Deployment of Machine Learning Model to Amazon Web Services

Preface

This Document is intended to hell all Data Scientists. Document cover step by step guide for creating ML model from scratch. Creating Flask Application and Deploying Flask App in AWS Cloud. This book uses a dataset from Kaggle to predict the chances of the admission of a student into foreign universities based on different evaluation criteria. This book tries to explain the concepts simply, extensively, and thoroughly to approach the problem from scratch and then its deployment to a cloud environment.

Happy Learning!

Index

The Problem Statement

The goal here is to find the chance of admission of a candidate based on his/her GRE Score (out of 340), TOEFL Score (out of 120), rating of the University (out of 5) in which he/she is trying to get admission, Strength of the SOP (out of 5), strength of the Letter of Recommendation (out of 5), CGPA (out of 10) and the research experience (0 or 1).

Pre-requisites

- Basic Knowledge
  I. Python Machine Learning Libraries.
  II. HTML and CSS (Java Script Optional).
  III. Flask Framework.

- Software/Packages
  I. AWS Account.
  II. Python IDE (I have used PyCharm).

Python Implementation

Below you will see snap shots of my Jupyter Notebook I have used Markdown for better understanding, you can download my notebook from GitHub and use for reference.

Linear Regression (Cloud Implementation)

Importing Necessary libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model  import Ridge,Lasso,RidgeCV, LassoCV, ElasticNet, ElasticNetCV, LinearRegression
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
import pickle

%matplotlib inline
```

Reading the data file

In [2]:
```
data= pd.read_csv('Admission_Prediction.csv')
```

Checking the first five rows from the dataset

In [3]:
```
data.head()
```

Out[3]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337.0	118.0	4.0	4.5	4.5	9.65	1	0.92
1	2	324.0	107.0	4.0	4.0	4.5	8.87	1	0.76
2	3	NaN	104.0	3.0	3.0	3.5	8.00	1	0.72
3	4	322.0	110.0	3.0	3.5	2.5	8.67	1	0.80
4	5	314.0	103.0	2.0	2.0	3.0	8.21	0	0.65

In [4]:
```
data.describe(include='all')
```

Out[4]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	500.000000	485.000000	490.000000	485.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	250.500000	316.558763	107.187755	3.121649	3.374000	3.48400	8.576440	0.560000	0.72174
std	144.481833	11.274704	6.112899	1.146160	0.991004	0.92545	0.604813	0.496884	0.14114
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000
25%	125.750000	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	250.500000	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	375.250000	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	500.000000	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         500 non-null    int64  
 1   GRE Score          485 non-null    float64
 2   TOEFL Score        490 non-null    float64
 3   University Rating  485 non-null    float64
 4   SOP                500 non-null    float64
 5   LOR                500 non-null    float64
 6   CGPA               500 non-null    float64
 7   Research           500 non-null    int64  
 8   Chance of Admit    500 non-null    float64
dtypes: float64(7), int64(2)
memory usage: 35.3 KB
```

As we have very less null data lets deal with them using basic mean and mode method.

In [6]:
```
data['University Rating']=data['University Rating'].fillna(data['University Rating'].mode()[0])
data['GRE Score']=data['GRE Score'].fillna(data['GRE Score'].mean())
data['TOEFL Score'] = data['TOEFL Score'].fillna(data['TOEFL Score'].mean())
```

In [7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         500 non-null    int64  
 1   GRE Score          500 non-null    float64
 2   TOEFL Score        500 non-null    float64
 3   University Rating  500 non-null    float64
 4   SOP                500 non-null    float64
 5   LOR                500 non-null    float64
 6   CGPA               500 non-null    float64
 7   Research           500 non-null    int64  
 8   Chance of Admit    500 non-null    float64
dtypes: float64(7), int64(2)
memory usage: 35.3 KB
```

Now the data looks good and there are no missing values. Also, the first cloumn is just serial numbers, so we don' need that column. Let's drop it from data and make it more clean.

In [8]:
```
data= data.drop(columns=  ['Serial No.'])
data.head()
```

Out[8]:

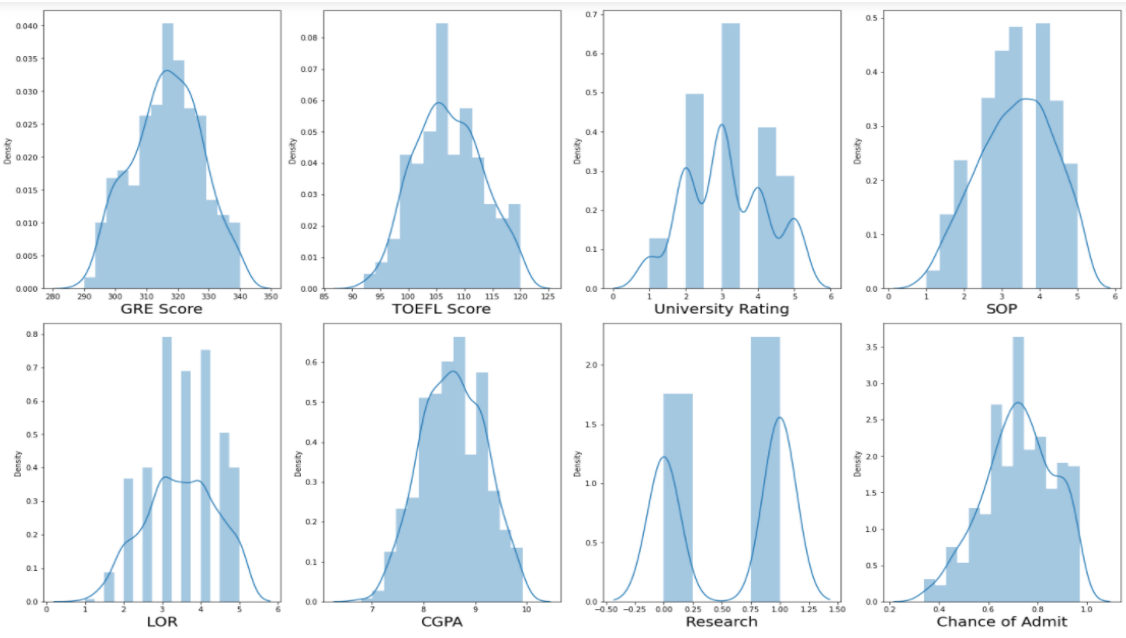
	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337.000000	118.0	4.0	4.5	4.5	9.65	1	0.92
1	324.000000	107.0	4.0	4.0	4.5	8.87	1	0.76
2	316.558763	104.0	3.0	3.0	3.5	8.00	1	0.72
3	322.000000	110.0	3.0	3.5	2.5	8.67	1	0.80
4	314.000000	103.0	2.0	2.0	3.0	8.21	0	0.65

Let's visualize the data and analyze the relationship between independent and dependent variables:

let's see how data is distributed for every column

In [9]:
```
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in data:
    if plotnumber<=16 :
        ax = plt.subplot(4,4,plotnumber)
        sns.distplot(data[column])
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.tight_layout()
```



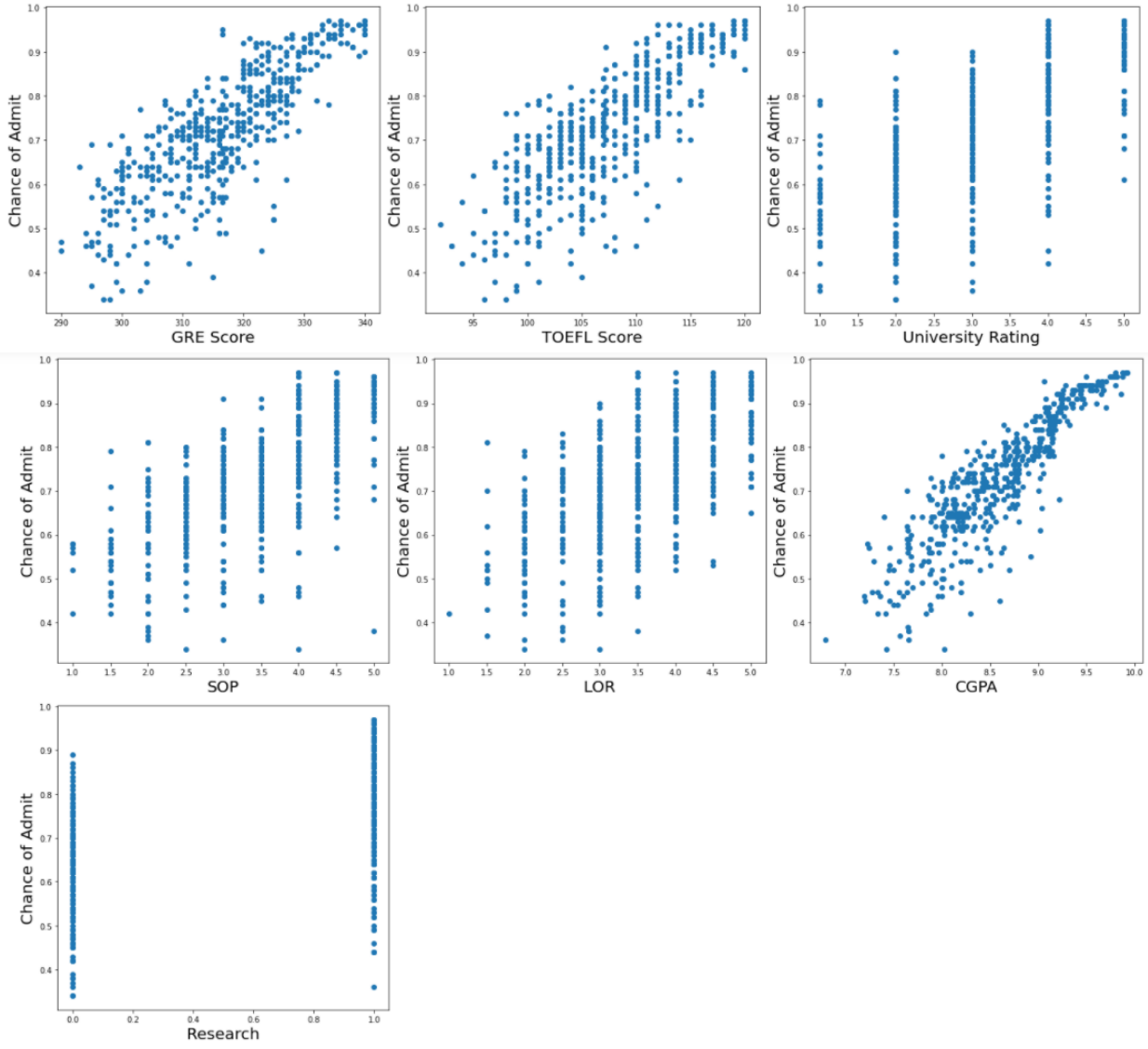
The data distribution looks decent enough and there doesn't seem to be any skewness. Great let's go ahead!

Let's observe the relationship between independent variables and dependent variable.

In [10]:
```
y = data['Chance of Admit']
X =data.drop(columns = ['Chance of Admit'])
```

In [11]:
```
plt.figure(figsize=(20,30), facecolor='white')
plotnumber = 1

for column in X:
    if plotnumber<=15 :
        ax = plt.subplot(5,3,plotnumber)
        plt.scatter(X[column],y)
        plt.xlabel(column,fontsize=20)
        plt.ylabel('Chance of Admit',fontsize=20)
    plotnumber+=1
plt.tight_layout()
```



Great, the relationship between the dependent and independent variables look fairly linear. Thus, our linearity assumption is satisfied.

Let's move ahead and check for multicollinearity.

In [12]:
```
scaler =StandardScaler()

X_scaled = scaler.fit_transform(X)
```

In [13]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
variables=X_scaled
```

- we create a new data frame which will include all the VIFs
- note that each variable has its own variance inflation factor as this measure is variable specific (not model specific)
- we do not include categorical values for mulitcollinearity as they do not provide much information as numerical ones do

In [14]:
```
vif=pd.DataFrame()
```

- here we make use of the variance_inflation_factor, which will basically output the respective VIFs

In [15]:
```
vif["VIF"]=[variance_inflation_factor(variables,i)for i in range(variables.shape[1])]
vif["Feautres"]=X.columns
```

In [16]:
```
vif
```

Out[16]:

	VIF	Feautres
0	4.152735	GRE Score
1	3.793345	TOEFL Score
2	2.517272	University Rating
3	2.776393	SOP
4	2.037449	LOR
5	4.654369	CGPA
6	1.459411	Research

Here, we have the correlation values for all the features. As a thumb rule, a VIF value greater than 5 means a very severe multicollinearity. We don't any VIF greater than 5 , so we are good to go.

Great. Let's go ahead and use linear regression and see how good it fits our data. But first. let's split our data in train and test.

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

Linear Regression

In [18]:
```
regression = LinearRegression()
```

In [19]:
```
regression.fit(X_train,y_train)
```

Out[19]:
```
LinearRegression()
```

Saving the Model to the local file

In [20]:
```
filename = 'finalized_model.pickle'
pickle.dump(regression, open(filename, 'wb'))
```

This is demo of how we are going to give data to our model, and model will give prediction!

In [21]:
```
loaded_model = pickle.load(open(filename, 'rb'))
a=loaded_model.predict(scaler.transform([[300,110,5,5,5,10,1]]))
a
```

Out[21]:
```
array([-0.89970354])
```

Result

In [22]:
```
regression.score(X_test,y_test)
```

Out[22]:
```
0.8215045514507888
```

We got regression score as 82% which is pretty good! I have tried to keep Python Implementation basic as our main focus is to check for implementation in Cloud (AWS).
There are few things in notebook which might be new to you! Like Pickle etc Don't worry I will explain you.

Once we satisfy with our model we can save the model using pickle.

Python **pickle** module is used for serializing and de-serializing a Python object structure. Pickling is a way to convert a python object (list, dict, etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script.

```
:  filename = 'finalized_model.pickle'
   pickle.dump(regression, open(filename, 'wb'))
```

So here we are creating a file name finalized_model.pickle and storing regression (Model Variable) in that file . Once this command is executed a file name finalized_model with pickle format will be stored in you local folder. You can use that file in your flask Application.

Demo

```
In [21]: loaded_model = pickle.load(open(filename, 'rb'))
         a=loaded_model.predict(scaler.transform([[300,110,5,5,5,10,1]]))
         a
Out[21]: array([-0.89970354])
```

Above Image shows demo of how we can use stored model and get prediction.
We are manually giving score for all our features and getting Score 89%.

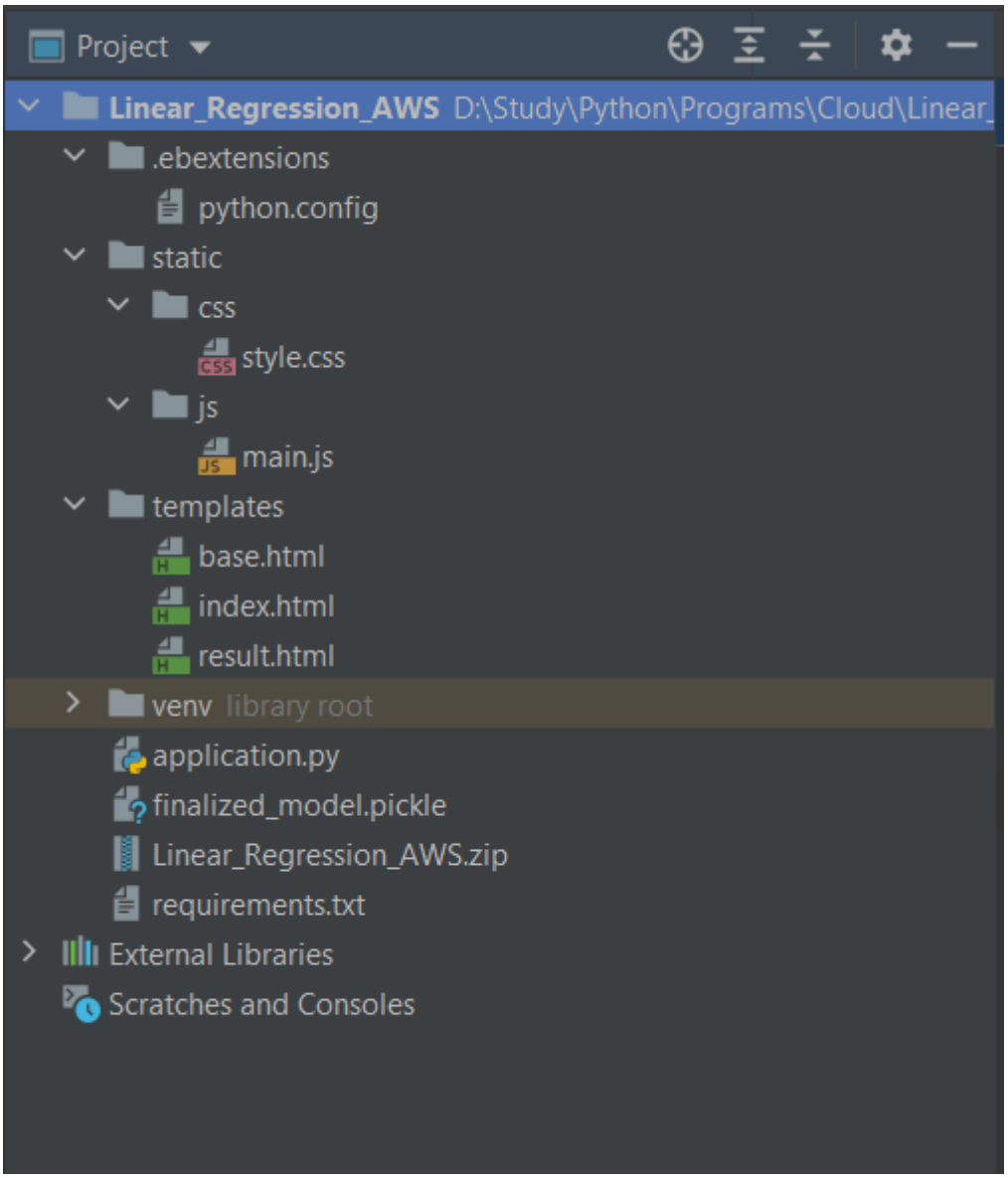
Now we are ready to create a Web App.

Flask App

Flask will help python to interact with Html, CSS and Java Script and work as a Web Application.

