Data Glacier
Your Deep Learning Partner

Deployment of a ML Model on Flask

Zakaria Arshad
LISUM18

Submitted for Data Glacier

February 12, 2023

Data Glacier
Your Deep Learning Partner

Table of Contents

1) Introduction
2) Choosing a model
3) Model Building
    a) 3.1 - model.py
    b) 3.2 - app.py
4) Web App

Data Glacier
Your Deep Learning Partner

1. Introduction

The objective is to build a Machine Learning model with a dataset, and deploy the model through the Flask Framework available through Python.

The selected dataset takes car data from 1987, and has fields such as car_id and fuel type.

The four selected fields for this model are:

- *Highwaympg* is the miles per gallon the car gets on the highway.
- *Horsepower* is the total horsepower of the car.
- *Enginesize* is the size of the engine.
- *Price* is the price of the car.

Here is the head of the dataset:

	car_ID	symboling	CarName	...	citympg	highwaympg
0	1	3	alfa-romero giulia	...	21	27
1	2	3	alfa-romero stelvio	...	21	27
2	3	1	alfa-romero Quadrifoglio	...	19	26
3	4	2	audi 100 ls	...	24	30
4	5	2	audi 100ls	...	18	22

2. Choosing a Model

The dependent variable for this model is the price. The model will be using the independent variables of *Highwaympg, Horsepower,* and *Enginesize* to predict the price.

Because the dependent variable is continuous, I have selected to use a linear regression model.

3. Model Building

3.1 model.py

I first imported all the necessary modules required.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import pickle
```

Then I loaded the csv file and made a copy in order to filter it.

```
# Load CSV file and filter for specific columns
original_df = pd.read_csv("CarPrice_Assignment.csv")
df = original_df.copy()
```

I filtered the copied dataset by the columns I was looking for. Because the dataset was from cars from 1987, I adjusted the prices for inflation of what they would be today. I then checked if there were any null values.

```
original_df = pd.read_csv("CarPrice_Assignment.csv")
df = original_df.copy()
print(df.head())
df = df[['horsepower', 'enginesize', 'highwaympg', 'price']]
df['price'] = df['price'] * 2.67
nan_counts = df.isna().sum()
```

I made my X and y variables for my model. I set X equal to all columns except for Price, and then set my y to Price. The test size is .3, which means 30% of the total dataset is randomly selected to be used to test the model. Thus, 70% is used to train the model.

```
X = df.drop("price", axis=1)
y = df["price"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
```

I then trained the model, and checked the R^2 value for model accuracy.

```
R² Score: 0.8103559685269346
```

Thus 80% of the variance in price can be predicted by the independent variable.

Finally, the model is saved as a pickle file to be stored for later.

3.2 app.py

I imported numpy, pickle, and three methods from the Flask package. I then created the Flask app, and loaded the pickle module

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

# Create flask app
app = Flask(__name__)

# Load the pickle model
model = pickle.load(open("model.pkl", "rb"))
```

I used a decorator to modify the routing of the root, or home page. Within this, I defined a function that returned an HTML file containing the HTML/CSS for the web app.

```
@app.route("/")
def Home():
    return render_template("index.html")
```

I used another decorator to modify the routing for when the “predict” button is clicked. I used the post method as the parameters are stored in the message body, when the user inputs their data.

Then, I used a list comprehension to make the inputted values by the user integers. I put these values into a numpy array, and used the array to make a prediction. Then, the function returns the HTML file while changing the prediction_text accordingly.

```
@app.route("/predict", methods = ["POST"])
def predict():
    int_features = [int(x) for x in request.form.values()]
    features = [np.array(int_features)]
    prediction = model.predict(features)

    return render_template("index.html", prediction_text = "The predicted price is {}".format(prediction))
```

Finally, the code to run the program.

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

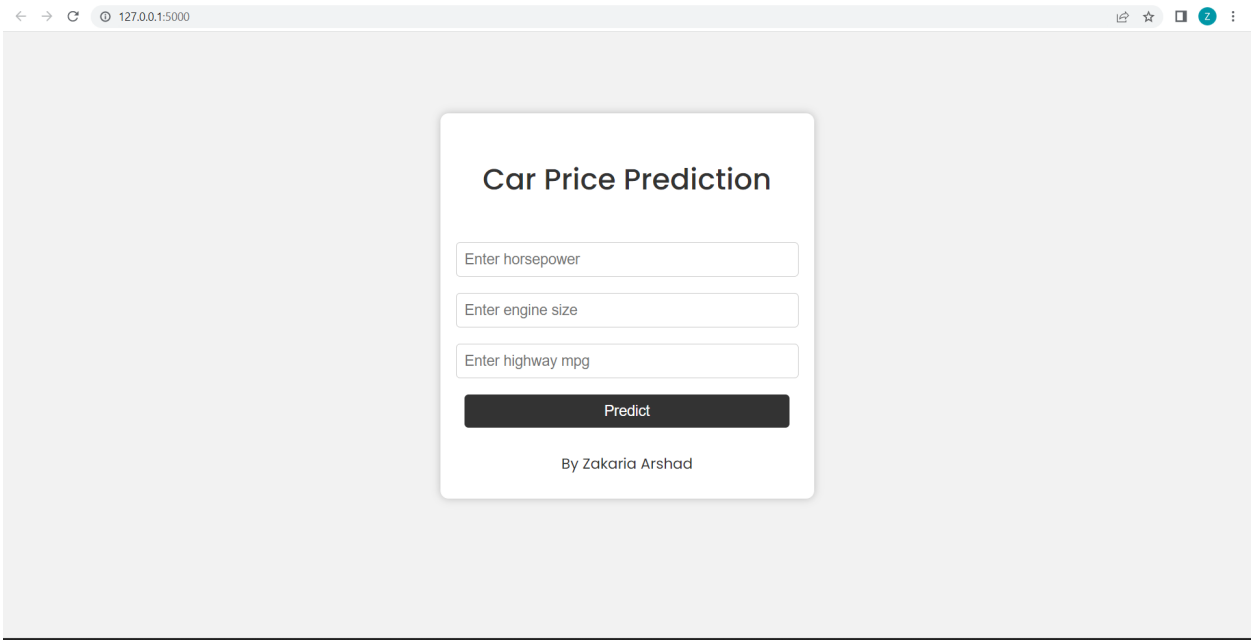
3.3 index.html

The most important part of this file is below, where the code generates a url for prediction. It also contains the code for the three input parameters. Finally, there is a predict button. Notice how the method is also post here.

```
<form action="{{ url_for('predict')}}" method="post">
  <input type="text" name="horsepower" placeholder="Enter horsepower" required="required" />
  <input type="text" name="engine_size" placeholder="Enter engine size" required="required" />
  <input type="text" name="highway_mpg" placeholder="Enter highway mpg" required="required" />
  <button type="submit" class="btn">Predict</button>
</form>
```

4. Web App

This is what the completed web app looks like.



Final.