Data Glacier Virtual Internship: Week 4-Flask Deployment

Name: Gladys Kalas
Batch Code: LISUM19
Submission date: April 3,2023
Submitted to: Data Glacier

Contents

Abstract

Living in an Information Technology age all sectors of economy are thriving on data. The power of data analytics has enabled us to predict outcomes, with insights drawn from powerful data modeling.

This document constitutes the analysis and prediction, projected based on a file that constitutes Car details obtained from an open public source (Kaggle).
The first section of the analysis involves a summary of the Cars data (car data.csv file). The data file comprises of various data elements in its original state that requires the data to be in a clean state to make the data ready for consumption, processing and modelling the data.

Part A of the process is Data Cleansing. This is achieved by removing data not required for analysis, checking for duplicate records, or deleting any erroneous record.
Part B explains the Linear regression model in Jupyter Notebook.
Part C of the process involves creating a web page using HTML that bridges the information to allow you to easily project onto webpages using Python and Flask.
Part D summarizes the deployment of the model in Flask app.

Part A -Data Cleaning and Summarization:

- The downloaded file is a small and simple data set that comprises 8 features (columns) and around 301 observations(rows) of data. The data in the file gives us information about the Car names, year of purchase, the price of purchase, the distance the car has travelled (in kms),fuel type of the car(ex-Petrol/Diesel), the type of seller and Transmission type. This data is made presentable.

```python
# Importing the Libraries:
import numpy as np
import pandas as pd
from pandas import Series,DataFrame
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import pylab
import statsmodels.api as sm
import statistics
from scipy import stats
import sklearn
import datetime
from sklearn.model_selection import train_test_split
from sklearn.ensemble.forest import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
import warnings
warnings.filterwarnings("ignore", message=r"Passing", category=FutureWarning)
```

```python
Cardata=pd.read_csv("D:\DataGlacier\Week4_Flask\Flask-Deployment\car data.csv",low_memory=False)
Cardata
```

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual
...	...	...	...	...	...	...	...	...
296	city	2016	9.50	11.60	33988	Diesel	Dealer	Manual
297	brio	2015	4.00	5.90	60000	Petrol	Dealer	Manual
298	city	2009	3.35	11.00	87934	Petrol	Dealer	Manual
299	city	2017	11.50	12.50	9000	Diesel	Dealer	Manual
300	brio	2016	5.30	5.90	5464	Petrol	Dealer	Manual

- The average Selling and Present price values are 4.6 and 7.6 lakhs respectively. Average distance travelled is 36947.2 kms
- There are no null values but there are 2 records that are duplicates hence we drop the 2 duplicate observations.

```
: ▶| Cardata.describe()
```

[4]:

	Year	Selling_Price	Present_Price	Kms_Driven
count	301.000000	301.000000	301.000000	301.000000
mean	2013.627907	4.661296	7.628472	36947.205980
std	2.891554	5.082812	8.644115	38886.883882
min	2003.000000	0.100000	0.320000	500.000000
25%	2012.000000	0.900000	1.200000	15000.000000
50%	2014.000000	3.600000	6.400000	32000.000000
75%	2016.000000	6.000000	9.900000	48767.000000
max	2018.000000	35.000000	92.600000	500000.000000

```
▶| Cardata.isnull().sum()
```

```
[3]: Car_Name         0
     Year             0
     Selling_Price    0
     Present_Price    0
     Kms_Driven       0
     Fuel_Type        0
     Seller_Type      0
     Transmission     0
     dtype: int64
```

```
▶| Cardata.duplicated().sum()
```

[5]: 2

```
▶| duplicaterows = Cardata[Cardata.duplicated()]
   duplicaterows
```

[6]:

|	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission
17	ertiga	2016	7.75	10.79	43000	Diesel	Dealer	Manual
93	fortuner	2015	23.00	30.61	40000	Diesel	Dealer	Automatic

```
▶| cardata=(Cardata.drop_duplicates())
```

cardata

|	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual
...	...	...	...	...	...	...	...	...
296	city	2016	9.50	11.60	33988	Diesel	Dealer	Manual
297	brio	2015	4.00	5.90	60000	Petrol	Dealer	Manual
298	city	2009	3.35	11.00	87934	Petrol	Dealer	Manual
299	city	2017	11.50	12.50	9000	Diesel	Dealer	Manual
300	brio	2016	5.30	5.90	5464	Petrol	Dealer	Manual

299 rows × 8 columns

- Features Selling_price and Present_price are likely correlated with each other and are skewed to the right.
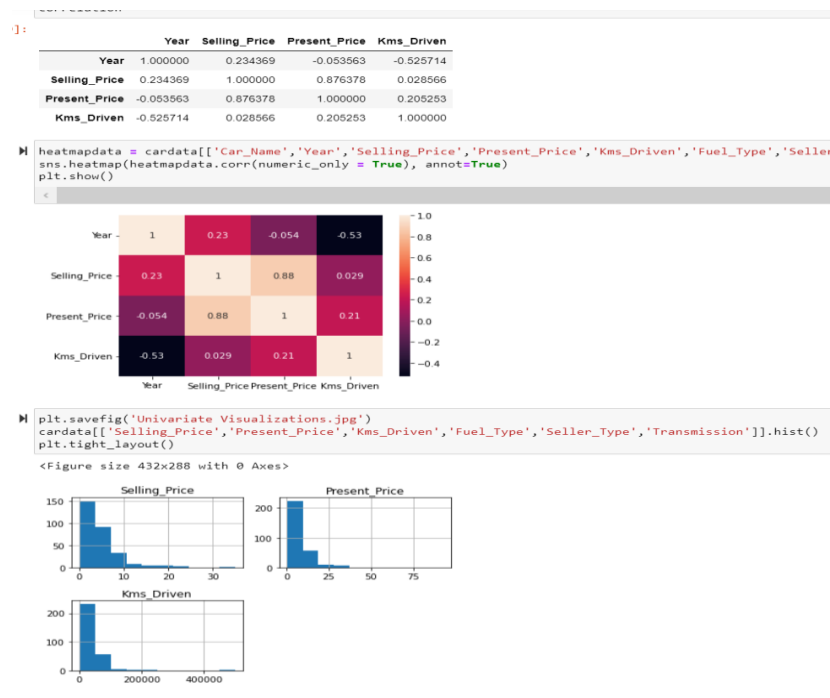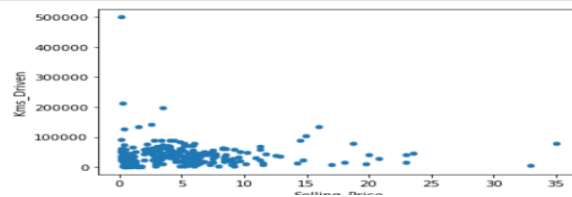
```
cardata.plot(x='Present_Price', y = 'Selling_Price', kind='scatter')
plt.show()
```
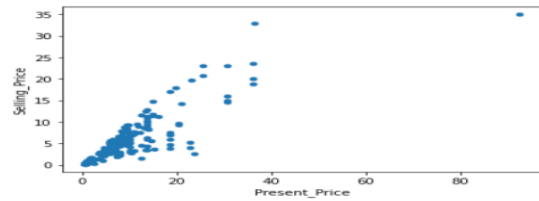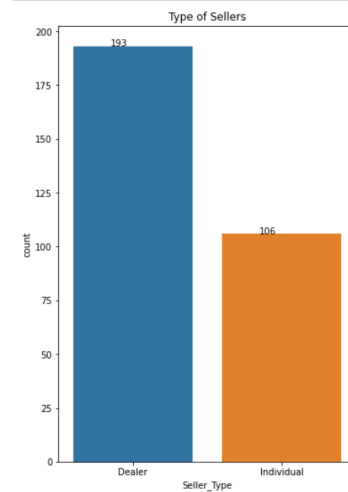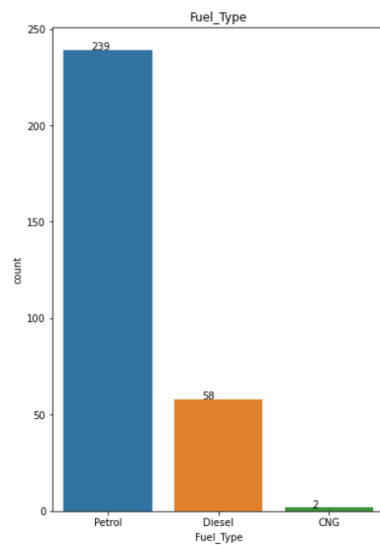


```
cardata.plot(x='Present_Price', y = 'Kms_Driven', kind='scatter')
plt.show()
```
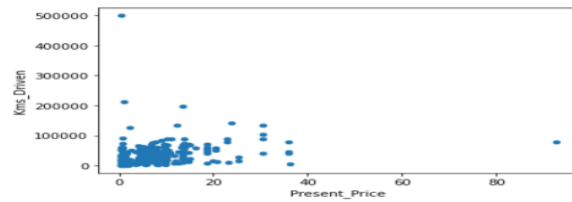


```
cardata.plot(y='Kms_Driven', x = 'Selling_Price', kind='scatter')
plt.show()
```


- There are 3 types of fuel (Petrol/Diesel/CNG) and Petrol is highly popular in the fuel used by these used cars.
- Two types of sellers (Dealers and Individuals) and higher number of used cars are mostly sold by Dealers.

- Among the Transmission types of the car Manual cars are highest and popular than automatic cars.
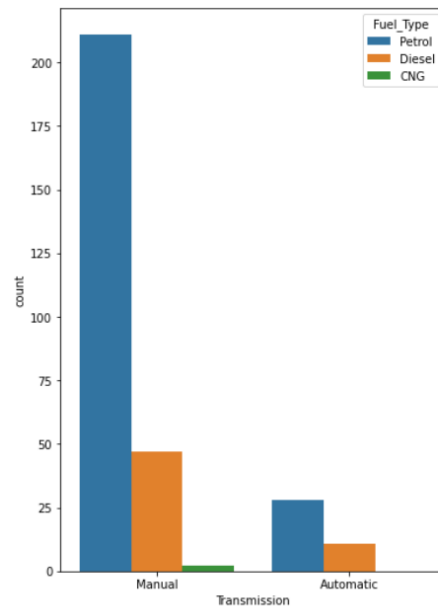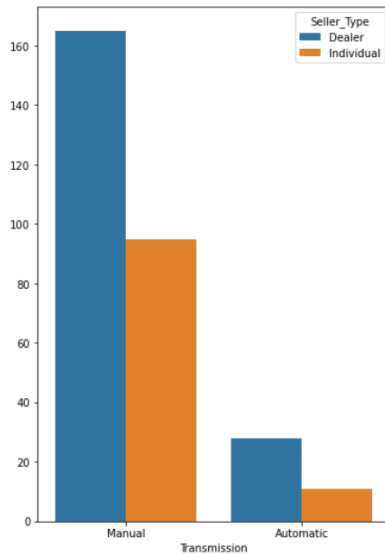
Part B -Create a model using Linear Regression:

- We identify the numerical independent features to project the predicted values using linear regression.
- Divide the data in training and testing sets and check the shapes of train and test sets.
- Fit a linear regression.
- The r-squared value is 0.88077 Meaning that 88.07% of the variance in 'Selling_Price' is explained by the independent features(Kms_Driven, Present_Price, Transmission etc)
- Saving the regression model using Pickle with file name 'Usedcarpriceprediction2.pkl'.

```
cardata1['Transmission']=[1 if v == 'Manual' else 0 for v in cardata1['Transmission']]
```

```
#cardata1 = pd.get_dummies(cardata1 , drop_first=True)
cardata1.head()
```

]:

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Age of car
0	2014	3.35	5.59	27000	Petrol	1	1	9
1	2013	4.75	9.54	43000	Diesel	1	1	10
2	2017	7.25	9.85	6900	Petrol	1	1	6
3	2011	2.85	4.15	5200	Petrol	1	1	12
4	2014	4.60	6.87	42450	Diesel	1	1	9

```
dataset = cardata1.drop(columns=['Selling_Price','Fuel_Type','Seller_Type'])
response = cardata1['Selling_Price']
```

```
X_train, X_test, y_train, y_test = train_test_split(dataset, response, test_size = 0.33,random_state=80)

print("Number transactions X_train dataset: ", X_train.shape)
print("Number transactions y_train dataset: ", y_train.shape)
print("Number transactions X_test dataset: ", X_test.shape)
print("Number transactions y_test dataset: ", y_test.shape)
```

```
Number transactions X_train dataset:  (200, 5)
Number transactions y_train dataset:  (200,)
Number transactions X_test dataset:  (99, 5)
Number transactions y_test dataset:  (99,)
```

```
reg=LinearRegression()
```

```
reg.fit(X_test,y_test)
```

```
LinearRegression()
```

```
predict1=reg.predict(X_test)
```

```
r2_score(predict1 , y_test)
```

```
0.8807722889196092
```

```
import pickle
```

```
newfile = open('Usedcarpriceprediction2.pkl','wb')
pickle.dump(rf,newfile)
```

```
newfile
```

```
<_io.BufferedWriter name='Usedcarpriceprediction2.pkl'>
```

Part C -HTML:

- Now, there are 5 features that HTML will have to extract feature values and save it in a list.
- We create a Templates folder and create a page (user form) to bridge the information from the flask app and the user inputs.
- One of the important criteria while creating inputs is to bear in mind that the number attributes passed from the flask and number of input attributes created in the html should **match,** else there will be an error.