Flow of Our Flask App will be

You Flask Application should consists of all files mentioned bellow in Image



Note: This Files are totally optional

1) VENV This file was created by PyCharm while I was installing required package. This can be totally optional while Zipping the Project in later state. (This File is high in size which may cause delay in deployment).
2) main.js this file is also Optional as I have used to apply validation on my form such as Range and not null.
3) Style.css is CSS file used for styling my Front End also can be kept as optional.
   Rest All file all Mandatory missing of any single file can cause failure in deployment.

Now let's Explore all files in details

Base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device=width, initial-scale=1.0 ">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link rel="stylesheet" href="{{url_for('static', filename='css/main.css')}}">
    {% block head %}{% endblock %}
</head>
<body>
    {% block body %}{% endblock %}
</body>
</html>
```

This file If just Base HTML page which we are going to extend and use in index.html

Index.html

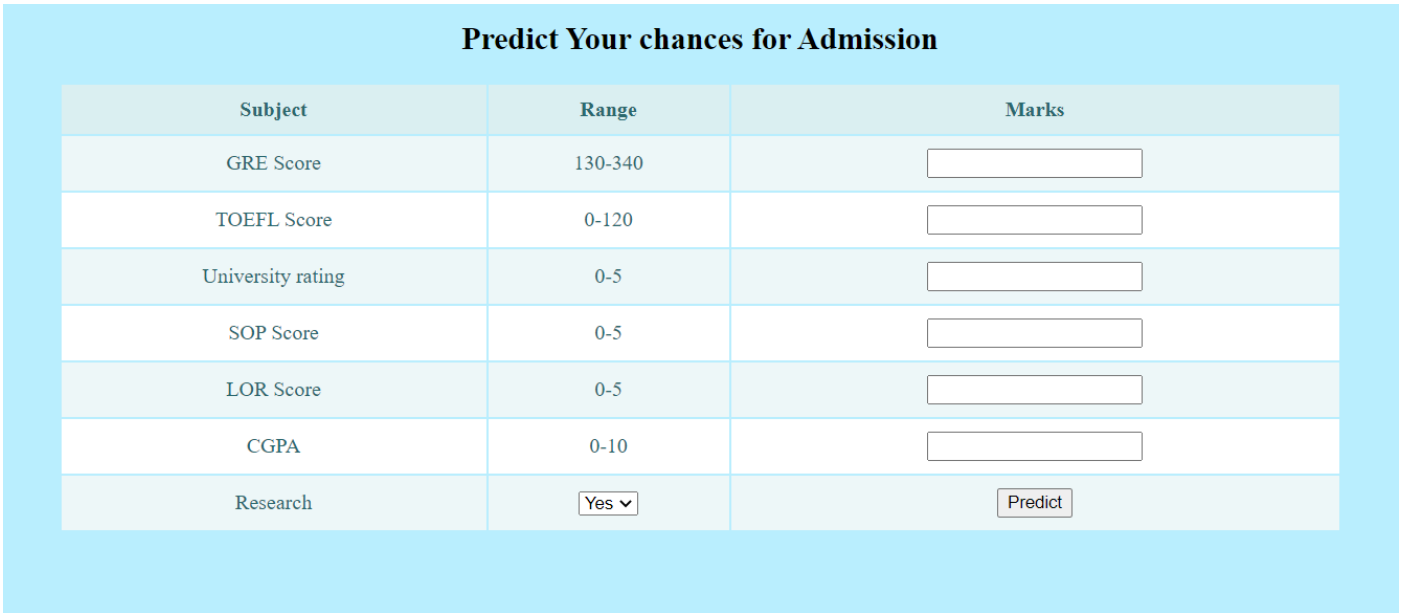
```
<div class="container">
    <h2 style="text-align: center">Predict Your chances for Admission</h2>

    <div class="form">
        <table>
            <tr>
                <th><b>Subject </b></th>
                <th><b>Range</b></th>
                <th><b>Marks</b></th>

            </tr>
            <form name="myForm" action="/predict" onsubmit="return validateForm()"
method="POST">
                <tr><td>GRE Score</td>
                    <td>130-340</td>
                    <td><center><input type="number" name="gre_score"
id="gre_score"></center></td>
                </tr>
                <tr><td>TOEFL Score</td>
                    <td>0-120</td>
                    <td><center> <input type="number" name="toefl_score"
id="toefl_score"></center></td>
                </tr>
                <tr><td>University rating</td>
```

