1. **Pickling the model (or other containers):**

[***Pickle***](https://www.tutorialspoint.com/python-pickling) ***:*** *Python pickle module is used for serializing and de-serializing python object structures. The process to convert any kind of python object (list, dict, etc.) into byte streams (0s and 1s) is called pickling or serialization or flattening or marshalling. We can converts the byte stream (generated through pickling) back into python objects by a process called as unpickling.*

import pandas as pd

from sklearn.linear\_model import LinearRegression

import pickle

df = pd.read\_csv("FuelConsumption.csv")

#use required features

cdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION\_COMB','CO2EMISSIONS']]

#Training Data and Predictor Variable

# Use all data for training (tarin-test-split not used)

x = cdf.iloc[:, :3]

y = cdf.iloc[:, -1]

regressor = LinearRegression()

#Fitting model with trainig data

regressor.fit(x, y)

# Saving model to current directory

# Pickle serializes objects so they can be saved to a file, and loaded in a program again later on.

pickle.dump(regressor, open('model.pkl','wb'))

'''

#Loading model to compare the results

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

print(model.predict([[2.6, 8, 10.1]]))

'''

1. **Getting started on Flask or Django:**

**#import libraries**

**import numpy as np**

**from flask import Flask, render\_template,request**

**import pickle#Initialize the flask App**

**app = Flask(\_\_name\_\_)**

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

**Define app route:**

**#default page of our web-app**

**@app.route('/')**

**def home():**

**return render\_template('index.html')**

**Redirect api for predictions:**

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

**def predict():**

**#For rendering results on HTML GUI**

**int\_features = [float(x) for x in request.form.values()]**

**final\_features = [np.array(int\_features)]**

**prediction = model.predict(final\_features)**

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

**return render\_template('index.html', prediction\_text='CO2 best crop to use :{}'.format(output))**

**Start flask server:**

**if \_\_name\_\_ == "\_\_main\_\_":**

**app.run(debug=True)**

1. **Building platform:**

Deploy your Web Application on Heroku

Now that our application has been successfully tested on the local server, it’s time to deploy our application on Heroku- cloud platform. There are two prerequisites to deploy any flask web-app to Heroku.

On the Project Structure, you might have noticed two new files named “Procfile” and “requirements.txt”. These two files are required to deploy your app on Heroku.

Before creating Procfile, we need to install Gunicorn. You can use the command ‘pip install gunicorn’ or use the above link to install libraries in PyCharm

1. **Embedding model:**

#### **Create Procfile :**

**A** [**Procfile**](https://devcenter.heroku.com/articles/procfile) **specifies the commands that are executed by a Heroku app on startup. Open up a new file named Procfile (without any extension) in the working directory and paste the following.**

**Resources:**

[**https://www.analyticsvidhya.com/blog/2020/09/integrating-machine-learning-into-web-applications-with-flask/**](https://www.analyticsvidhya.com/blog/2020/09/integrating-machine-learning-into-web-applications-with-flask/)

[**https://medium.com/fintechexplained/flask-host-your-python-machine-learning-model-on-web-b598151886d**](https://medium.com/fintechexplained/flask-host-your-python-machine-learning-model-on-web-b598151886d)

[**https://www.youtube.com/playlist?list=PLillGF-RfqbbbPz6GSEM9hLQObuQjNoj\_**](https://www.youtube.com/playlist?list=PLillGF-RfqbbbPz6GSEM9hLQObuQjNoj_)

**CHECKLIST SCHEDULE:**

* **Tuesday: Dry Run Test with example Pickle File**

**--See where and what we need to improve/adjust to potentially consider alternative methods if needed.**

* **Wednesday: Try Improving**
* **Thursday: Dry Run Deployment   
   --- Access and report what did and did not work**