



iNeuron

Deployment of Machine Learning Model to Heroku Cloud

Contents

1. The Problem statement:	3
2. Application Design:	3
3. Pre-requisites:	5
4. Python Implementation:	5
5. Flask App:	10
6. Steps before cloud deployment:	14
7. Deployment to Heroku:	15

iNeuron

Preface

This book is intended to help all the data scientists out there. It is a step by step guide for creating a machine learning model right from scratch and then deploying it to Heroku 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!

iNeuron

Machine Learning with Deployment to Heroku Cloud Platform

1. The Problem statement:

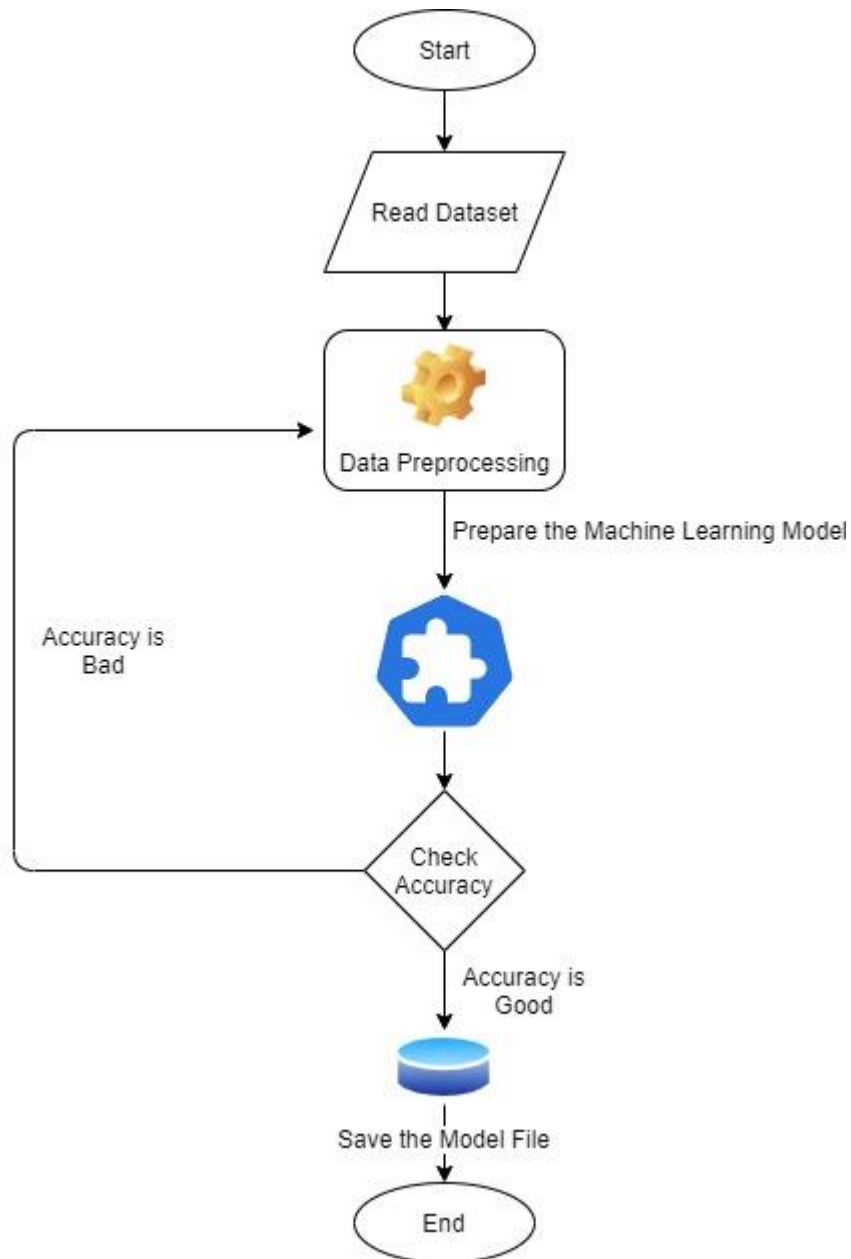
The goal here is to find the chance of admission of a candidate based on his/her GRE Score (out of 340), TOEFL Score (out of 120), rating of the University (out of 5) in which he/she is trying to get admission, Strength of the SOP (out of 5), strength of the Letter Of Recommendation (out of 5), CGPA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

