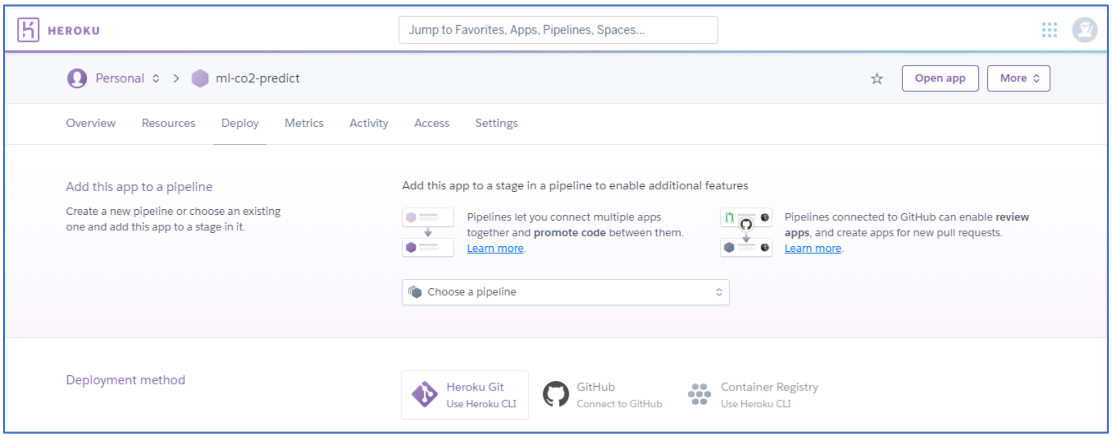
```
web: gunicorn deploy_flask:app
```

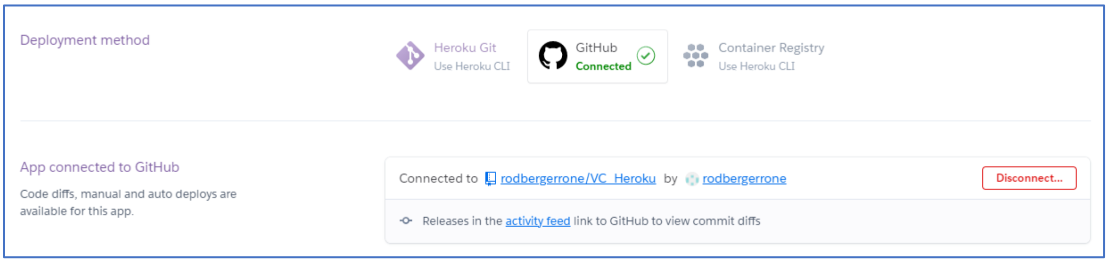
b) creating requirements.txt which will guide Heroku about installation of necessary libraries:

Procfile × requirements.txt ×

```
numpy==1.19.2
flask==1.1.2
pickleshare==0.7.5
scikit-learn==0.24.1
gunicorn==19.9.0
```

c) linking Github repository with Heroku account and setting deployment method:



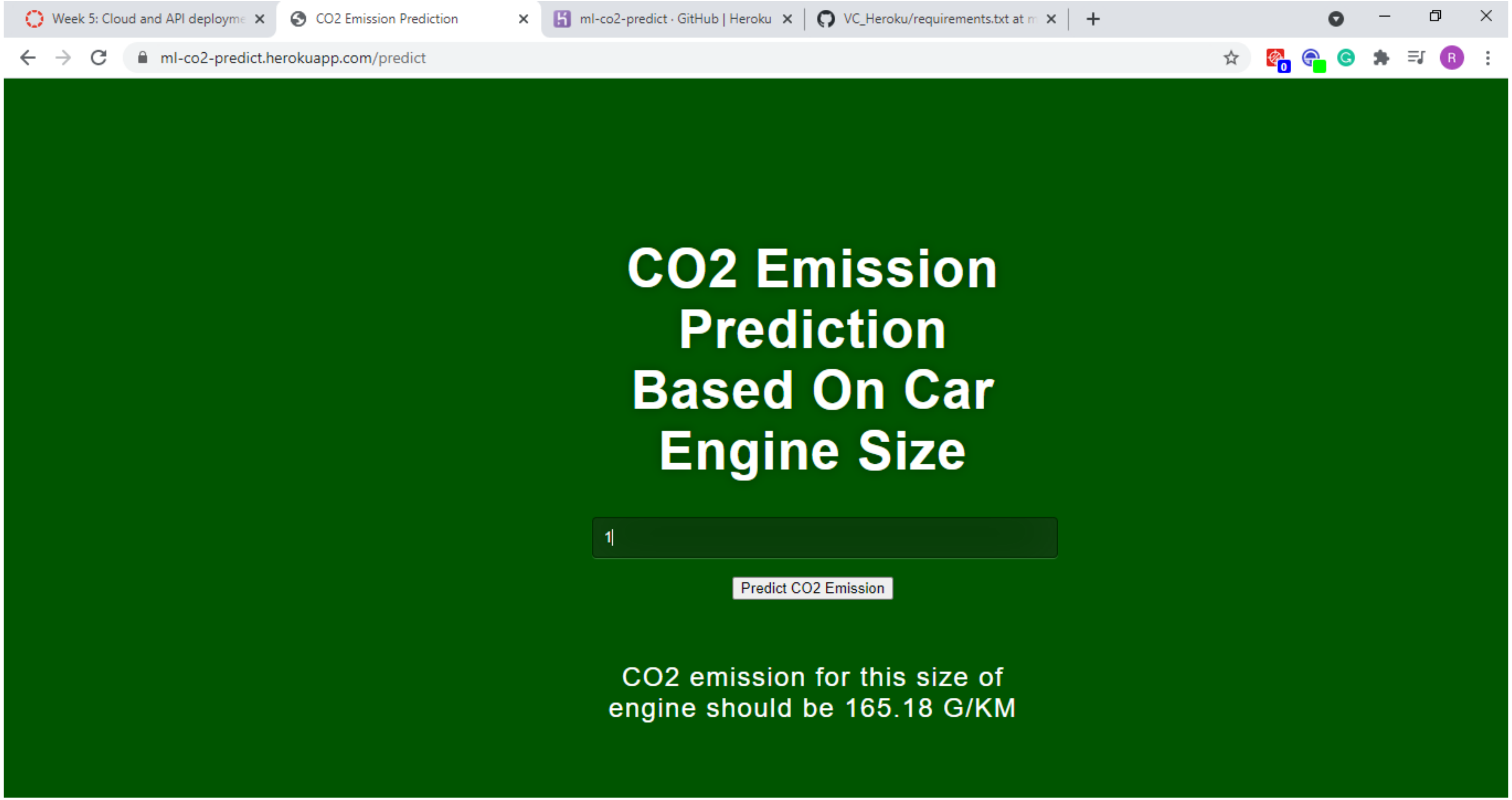


My model was successfully deployed to Heroku and can be accessed at: https://ml-co2-predict.herokuapp.com/

7. API feature was implemented as the separate method in flask app (see code below). To access API please go to https://ml-co2-predict.herokuapp.com/api/ and after slash provide size of engine.

```
@app.route('/api')
@app.route('/api/')
@app.route('/api/<l>')
@app.route('/api/<l>/')
def api_l(l=None):
    if not l:
        message = jsonify(message='Size of engine (l) not provided')
        return make_response(message, 400)
    else:
        l = float(l)
        prediction = model.predict(np.array([[l]]))
        prediction = round(prediction[0][0], 2)
        return jsonify(
            CO2_emission=prediction,
            engine_size=l)
```

Web based implementation:



API based implementation: