Data Glacier

Deployment on Flask

Name: Abishek James

Batch Code: LISUM19

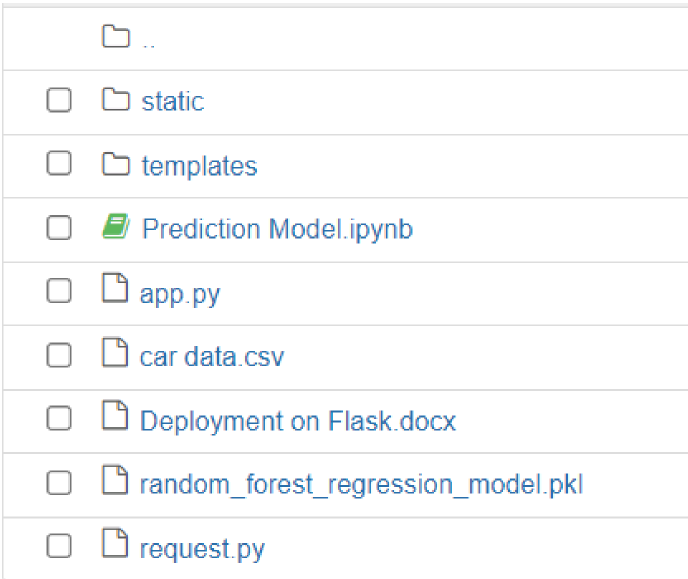
Date: 27 March 2023

Submitted to: Data Glacier

Submitted link: https://github.com/abishekjames/Data-Glacier-intern-week4

Overall structure of the project

We create a folder for this project. Below figure shows the directory tree inside the folder



Project directory structure

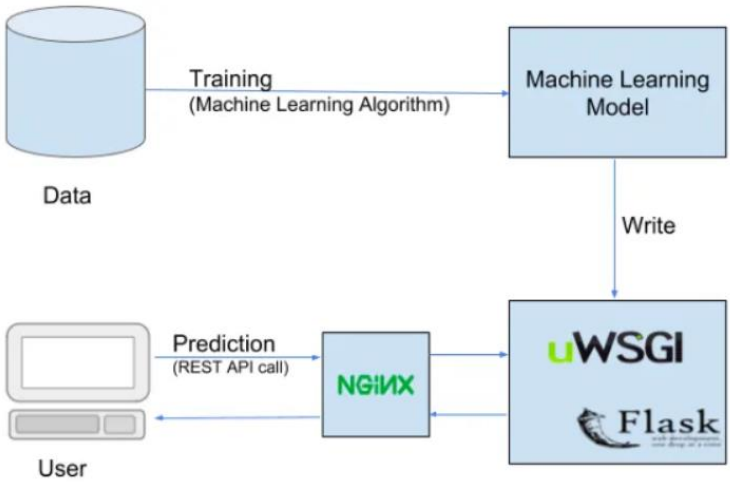
1. **prediction model.ipynb** (jupyter notebook) — This contains code for the machine learning model to predict price base on used car data in 'car.csv' file.

2. **app.py** — This contains Flask APIs that receives sales details through GUI or API calls, computes the predicted value based on our model and returns it.request.py — This uses requests module to call APIs defined in app.py and displays the returned value.

3. **template** — This folder contains the HTML template (**index.html**) to allow user to enter enter car or price details and displays the predicted price in lakhs.

4. **static** — This folder contains the **css** folder with **(style.css)** file which has the styling required for out index.html file.

Application workflow



We will be building a machine model, then create an API for the model using Flask, the python micro-framework for building web applications. This API allows us to utilize predictive capabilities through HTTP requests

Install Required libraries

We must install many required libraries which will be used in this model. Use pip command to install all the libraries.

```
pip install pandas
pip install numpy
pip install scikit-learn
pip install flask
```

About the Dataset

This dataset contains information about used cars listed on www.cardekho.com This data can be used for a lot of purposes such as price prediction to exemplify the use of linear regression in Machine Learning. The columns in the given dataset is as follows:

- Car_Name
- Year
- Selling_Price
- Present_Price
- Kms_Driven
- Fuel_Type
- Seller_Type
- Transmission
- Owner

Import Required libraries and Dataset

In [2]:
```
# import required libraries
import warnings
import pandas as pd
import numpy as np
import seaborn as sns
import pickle
warnings.filterwarnings("ignore")
```

In [3]:
```
df=pd.read_csv('car data.csv')
df.head()
```

Out[3]:

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
0	Innova	2022	18.25	19.99	17000	Petrol	Dealer	Manual	0
1	Fortuner	2020	31.50	34.17	23000	Diesel	Dealer	Manual	0
2	Swift	2019	5.40	8.98	36900	Petrol	Dealer	Manual	0
3	Polo	2019	7.85	9.15	5200	Petrol	Dealer	Manual	0
4	Corolla	2020	19.50	20.25	42450	Diesel	Dealer	Manual	0

In [141]:
```
df.shape
```

Out[141]: (301, 9)

In [142]:
```
print(df['Seller_Type'].unique())
print(df['Fuel_Type'].unique())
print(df['Transmission'].unique())
print(df['Owner'].unique())
```

```
['Dealer' 'Individual']
['Petrol' 'Diesel' 'CNG']
['Manual' 'Automatic']
[0 1 3]
```

Data pre-processing

1)To calculate the vehicle age, we are adding a new column and difference of current year and vehicle's year will be the age of vehicle

In [7]:
```
# setting current year as 2023
final_dataset['Current Year']=2023
```

In [12]:
```
#calculating no of year by subtracting current year-year
final_dataset['no_year']=final_dataset['Current Year']- final_dataset['Year']
```

In [11]:
```
final_dataset.head()
```

Out[11]:

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Current Year	no_year
0	2022	18.25	19.99	17000	Petrol	Dealer	Manual	0	2023	1
1	2020	31.50	34.17	23000	Diesel	Dealer	Manual	0	2023	3
2	2019	5.40	8.98	36900	Petrol	Dealer	Manual	0	2023	4
3	2019	7.85	9.15	5200	Petrol	Dealer	Manual	0	2023	4
4	2020	19.50	20.25	42450	Diesel	Dealer	Manual	0	2023	3

Car age is affecting negatively as the Selling Price decreases for an older car.

2)We need to convert categorical features to numeric type. To produce an actual dummy encoding from a DataFrame, we need to pass drop_first=True.

```
In [14]: final_dataset=pd.get_dummies(final_dataset,drop_first=True)
```

3)Selecting indepent and dependent varaible

```
In [15]: # Select independent and dependent variable
         X=final_dataset.iloc[:,1:]
         y=final_dataset.iloc[:,0]
```

4)Here, we are using ExtraTreeRegressor to get the important of the features in the dataset.

```
In [19]: ### Feature Importance

         from sklearn.ensemble import ExtraTreesRegressor
         import matplotlib.pyplot as plt
         model = ExtraTreesRegressor()
         model.fit(X,y)

Out[19]: ▸ ExtraTreesRegressor
```

```
In [106]: print(model.feature_importances_)

          [0.37587921 0.03735726 0.00097047 0.08064828 0.22844919 0.00780328
           0.13445871 0.13443361]
```

5)Now split the data into train and test for building a model.

```
In [42]: # Split the dataset into train and test
         from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

Build the model

1)Now I used Random Forest Regressor algorithm for predicting the car price. It is based on Decision trees.

```
In [25]: from sklearn.ensemble import RandomForestRegressor
```

```
In [26]: regressor=RandomForestRegressor()
```

2) Then I Used Hyperparameter Tuning for improving the performance of the model and algorithm. I have used RandomizerSearchCV for hyperparameter tuning.

```python
from sklearn.model_selection import RandomizedSearchCV
```

```python
#Randomized Search CV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
# max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]
```

```python
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}

print(random_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200], 'max_features': ['auto', 'sqrt'], 'max_dept
h': [5, 10, 15, 20, 25, 30], 'min_samples_split': [2, 5, 10, 15, 100], 'min_samples_leaf': [1, 2, 5, 10]}
```

3)Now fit the data in the model and this will take time to train the model.

```python
In [48]: rf_random.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   2.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   1.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   1.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   1.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   1.1s
```

Model evaluation

By three common evaluation metrices for regression problems.

```python
In [127]: from sklearn import metrics
```

```python
In [128]: print('MAE:', metrics.mean_absolute_error(y_test, predictions))
          print('MSE:', metrics.mean_squared_error(y_test, predictions))
          print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 0.9021203296703302
MSE: 3.6570348651351705
RMSE: 1.912337539540332
```

Save the model

Now we can save the model using pickle

```python
In [129]:
# open a file, where you ant to store the data
file = open('random_forest_regression_model.pkl', 'wb')

# dump information to that file
pickle.dump(rf_random, file)
```

App.py

Create a new app.py file. Now, import every important module and library to deploy the model. Also load the model in the app.py file.

The next part was to make an API which receives prediction details through GUI and computes the predicted price on our model. For this I de- serialized the pickled model in the form of python object. I set the main page using **index.html.** On submitting the form values using POST request to `/predict`, we get the predicted price value.

app.py ● request.py index.html style.css

C: > Users > Admin > Desktop > Data Glacier > Week4 > app.py > standard_to

```python
from flask import Flask, request, jsonify, render_template
import pickle
import numpy as np
from sklearn.preprocessing import StandardScaler
app = Flask(__name__)
model = pickle.load(open('random_forest_regression_model.pkl', 'rb'))
@app.route('/',methods=['GET'])
def Home():
    return render_template('index.html')

standard_to = StandardScaler()
@app.route("/predict", methods=['POST'])
def predict():
    Fuel_Type_Diesel=0
    if request.method == 'POST':
        Year = int(request.form['Year'])
        Present_Price=float(request.form['Present_Price'])
        Kms_Driven=int(request.form['Kms_Driven'])
        Kms_Driven2=np.log(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
        else:
            Fuel_Type_Petrol=0
            Fuel_Type_Diesel=1
        Year=2023-Year
        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
```

```python
        else:
            Transmission_Mannual=0
        prediction=model.predict([[Present_Price,Kms_Driven2,Owner,Year,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="Sorry you cannot sell this car")
        else:
            return render_template('index.html',prediction_text="You Can Sell The Car at {} lakhs".format(output))
    else:
        return render_template('index.html')

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

```

The flask code can be explained in three sections:

1.Loading the saved model

We load the random_forest_regression_model.pkl file and initialize the flask app.

2. Redirecting the API to the home page index.html

After initializing the app, we have to tell Flask what we want to do when the web page loads. The line @app.route("/", methods = ["GET","POST"]) tells Flask what to do when we load the home page of our website.

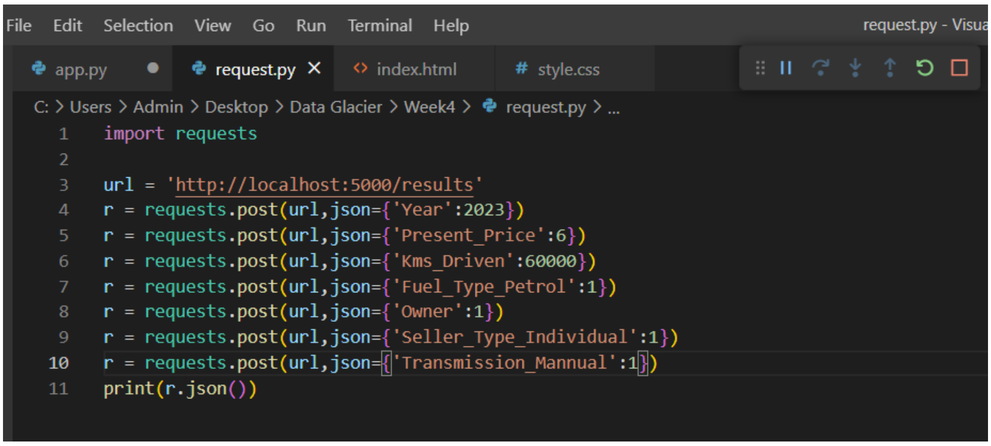
We use **@app.route('/')** to define functions which are used to redirect them into any number of URI with respect to the API. So, when you start the flask server, it redirects to index.html file by default in our case.

3. Redirecting the API to predict the result

Since it is a **'POST'** request, it will be reading the input values from **request.form.values().** Now that we have the input values in the variable **int_features**, we will convert it into an array and then use the model to predict it and round the final prediction to two decimal places.

app.run() and **run** our web page locally, hosted on your computer.

Finally we used requests **request.py** to call APIs defined in app.py

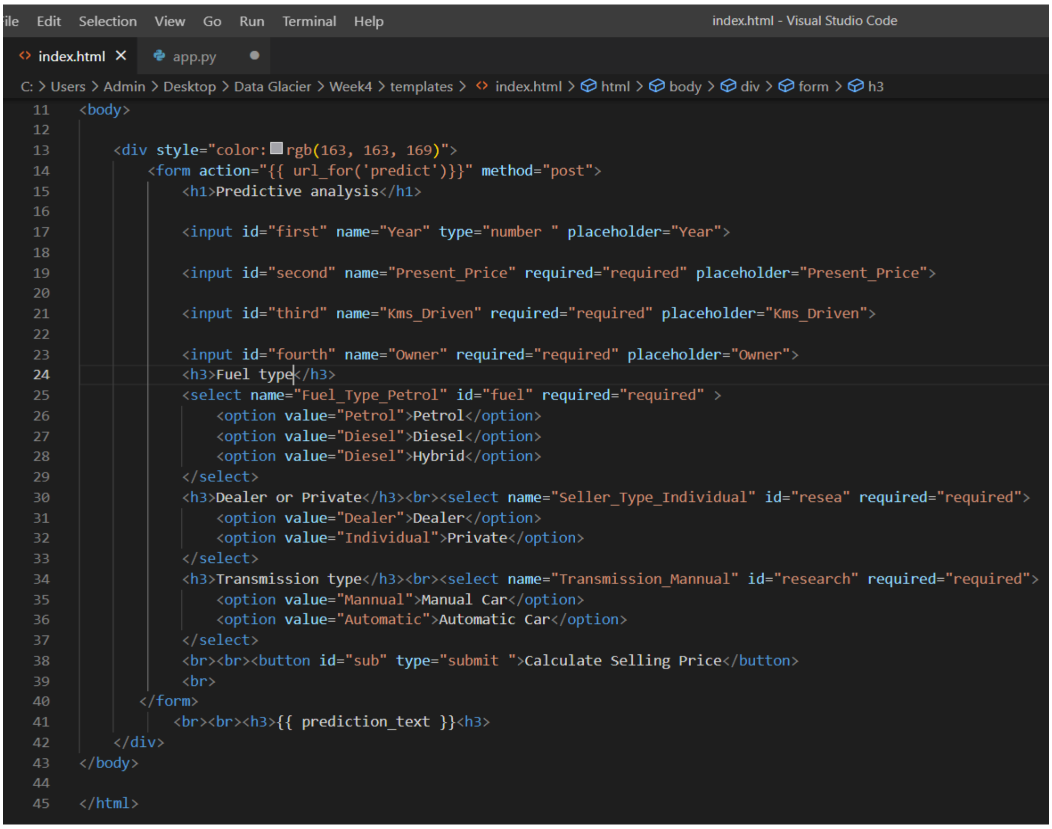



```python
import requests

url = 'http://localhost:5000/results'
r = requests.post(url,json={'Year':2023})
r = requests.post(url,json={'Present_Price':6})
r = requests.post(url,json={'Kms_Driven':60000})
r = requests.post(url,json={'Fuel_Type_Petrol':1})
r = requests.post(url,json={'Owner':1})
r = requests.post(url,json={'Seller_Type_Individual':1})
r = requests.post(url,json={'Transmission_Mannual':1})
print(r.json())
```

Index.html

The form action contains url_for('predict') which means that when the form is submitted which method to be invoked in the app.py file.



```
ile  Edit  Selection  View  Go  Run  Terminal  Help        index.html - Visual Studio Code
index.html ×   app.py
C: > Users > Admin > Desktop > Data Glacier > Week4 > templates > index.html > html > body > div > form > h3
<body>

    <div style="color:rgb(163, 163, 169)">
        <form action="{{ url_for('predict')}}" method="post">
            <h1>Predictive analysis</h1>

            <input id="first" name="Year" type="number " placeholder="Year">

            <input id="second" name="Present_Price" required="required" placeholder="Present_Price">

            <input id="third" name="Kms_Driven" required="required" placeholder="Kms_Driven">

            <input id="fourth" name="Owner" required="required" placeholder="Owner">
            <h3>Fuel type</h3>
            <select name="Fuel_Type_Petrol" id="fuel" required="required" >
                <option value="Petrol">Petrol</option>
                <option value="Diesel">Diesel</option>
                <option value="Diesel">Hybrid</option>
            </select>
            <h3>Dealer or Private</h3><br><select name="Seller_Type_Individual" id="resea" required="required">
                <option value="Dealer">Dealer</option>
                <option value="Individual">Private</option>
            </select>
            <h3>Transmission type</h3><br><select name="Transmission_Mannual" id="research" required="required">
                <option value="Mannual">Manual Car</option>
                <option value="Automatic">Automatic Car</option>
            </select>
            <br><br><button id="sub" type="submit ">Calculate Selling Price</button>
            <br>
        </form>
            <br><br><h3>{{ prediction_text }}<h3>
    </div>
</body>

</html>
```

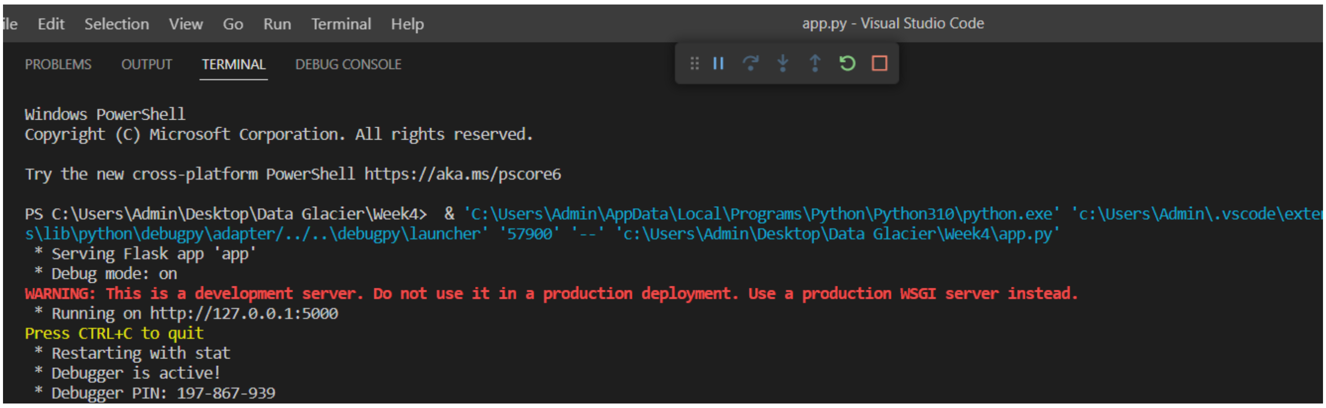
The {{ **prediction_text** }} placeholder you see here is where your output prediction(salary predicted) from the model will be placed in our index.html file.

Style.css

Create a css file and write code to design or style on a webpage.

Running Procedure

Once we have done all of the above, we can start running the API by either executing the command in the terminal or double click *app.py*



```
ile  Edit  Selection  View  Go  Run  Terminal  Help        app.py - Visual Studio Code
PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Admin\Desktop\Data Glacier\Week4>  & 'C:\Users\Admin\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\Admin\.vscode\exte
s\lib\python\debugpy\adapter/../..\debugpy\launcher' '57900' '--' 'c:\Users\Admin\Desktop\Data Glacier\Week4\app.py'
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 197-867-939
```

Now we can open web browser and navigate to http://127.0.0.1:5000/

Enter valid numerical values in all input boxes and hit the button **Predict Selling price**. If everything goes well, you should be able to see the predicted salary value on the HTML page!