```
                    <td>0-5</td>
                    <td><center>  <input type="number" name="university_rating"
id="university_rating" step="any"></center></td>
                </tr>
                <tr><td>SOP Score</td>
                    <td>0-5</td>
                    <td><center>   <input type="number" name="sop" id="sop"
step="any"></center></td>
                </tr>
                <tr><td>LOR Score</td>
                    <td>0-5</td>
                    <td><center> <input type="number" name="lor" id="lor"
step="any"></center></td>
                </tr>
                <tr><td>CGPA</td>
                    <td>0-10</td>
                    <td><center>  <input type="number" name="cgpa" id="cgpa"
step="any"></center></td>
                </tr>
                <tr><td>Research</td>
                    <td><select name="research"id="research">
                        <option value="yes">Yes</option>
                        <option value="no">No</option>
                        </select></td>
                    <td><center> <input type="submit" value="Predict"></center></td>
                </tr>
            </form>
        </table>
    </div>
</div>
{% endblock %}
```

This Index.html will result as:



There are 2 more files (Style.css and Main.js) which I am not uploading in my document but they are available in my GitHub. You can go ahead and view these files there.

Index.html is a page containing Form where Student will enter their Scores. Further Data will be Extracted by Python Script (Application.py) and then give data to model for prediction.

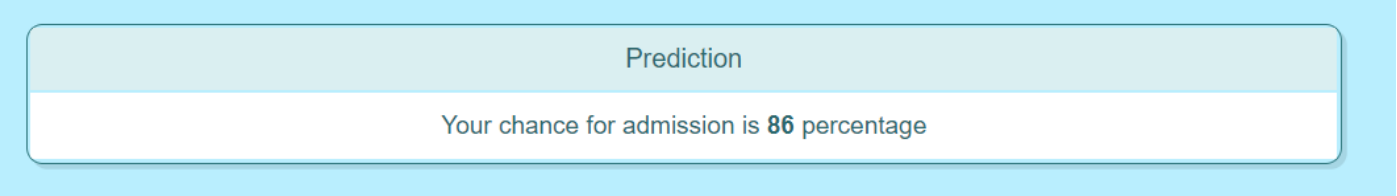
Result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title> Review Page </title>
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/normalize/5.0.0/normalize.min.css">
    <link rel="stylesheet" href="./style.css">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>

    <div class="table-users">
        <table>
            <th>
        <div class="header">Prediction </div>
        </th>
            <tr><td><center>Your chance for admission is <b> {{prediction}}
</b>percentage</center></td></tr>

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

Result.html will display predicted value from model as below



Application.py

```
# Importing# Importing Necessary Libraries
from flask import Flask, render_template, request, jsonify
from flask_cors import CORS, cross_origin
import pickle

# Initializing a flask app
application = Flask(__name__)

# When first time application is called it will render to index.html page
@application.route('/', methods=['GET'])
@cross_origin()
def homepage():
    return render_template("index.html")

# When Post Method is initiated it will call index function
```

```
# Index function will save data provided my user in variables


@application.route('/predict', methods=['POST', 'GET'])
@cross_origin()
def index():
    if request.method == 'POST':
        try:
            gre_score = float(request.form['gre_score'])
            toefl_score = float(request.form['toefl_score'])
            university_rating = float(request.form['university_rating'])
            sop = float(request.form['sop'])
            lor = float(request.form['lor'])
            cgpa = float(request.form['cgpa'])
            is_research = request.form['research']
            if (is_research == 'yes'):
                research = 1
            else:
                research = 0
            filename = 'finalized_model.pickle'
            # loading the model file
            loaded_model = pickle.load(open(filename, 'rb'))
            # prediction using loaded model
            prediction = loaded_model.predict([[gre_score, toefl_score,
university_rating, sop, lor, cgpa, research]])
            print('Prediction is ', prediction)
            # showing the prediction results in a UI
            return render_template('result.html', prediction=round(100*prediction[0]))
        except Exception as e:
            print('The Exception message is : ', e)
            return 'Something is Wrong '

    else:
        return render_template("index.html")


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

Application.py is the python script which initialize Flask App and help to render it from one page to another. It help in building a dynamic Web Application.

Creating a Flask App

1) Import all necessary libraries
   In this app we have used Flask, Flask-Cors and Pickle

2) Initializing a flask app
   application = Flask(__name__)
   change main class details with flask application name you have created while initializing.
   if __name__ == '__main__':
   application.run(debug=True)

   Note: For Deploying your application In AWS it in mandatory to have name of your app to be “application” or “app” So I would suggest to keep your Python script name as application.py and Flask variable name application.

3) Now we create @Flask.route which help application to Render Html pages
   @application.route('/', methods=['GET'])
   @cross_origin()

```
def homepage():
    return render_template("index.html")

@application.route('/predict', methods=['POST', 'GET'])
@cross_origin()
def index():
```

This Index Function will be called when predict button is clicked on form.
Index Function → Collects Data from Form →Store in an variables → Load Model Machin Learning Model from Pickle File → Provide Data via stored variable → Takes Result from Model and Send it to Result. Page.

Python Config File

We need to create a folder with name ".ebextensions" in our project .Later We need to create Python.config file which will contain below mention data.

```
option_settings:
  "aws:elasticbeanstalk:container:python":
    WSGIPath: application:application
```

This help AWS to deploy application on cloud

Note: This Forma is very important it should be as it is, Any change is format/syntax can cause failure for deployment.