2. Application Design:

Once we have the data source fixed, the machine learning approach majorly consists of two pipelines:

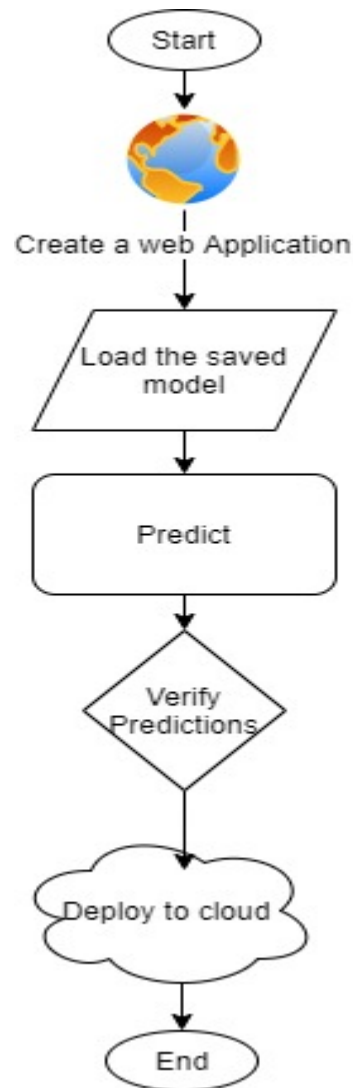
- **The Training Pipeline**

The training pipeline includes data pre-processing, selecting the right algorithm for creating the machine learning model, checking the accuracy of the created model and then saving the model file.



- The Testing Pipeline**

Once the training is completed, we need to expose the trained model as an API for the user to consume it. For prediction, the saved model is loaded first and then the predictions are made using it. If the web app works fine, the same app is deployed to the cloud platform.



3. Pre-requisites:

- Basic knowledge of flask framework.
- Any Python IDE installed(we are using PyCharm).
- A Heroku account.
- Basic understanding of HTML.

4. Python Implementation:

4.1 Importing the necessary Files

We'll first import all the required libraries to proceed with our machine learning model.

```
# necessary Imports
import pandas as pd
import matplotlib.pyplot as plt
import pickle
%matplotlib inline
```

4.2 Reading the Data File

```
df= pd.read_csv('Admission_Prediction.csv') # reading the CSV file
```

4.3 Data Pre-processing and Exploratory Data Analysis

- First, we print a small sample from the data.

```
df.head() # checking the first five rows from the dataset
```

	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

- We check for the datatypes and missing values in the dataset.

```
df.info() # printing the summary of the dataframe
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
Serial No.      500 non-null int64
GRE Score       485 non-null float64
TOEFL Score     490 non-null float64
University Rating 485 non-null float64
SOP             500 non-null float64
LOR             500 non-null float64
CGPA            500 non-null float64
Research        500 non-null int64
Chance of Admit 500 non-null float64
dtypes: float64(7), int64(2)
memory usage: 35.2 KB
```

As shown in the screenshot above, the highlighted columns have some missing values. Those missing values need to be imputed.

- Imputing the missing values in the dataset.

```
df['GRE Score'].fillna(df['GRE Score'].mode()[0],inplace=True)
# to replace the missing values in the 'GRE Score' column with the
# mode of the column
# Mode has been used here to replace the scores with the most
# occurring scores so that data follows the general trend

df['TOEFL Score'].fillna(df['TOEFL Score'].mode()[0],inplace=True)
# to replace the missing values in the 'TOEFL Score' column with the
# mode of the column
# Mode has been used here to replace the scores with the most
```

occurring scores so that data follows the general trend

```
df['University Rating'].fillna(df['University
Rating'].mean(),inplace=True)
# to replace the missing values in the 'University Rating' column
with the mode of the column
# Mean has been used here to replace the scores with the average
score
```

- Now, we create separate training and test data sets.

```
# dropping the 'Chance of Admit' and 'serial number' as they are not
going to be used as features for prediction
x=df.drop(['Chance of Admit','Serial No.'],axis=1)
# 'Chance of Admit' is the target column which shows the probability
of admission for a candidate
y=df['Chance of Admit']
```

The new data set looks like:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	337.0	118.0	4.0	4.5	4.5	9.65	1
1	324.0	107.0	4.0	4.0	4.5	8.87	1
2	312.0	104.0	3.0	3.0	3.5	8.00	1
3	322.0	110.0	3.0	3.5	2.5	8.67	1
4	314.0	103.0	2.0	2.0	3.0	8.21	0

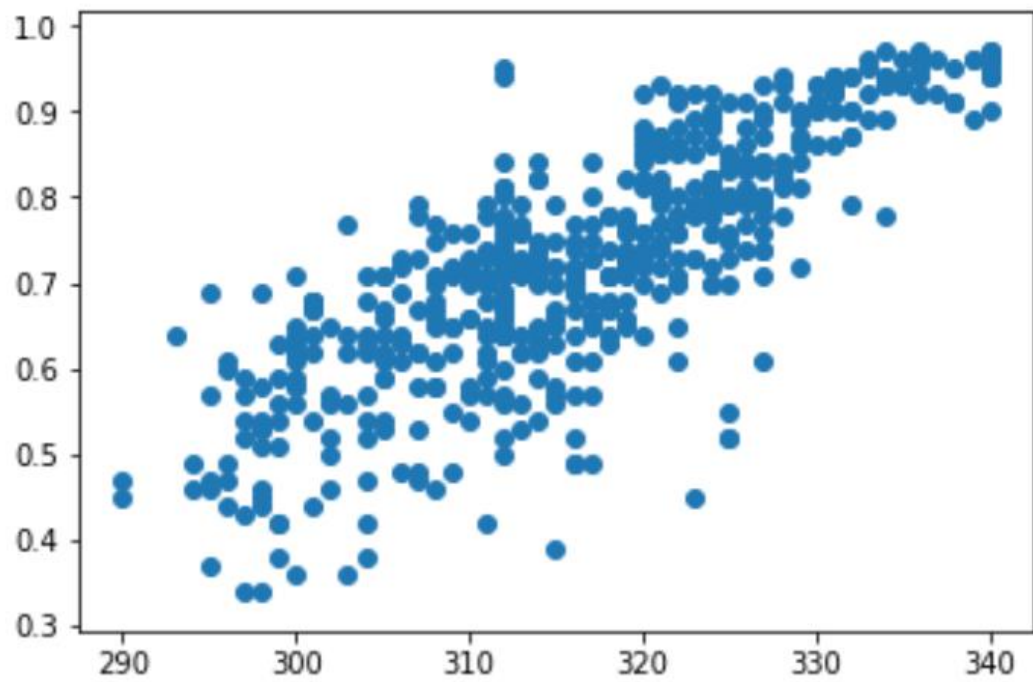
Generally, we'd use a scaler to transform data to the same scale. But as we are just at the beginning of the curriculum, we are skipping that. It'll be discussed in the forthcoming reading materials.

- Once the feature columns have been separated, we'll plot the graphs among the feature columns and the label column to see the relationship between them.

Note: If the same code is being written in a python IDE, instead of a Jupyter Notebook, please use `plt.show()` for the showing the graphs.

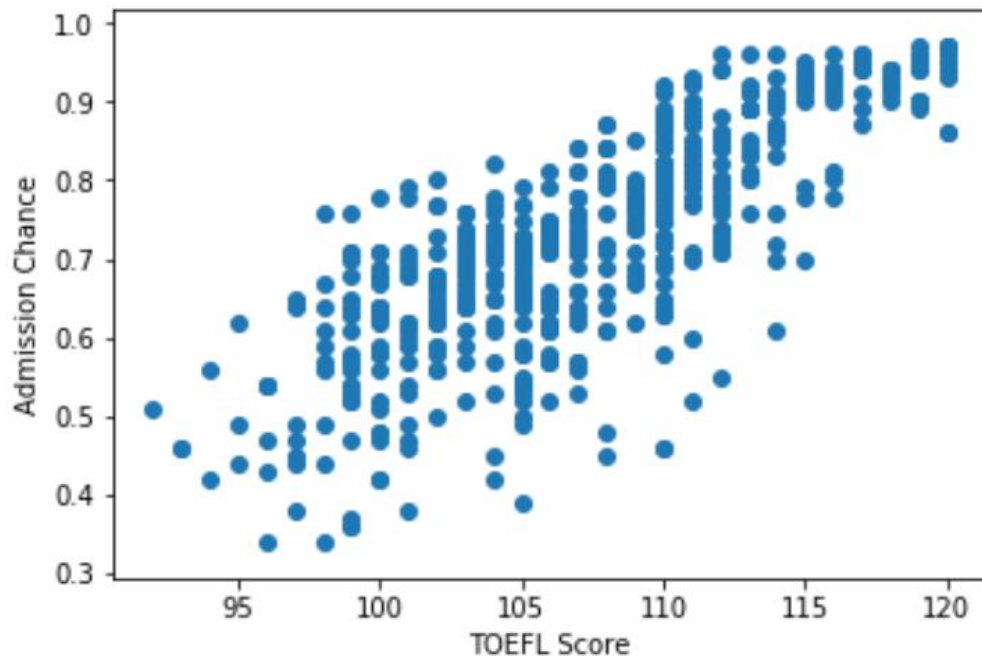
- A graph between GRE Score and Chance of Admission

```
plt.scatter(df['GRE Score'],y) # Relationship between GRE Score and
Chance of Admission
```

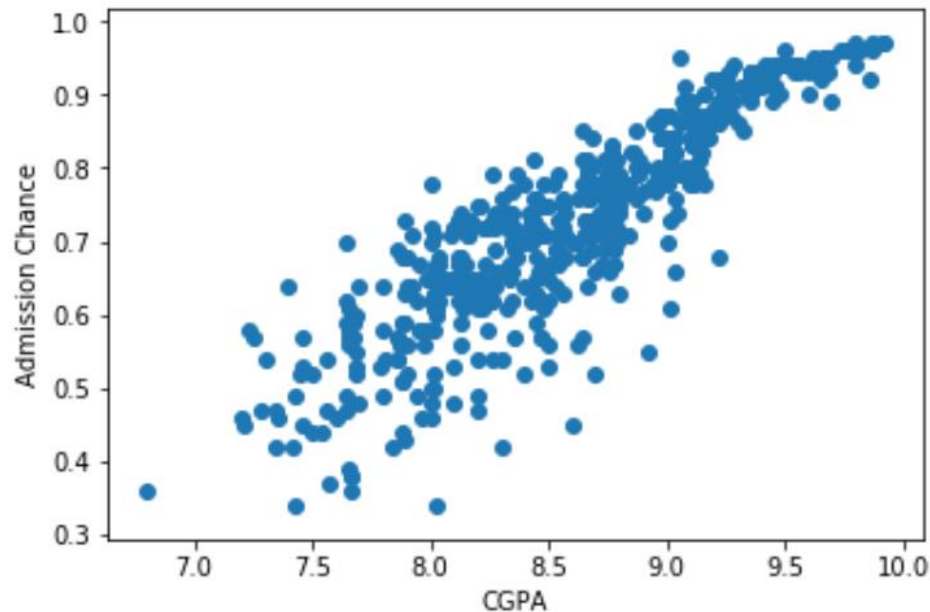
- A graph between TOEFL Score and Chance of Admission

```
plt.scatter(df['TOEFL Score'],y) # Relationship between TOEFL Score and Chance of Admission
```



- A graph between CGPA and Chance of Admission

```
plt.scatter(df['CGPA'],y) # Relationship between CGPA and Chance of Admission
```



- From the above graphs between the continuous feature variables and the label column, it can be concluded that they exhibit a linear relationship amongst them. So, we'll use Linear regression for prediction.
- Once we have determined the Machine Learning algorithm to use, we'll split the datasets into train and test sets as shown below:

```
# splitting the data into training and testing sets
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.33,
                                                    random_state=100)
```

- Now, we'll fit this data to the Linear Regression model.

```
# fitting the data to the Linear regression model
from sklearn import linear_model
reg = linear_model.LinearRegression()
reg.fit(train_x, train_y)
```

- Let's check the accuracy of our model now. Accuracy is calculated by comparing the results to the test data set.

```
# calculating the accuracy of the model
from sklearn.metrics import r2_score
score = r2_score(reg.predict(test_x), test_y)
```

- If we are content with the model accuracy, we can now save the model to a file.

```
# saving the model to the local file system
filename = 'finalized_model.pickle'
pickle.dump(reg, open(filename, 'wb'))
```

- Let's predict using our model.

```
# prediction using the saved model.
loaded_model = pickle.load(open(filename, 'rb'))
prediction=loaded_model.predict([[320,120,5,5,5,10,1]])
print(prediction[0])
```

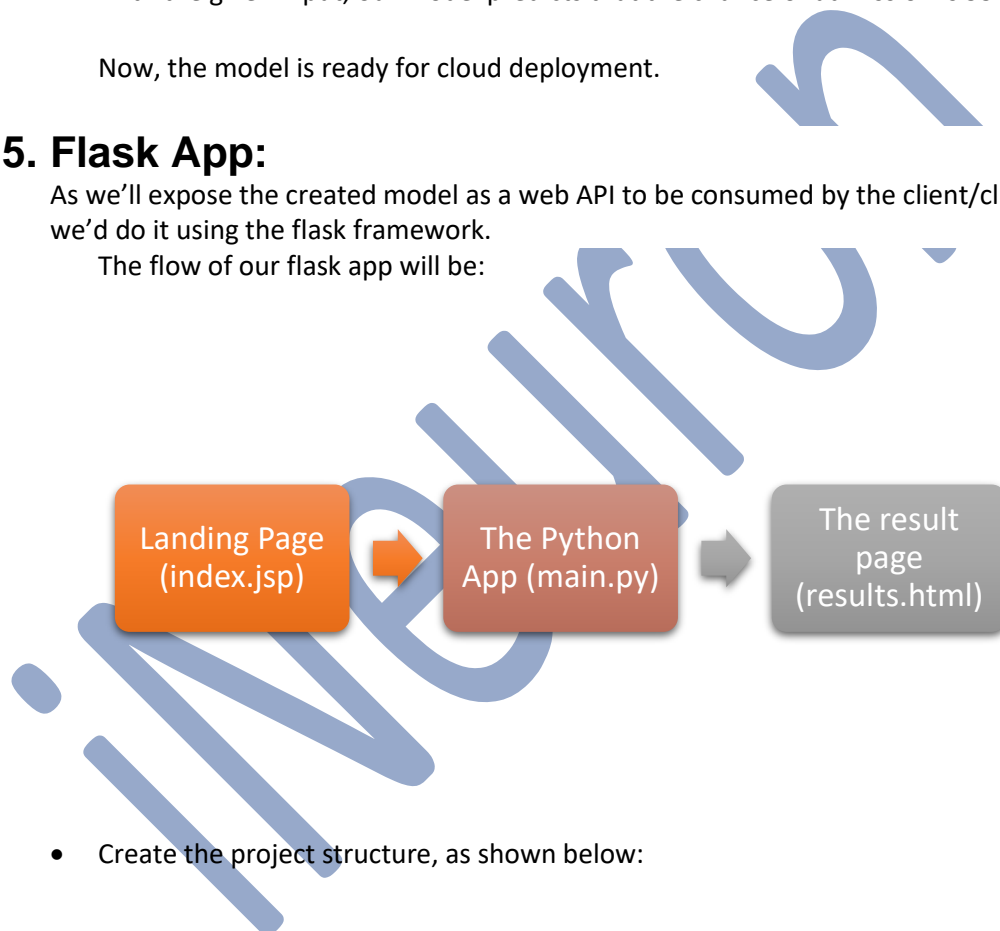
With the given input, our model predicts that the chance of admission is 99.57 per cent.