Name

O index.html

```
1   <!DOCTYPE html>
2   <html lang="en">
3
4   <head>
5       <meta charset="UTF-8">
6       <meta name="viewport" content="width=device-width, initial-scale=1.0">
7       <title>Document</title>
8   </head>
9
10  <body>
11
12      <div style="color:□rgb(0,0,0)">
13          <form action="{{ url_for('predict')}}" method="post">
14              <h2>  <font size = "+3" </font>Used Car price prediction</h2>
15              <h3><font size = "+1"</font>Year </h3><input id="first" name="Year" required="required" placeholder="yyyy" >
16              <h3><font size = "+1"</font>Present price </h3><input id="second" name="Present_Price" required="required" placeholder="in lakhs" >
17              <h3><font size = "+1"</font>Kilometers driven</h3><input id="third" name="Kms_Driven" required="required" placeholder="Kms travelled
18              <h3><font size = "+1"</font>Age of the car</h3><input id="fourth" name="Age of car" required="required" placeholder="Car age in year
19              <h3><font size = "+1"</font>Transmission type</h3><input id="sixth" name="Transmission" required="required" placeholder="Manual/Auto
20
21              </select>
22              <br><br><button id="fifth" type="submit">Calculate Car Price</button>
23              <br>
24          </form>
25
26          <br><br><h3><font size = "+3" </font> {{ prediction_text }}<h3>
27      </div>
```

- The HTML code starts with creating a form, creating headers like 'User Car price prediction' and sub headers for input fields.
- There are 5 input fields to the form: Year, Present_Price, Kms_Driven ,Age of the car and Transmission. Each input field has an id associated to make style changes to the fields like font size, styles, padding, color, alignment etc.
- One 'submit' button to calculate the Car price.
- A prediction text to display the final output received from flask app to the user.

```
<style>
    body {
        background-color:  rgba(255,255,128,.5);
        text-align: center;
        padding: 0px;
        border-spacing: 0;
        row-gap: 0;
        column-gap: 0;
        gap:0;

    }

    input::placeholder {
            font-size:15px;
            opacity: 0.5;
            color:  gray;
            font-style: italic;
    }
    }
    #first {

            width: 200px;
            height:30px;
            font-size: +1;
            text-align: center;
            padding: 0%;
            border-spacing: 0;
    }
    #second {
        height:30px;
        width:200px;
        font-size: +1;
        text-align: center;
        padding: 0%;
        border-spacing: 0;
        row-gap: 5;
        column-gap: 5;
    }
```
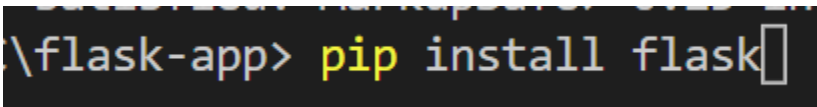
```css
#third {

    height: 30px;
    width:200px;
    font-size: +1;
    text-align: center;
    padding: 0%;
    border-spacing: 0;
    column-gap: 5;
    row-gap: 5;
    }
 #fourth  {
    height: 30px;
    width:200px;
    font-size: +1;
    text-align: center;
    padding: 0%;
    border-spacing: 0;
}
#fifth {
    width: 260px;
    height: 50px;
    text-align: center;
    font-size: +2;
    margin-top:30px;
    font:bolder;
    font-weight: 500px;
    padding: 0%;
    border-spacing: 5px;


}
#fifth:hover {
    background-color: white
}
#sixth  {
    height: 30px;
    width:200px;
    font-size: +1;
    text-align: center;
    padding: 0%;
    border-spacing: 0;
}
```

Part D -Summarizes deployment of the model on flask app:

Steps:
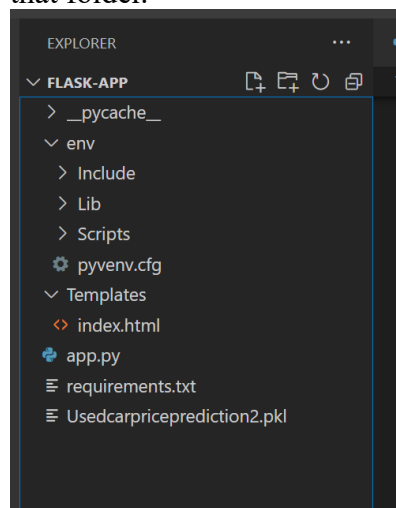- The first step is to ensure that Flask is installed on the system.

```
\flask-app> pip install flask
```

```
Requirement already satisfied: flask in c:\programdata\anaconda3\lib\site-packages (1.1.2)
Requirement already satisfied: itsdangerous>=0.24 in c:\programdata\anaconda3\lib\site-packages (from flask) (1.1.0)
Requirement already satisfied: Werkzeug>=0.15 in c:\programdata\anaconda3\lib\site-packages (from flask) (1.0.1)
Requirement already satisfied: Jinja2>=2.10.1 in c:\programdata\anaconda3\lib\site-packages (from flask) (2.11.2)
Requirement already satisfied: click>=5.1 in c:\programdata\anaconda3\lib\site-packages (from flask) (7.1.2)
Requirement already satisfied: MarkupSafe>=0.23 in c:\programdata\anaconda3\lib\site-packages (from Jinja2>=2.10.1->flask) (1.1.1)
```
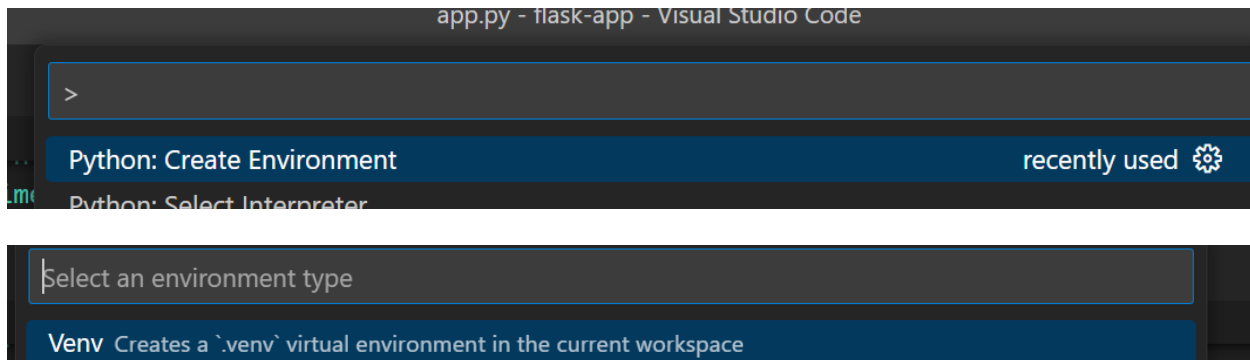
- To work with flask in VSC, I created a folder on the files system where flask is installed by the name "flask-app".
- Now when we run a flask code in the VS it will throw errors because there is no connection between the files on the system and the app.
- This prompts to create a virtual environment for the flask project on the system.

__pycache__	3/2:
env	3/2.
Templates	3/2.
app.py	3/2:
requirements.txt	3/2:
Usedcarpriceprediction2.pkl	3/2:

- Launch VSC, open the folder "flask-app", this will load all the files present in that folder.

```
EXPLORER                          ...
∨ FLASK-APP
  > __pycache__
  ∨ env
    > Include
    > Lib
    > Scripts
    ⚙ pyvenv.cfg
  ∨ Templates
    <> index.html
  🐍 app.py
  ≡ requirements.txt
  ≡ Usedcarpriceprediction2.pkl
```

- A requirements.txt file in Python lets you keep track of the modules and packages used in the projects Simply put, a requirements.txt file is simply a .txt file that This makes it easier to install the required packages.
- However, this caused me a huge issue when numpy was upgraded to 1.24. and faced errors, hence I had to downgrade the numpy version to 1.23.1 to resolve the issues. So, it's important to install and import the right versions.
- Next, I opened the Command Palette (from the View tab) and selected Python: Create Environment and then selected the Venv to create a virtual environment.
- The next step after creating the virtual environment is to start a new Terminal window that activates the virtual environment
- Ensure flask is installed in the virtual environment (python -m pip install flask).