Requirement.txt

This file has all libraries which you have used in your application. This File is also mandatory , AWS will validate first if all packages are install mentioned in file and later will go ahead with deployment. If any Missing packages consist in AWS Environment AWS will first install Packages.

Creating File Is very east with one Step

- ➔ Open Python Command Prompt Like Anaconda Prompt or any other and go to Folder where you have created Project using cd command.
- ➔ Type *pip freeze > requirements.txt* it will create a file with name requirement in your folder which might look like:

```
certifi==2020.12.5
click==7.1.2
Flask==1.1.2
Flask-Cors==3.0.10
gunicorn==20.0.4
itsdangerous==1.1.0
Jinja2==2.11.3
joblib==1.0.0
MarkupSafe==1.1.1
numpy==1.20.1
pandas==1.2.1
python-dateutil==2.8.1
pytz==2021.1
scikit-learn==0.24.1
scipy==1.6.0
six==1.15.0
```

```
threadpoolctl==2.1.0
Werkzeug==1.0.1
```

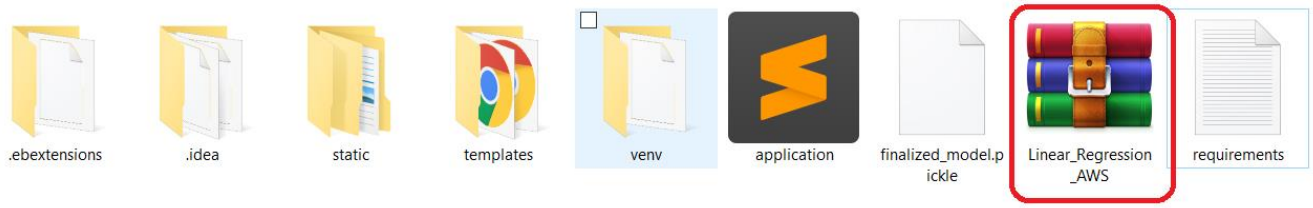
So Now we are ready with all files let review it one by one and move towards out last step.

1) .ebextensions → Python.config this file helps AWS while deploying giving application name .
2) Templates → Base,Index,Result.html this all are HTML files (front end) and should in Templates folder which help Flask App for rendering
3) Application.py → Main Python Script
4) finalized_model.pickle → Machine Learning Model
5) requirements.txt file for aws to verify packages and their versions

Optional

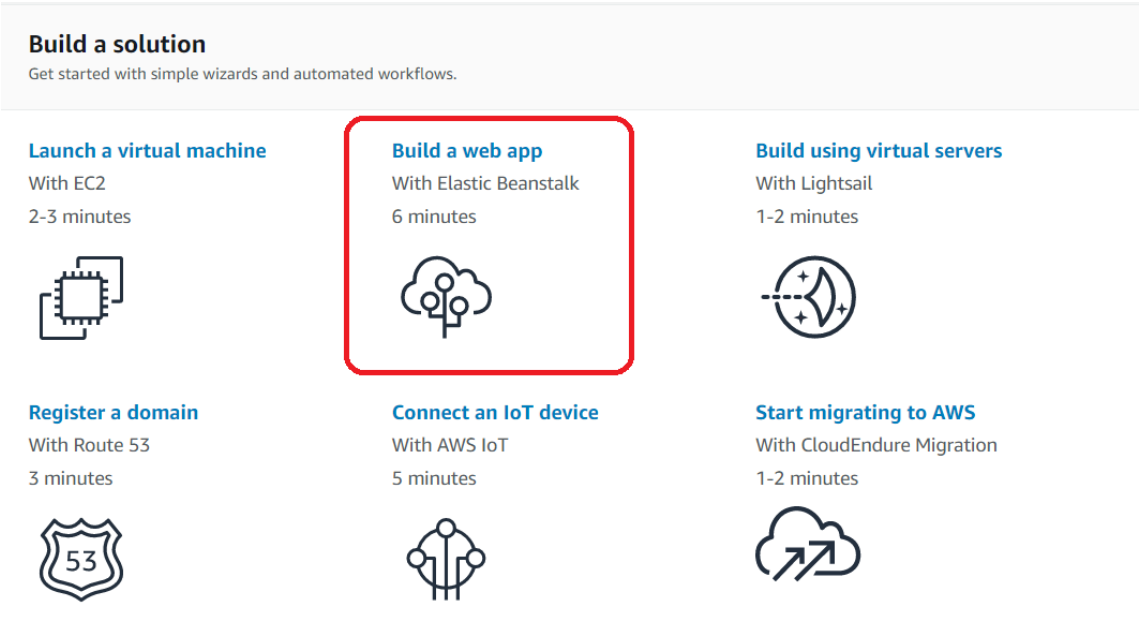
6) If we are using any css or js file it should be placed under static (main folder) → CSS,JS (sub folders).