Now, the model is ready for cloud deployment.

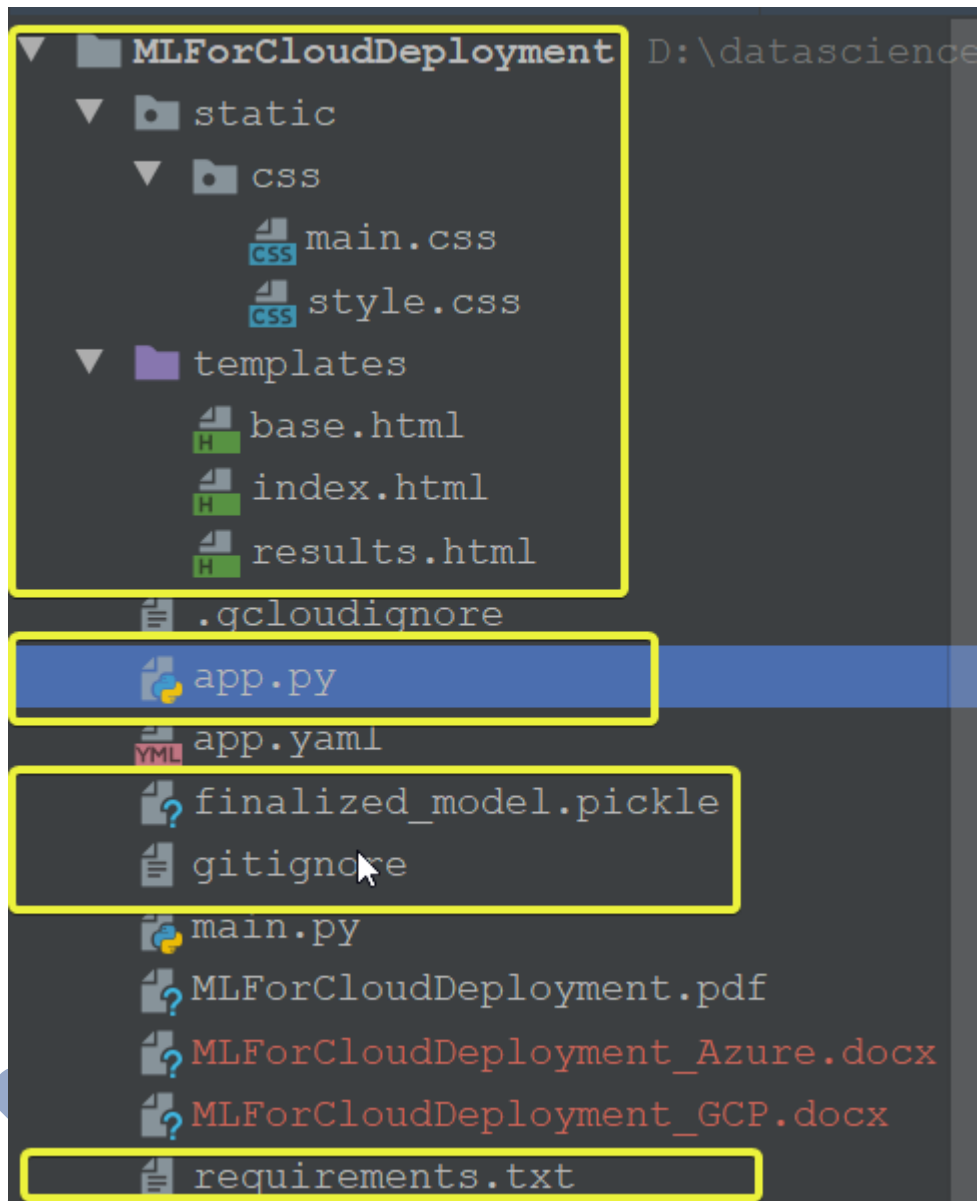
5. Flask App:

As we'll expose the created model as a web API to be consumed by the client/client APIs, we'd do it using the flask framework.

The flow of our flask app will be:



- Create the project structure, as shown below:



Only create the files and folders (marked in yellow), and put the saved model file in the same folder as your app.py file.

- Index.html:

```
{% extends 'base.html' %}

{% block head %}

<title>Search Page</title>
<link rel="stylesheet" href="{% url_for('static',
filename='css/style.css') %}">
{% endblock %}

{% block body %}
<div class="content">
  <h1 style="text-align: center">Predict Your chances for
  Admission</h1>
</div>
{% endblock %}
```

```
<div class="form">
    <form action="/predict" method="POST">
        <input type="number" name="gre_score" id="gre_score"
placeholder="GRE Score">
        <input type="number" name="toefl_score" id="toefl_score"
placeholder="TOEFL Score">
        <input type="number" name="university_rating"
id="university_rating" placeholder="University Rating">
        <input type="number" name="sop" id="sop"
placeholder="SOP Score">
        <input type="number" name="lor" id="lor"
placeholder="LOR Score">
        <input type="number" name="cgpa"
id="cgpa"placeholder="CGPA" step="any">
        <select name="research" id="research">
            <option value="yes">Yes</option>
            <option value="no">No</option>
        </select>
        <input type="submit" value="Predict">
    </form>
</div>
</div>
{% endblock %}
```

- app.py:

```
# importing the necessary dependencies
from flask import Flask, render_template, request, jsonify
from flask_cors import CORS, cross_origin
import pickle

app = Flask(__name__) # initializing a flask app

@app.route('/', methods=['GET']) # route to display the home page
@cross_origin()
def homePage():
    return render_template("index.html")

@app.route('/predict', methods=['POST', 'GET']) # route to show the
predictions in a web UI
@cross_origin()
def index():
    if request.method == 'POST':
        try:
            # reading the inputs given by the user
            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'
    loaded_model = pickle.load(open(filename, 'rb')) # loading
the model file from the storage
    # predictions using the loaded model file

prediction=loaded_model.predict([[gre_score,toefl_score,university_rati
ng,sop,lor,cgpa,research]])
    print('prediction is', prediction)
    # showing the prediction results in a UI
    return
render_template('results.html',prediction=round(100*prediction[0]))
    except Exception as e:
        print('The Exception message is: ',e)
        return 'something is wrong.'
    # return render_template('results.html')
    else:
        return render_template('index.html')

if __name__ == "__main__":
    #app.run(host='127.0.0.1', port=8001, debug=True)
    app.run(debug=True) # running the app

```

- results.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/normali
ze.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">
      <div class="header">Prediction</div>

      <p>Your chance for admission is {{prediction}} percent</p>
    </div>

```

```
</body>
</html>
```

6. Steps before cloud deployment:

We need to change our code a bit so that it works unhindered on the cloud, as well.

- a) Add a file called 'gitignore' inside the project folder. This folder contains the list of the files which we don't want to include in the git repository. My gitignore file looks like:

```
.idea
As I am using PyCharm as an IDE, and it's provided by the IntelliJ Idea
community, it automatically adds the .idea folder containing some metadata.
We need not include them in our cloud app.
```

- b) Add a file called 'Procfile' inside the 'reviewScrapper' folder. This folder contains the command to run the flask application once deployed on the server:

```
web: gunicorn app:app
Here, the keyword 'web' specifies that the application is a web application.
And the part 'app:app' instructs the program to look for a flask application
called 'app' inside the 'app.py' file. Gunicorn is a Web Server Gateway
Interface (WSGI) HTTP server for Python.
```

- c) Open a command prompt window and navigate to your 'reviewScrapper' folder. Enter the command 'pip freeze > requirements.txt'. This command generates the 'requirements.txt' file. My requirements.txt looks like:

```
beautifulsoup4==4.8.1
bs4==0.0.1
certifi==2019.9.11
Click==7.0
Flask==1.1.1
Flask-Cors==3.0.8
gunicorn==20.0.4
itsdangerous==1.1.0
Jinja2==2.10.3
MarkupSafe==1.1.1
numpy==1.17.4
opencv-python==4.1.2.30
Pillow==6.2.1
pymongo==3.9.0
requests==2.21.0
requests-oauthlib==1.2.0
six==1.13.0
soupsieve==1.9.5
Werkzeug==0.16.0
```

requirements.txt helps the Heroku cloud app to install all the dependencies before starting the webserver.

7. Deployment to Heroku:

- After installing the Heroku CLI, Open a command prompt window and navigate to your project folder.

- Type the command 'heroku login' to login to your heroku account as shown below:

```
D:\datascience\iNeuron\docs\MLForCloudDeployment>heroku login
heroku: Press any key to open up the browser to login or q to exit:
```

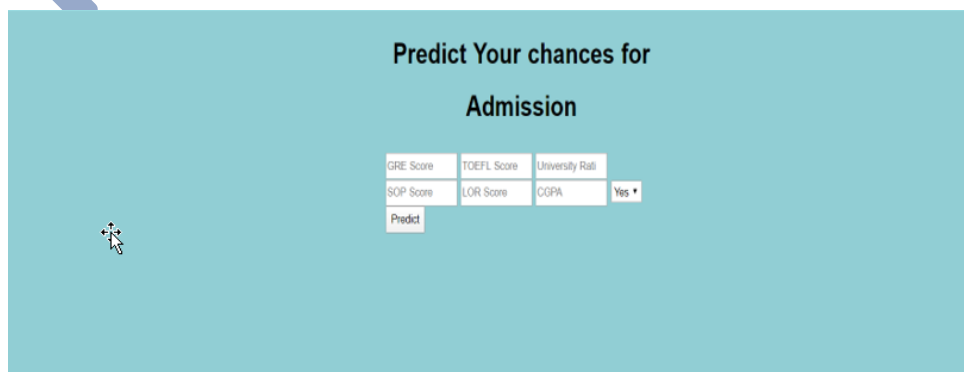
- After logging in to Heroku, enter the command 'heroku create' to create a heroku app. It will give you the URL of your Heroku app after successful creation.
- Before deploying the code to the Heroku cloud, we need to commit the changes to the local git repository.

- Type the command 'git init' to initialize a local git repository as shown below:

```
D:\datascience\iNeuron\docs\MLForCloudDeployment>git init
```

- Enter the command 'git status' to see the uncommitted changes
- Enter the command 'git add .' to add the uncommitted changes to the local repository.
- Enter the command 'git commit -am "make it better"' to commit the changes to the local repository.
- Enter the command 'git push heroku master' to push the code to the heroku cloud.
- After deployment, heroku gives you the URL to hit the web API.
- Once your application is deployed successfully, enter the command 'heroku logs --tail' to see the logs.

Final Result



Predict Your chances for Admission

GRE Score	TOEFL Score	University Rating
SOP Score	LOR Score	CGPA
		Yes
Predict		

Thank You!