app.py - flask-app - Visual Studio Code

```
>
```

Python: Create Environment   recently used ⚙

Python: Select Interpreter

Select an environment type

Venv Creates a `.venv` virtual environment in the current workspace

- Checked the working of the flask app through a simple program
  - Import Flask
  - Create the 'app.py' file
- The app.py returns content like a simple string in the below code it is Hello,World! , and the Flask's app.route decorator is used to map the URL route / to the function (at the given port of the server).

```python
app.py > ...
 1    from flask import Flask
 2    app = Flask(__name__)
 3
 4
 5    @app.route("/")
 6    def home():
 7        return "Hello, World!"
 8
 9    if __name__=="__main__":
10        app.run(port=5000,debug=True)
11
12
```

```
C:\Users\MammaPC\flask-app> python -m flask run
Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
Debug mode: off
Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

The output:
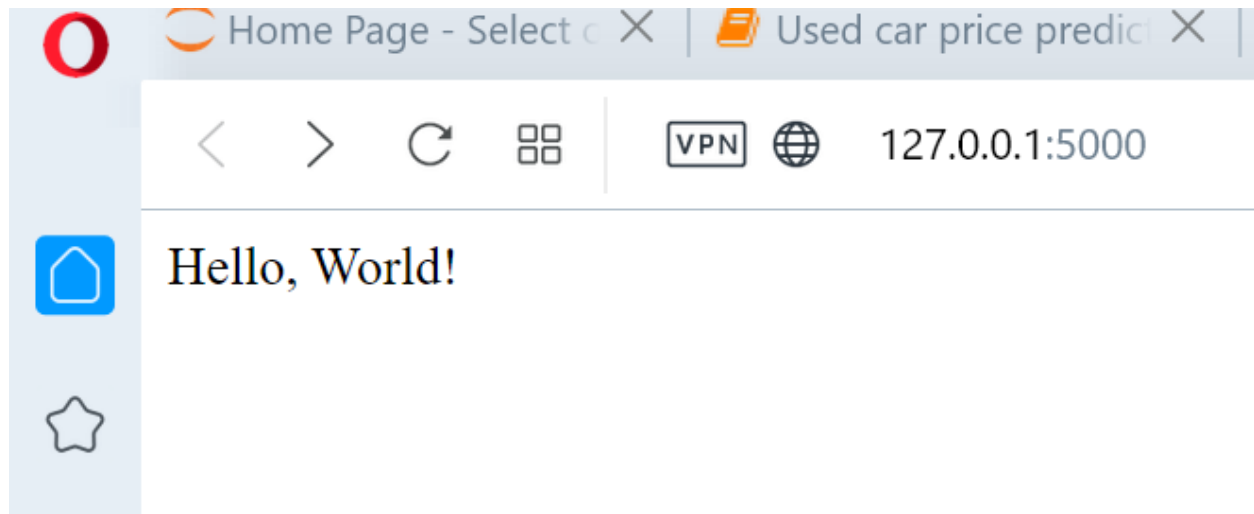
- Now that flask is up and running, the used car price prediction model is deployed on the flask app.

Part D.a -Used Car price prediction: Deployment on Flask:
Steps:

- Import libraries
- Load the flask instance class app.py and the python model through pickle.load method, passing the file object to be read from.
- request- contains all the data sent from the Client to Server. The data is retrieved from the Usedcarpriceprediction2.pkl file loaded earlier.
- Render_template- The index function renders a template index.html and hence we see the result in the browser.

```python
from flask import Flask, render_template, request
import requests
import pickle
import numpy as np
import sklearn
import datetime
from sklearn.preprocessing import StandardScaler


app = Flask(__name__)
model = pickle.load(open('Usedcarpriceprediction2.pkl', 'rb'))
```

- Then created two app.route methods:
- home() -which when run, establishes a connection to the server which displays the index.html webpage to the user.
- predict()-is called when input is fed to the index.html from the user and the final predict button is pressed to display the car prices.

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

```python
@app.route("/predict", methods=['POST'])
def predict():

    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)
        Age_of_car=int(request.form['Age of car'])

        Transmission=request.form['Transmission']
        if(Transmission=='Manual'):
            Transmission=1
        else:
            Transmission=0

        prediction=model.predict([[Year,Present_Price,Kms_Driven2,Age_of_car,Transmission]])
        output=round(prediction[0],2)

    return render_template('index.html',prediction_text="The price of the car is {} lakhs".format(output))
    return render_template('index.html')

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

- The data from request can be recovered using the POST methods. POST is used when the application expects the users input to be received by HTTP request.
- The predict() method (/predict) extracts features of the car data those are 5 features-Year, Present_Price, Kms_Driven, Age of car and Transmission and using these we will predict the used car price.
- Using these values rendering the template is in use to predict the car values.
- Save the app.py and run the code.

```
Use a production wSGI server instead.
* Debug mode: on
* Restarting with windowsapi reloader
* Debugger is ...
* Debugger PI  Follow link (ctrl + click)
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Part D.a -Output:

127.0.0.1:5000/predict

Used Car price prediction

Year

yyyy

Present price

in lakhs

Kilometers driven

Kms travelled

Age of the car

Car age in years

Transmission type

Manual/Automatic

Calculate Car Price

The price of the car is 6.52 lakhs

127.0.0.1:5000/predict

Used Car price prediction

Year

2016

Present price

7.8

Kilometers driven

24000

Age of the car

7

Transmission type

Automatic

Calculate Car Price

Used Car price prediction

Part E -Deployment a local app on PythonAnywhere:

 Steps:

The process of deploying a local web app on PythonAnywhere involves three main steps:
1. Uploading the code and necessary files to Python anywhere
2. Setting up a virtual environment
3. Configuring the application using WSGI

1.Uploading code via Github:

The code and the required files pertaining the used car price prediction were stored on cloud system Github and extracted to PythonAnywhere by the below steps:

 Consoles > Bash console > git clone (link to the git repo)

```
08:28 ~ $ git clone https://github.com/onlygladys/Flask.git
Cloning into 'Flask'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (7/7), 255.43 KiB | 920.00 KiB/s, done.
08:30 ~ $ $ mkvirtualenv myvirtualenv --python=/usr/bin/python3.7.9
bash: $: command not found
08:47 ~ $   mkvirtualenv myvirtualenv --python=/usr/bin/python3.10█
```

Enter new directory name	New directory		Enter new file name, eg hello.py	

.cache/	🗑
.local/	🗑
.virtualenvs/	🗑 🗑 🗑
Flask/	🗑
mysite/	🗑

📄 .bashrc	⬇ ✏ 🗑	2023-04-01 05:10	560 bytes
📄 .gitconfig	⬇ ✏ 🗑	2023-04-01 05:10	266 bytes
📄 .profile	⬇ ✏ 🗑	2023-04-01 05:10	79 bytes
📄 .pythonstartup.py	⬇ ✏ 🗑	2023-04-01 05:10	77 bytes
📄 .vimrc	⬇ ✏ 🗑	2023-04-01 05:10	4.6 KB
📄 README.txt	⬇ ✏ 🗑	2023-04-01 05:10	232 bytes

⊕ Upload a file

100MiB maximum size

2. Setting a virtual environment:
 PythonAnywhere has many pre-installed modules and libraries but it may not be the same version as the as the ones needed for our web app.Hence, a virtual environment is set up to the required libraries and dependencies.

```
08:28 ~ $ git clone https://github.com/onlygladys/Flask.git
Cloning into 'Flask'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (7/7), 255.43 KiB | 920.00 KiB/s, done.
08:30 ~ $ $ mkvirtualenv myvirtualenv --python=/usr/bin/python3.7.9
bash: $: command not found
08:47 ~ $   mkvirtualenv myvirtualenv --python=/usr/bin/python3.7.9
FileNotFoundError: [Errno 2] No such file or directory: '/usr/bin/python3.7.9'
08:48 ~ $ mkvirtualenv myvirtualenv --python=/usr/bin/python3.7
created virtual environment CPython3.7.13.final.0-64 in 24797ms
  creator CPython3Posix(dest=/home/gkalas/.virtualenvs/myvirtualenv, clear=False, no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=/home/gkalas/.local/share/virtualenv)
    added seed packages: pip==22.1.2, setuptools==62.6.0, wheel==0.37.1
  activators BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator
virtualenvwrapper.user_scripts creating /home/gkalas/.virtualenvs/myvirtualenv/bin/predeactivate
virtualenvwrapper.user_scripts creating /home/gkalas/.virtualenvs/myvirtualenv/bin/postdeactivate
virtualenvwrapper.user_scripts creating /home/gkalas/.virtualenvs/myvirtualenv/bin/preactivate
virtualenvwrapper.user_scripts creating /home/gkalas/.virtualenvs/myvirtualenv/bin/postactivate
virtualenvwrapper.user_scripts creating /home/gkalas/.virtualenvs/myvirtualenv/bin/get_env_details
(myvirtualenv) 08:51 ~ $
```