Now Merge all File in one .zip folder like



Deployment

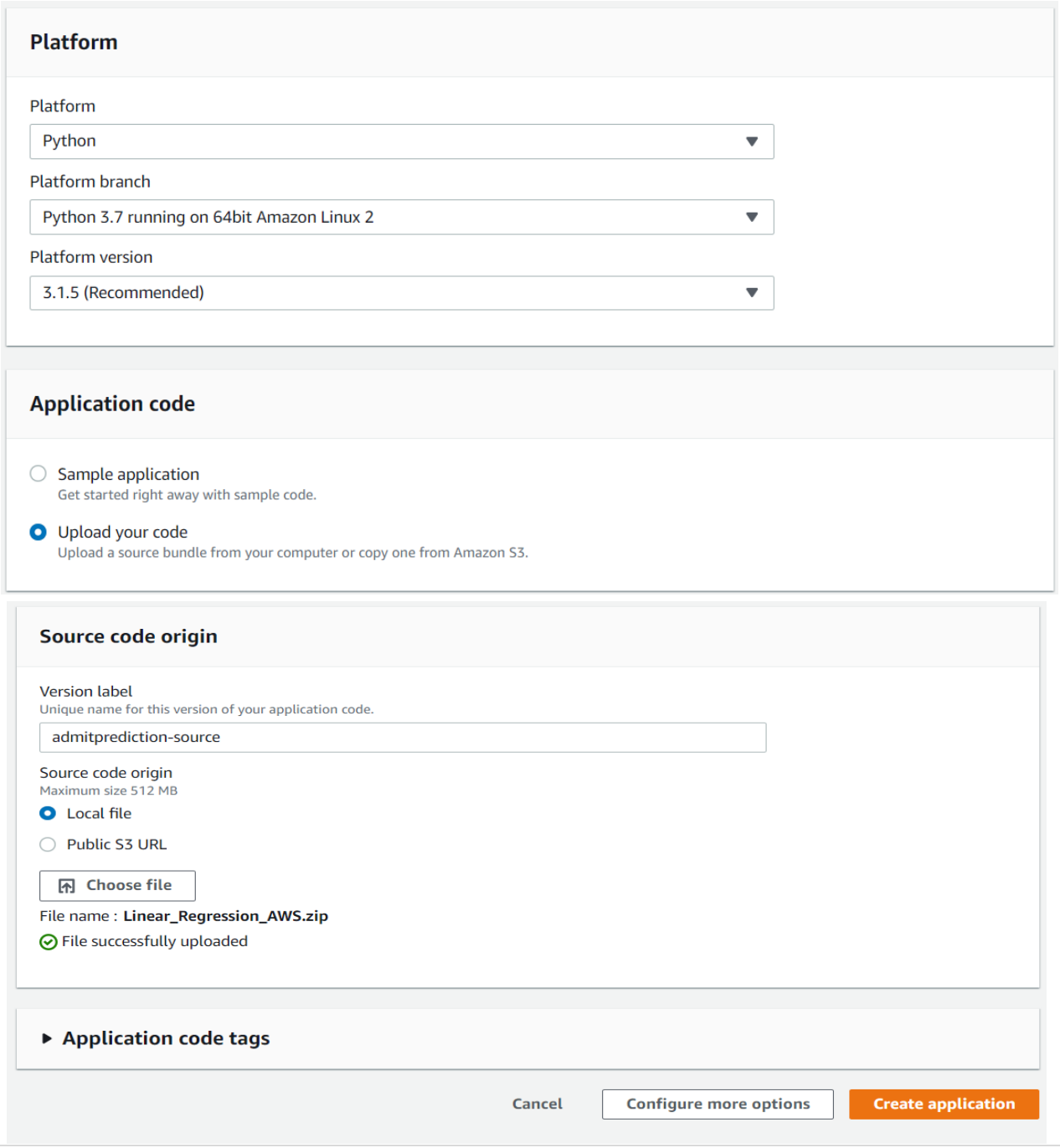
1) Go to https://aws.amazon.com/ and create an account if you don't have.
2) Go to the console and go to the 'Build a web app' section and click it.



3) Give Application Name

4) Select Python in Platform and Upload your Code in Application Code.

5) Upload your Zip file and click on Create application.



Final Result

Predict Your chances for Admission

Subject	Range	Marks
GRE Score	130-340	
TOEFL Score	0-120	
University rating	0-5	
SOP Score	0-5	
LOR Score	0-5	
CGPA	0-10	
Research	Yes	Predict

Prediction
Your chance for admission is **85** percentage