```
21:09 ~/mysite $ pip3 install -r requirements.txt --user
Looking in links: /usr/share/pip-wheels
Requirement already satisfied: certifi==2022.12.7 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 1)) (2022.12.7)
Requirement already satisfied: charset-normalizer==3.1.0 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 2)) (3.1.0)
Requirement already satisfied: click==8.1.3 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 3)) (8.1.3)
Requirement already satisfied: colorama==0.4.6 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 4)) (0.4.6)
Requirement already satisfied: Flask==2.2.3 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 5)) (2.2.3)
Requirement already satisfied: idna==3.4 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 6)) (3.4)
Requirement already satisfied: importlib-metadata==6.1.0 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 7)) (6.1.0)
Requirement already satisfied: itsdangerous==2.1.2 in /usr/local/lib/python3.10/site-packages (from -r requirements.txt (line 8)) (2.1.2)
Requirement already satisfied: Jinja2==3.1.2 in /usr/local/lib/python3.10/site-packages (from -r requirements.txt (line 9)) (3.1.2)
Requirement already satisfied: joblib==1.2.0 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 10)) (1.2.0)
Requirement already satisfied: MarkupSafe==2.1.2 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 11)) (2.1.2)
Requirement already satisfied: numpy==1.21.6 in /usr/local/lib/python3.10/site-packages (from -r requirements.txt (line 12)) (1.21.6)
Requirement already satisfied: package-name==0.1 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 13)) (0.1)
Requirement already satisfied: requests==2.28.2 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 14)) (2.28.2)
Requirement already satisfied: scikit-learn==1.0.2 in /usr/local/lib/python3.10/site-packages (from -r requirements.txt (line 15)) (1.0.2)
Requirement already satisfied: scipy==1.7.3 in /usr/local/lib/python3.10/site-packages (from -r requirements.txt (line 16)) (1.7.3)
Requirement already satisfied: sklearn==0.0.post1 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 17)) (0.0.post1)
Requirement already satisfied: threadpoolctl==3.1.0 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 18)) (3.1.0)
Requirement already satisfied: typing_extensions==4.5.0 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 19)) (4.5.0)
Requirement already satisfied: urllib3==1.26.15 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 20)) (1.26.15)
Requirement already satisfied: Werkzeug==2.2.3 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 21)) (2.2.3)
Requirement already satisfied: zipp==3.15.0 in /home/gkalas/.local/lib/python3.10/site-packages (from -r requirements.txt (line 22)) (3.15.0)
```

3.Configuring the application using WSGI:
To get the app working on cloud, there is a configuration needed, a WSGI file that imports the app, as a Python variable.

```
import sys

# add your project directory to the sys.path
project_home = '/home/gkalas/mysite'
if project_home not in sys.path:
    sys.path = [project_home] + sys.path

# import flask app but need to call it "application" for WSGI to work
from app import app as application  # noqa
```

```
import datetime
import pickle
import requests
import sklearn
from flask import Flask, render_template, request
import numpy as np
import os




app = Flask(__name__)

my_dir=os.getcwd()
pickle_file_path=os.path.join(my_dir,'mysite/Usedcarpriceprediction2.pkl')
with open (pickle_file_path,'rb') as pickle_file:
    model = pickle.load(pickle_file)

x =datetime.datetime.now()
@app.route('/')
def home():
    return render_template('index.html')


@app.route("/predict", methods=['POST'])
def predict():

    if request.method == 'POST':
        Year = int(request.form['Year'])
        Present_Price=float(request.form['Present_Price'])
        Kms Driven=int(request.form['Kms Driven'])
```

4. Deployment:

Create new web app   ✖

Your web app's domain name

Your account doesn't support custom domain names, so your PythonAnywhere web app will live at `gkalas.pythonanywhere.com`.

Want to change that? Upgrade now!

Otherwise, just click "Next" to continue.

Cancel   « Back   Next »

Hour (previous hour)   4   (0)

Want some more data? Paying accounts get pretty charts ;-)

Code:
What your site is running.

Source code:	/home/gkalas/mysite	➜Go to directory
Working directory:	/home/gkalas/	➜Go to directory
WSGI configuration file:	/var/www/gkalas_pythonanywhere_com_wsgi.py	
Python version:	3.7 ✏	

Virtualenv:

Part E.a -Output:

Configuration for gkalas.pythonanywhere.com

Reload:

C Reload gkalas.pythonanywhere.com

C 88   VPN ⚠ Not secure   gkalas.pythonanywhere.com

Used Car price prediction

Year

2018

Present price

7

Kilometers driven

2000

Age of the car

6

Transmission type

Manual

Calculate Car Price

Part F-POSTMAN:
 Testing the API in POSTMAN