

**Author: Oluwaseun Tope**

Title: Deployment of Machine Learning Model to Amazon Web Services.

Author: Oluwaseun Tope

Date: 07/09/2018

# Deployment of Machine Learning Model to Amazon Web Services

## Contents

<b>1. The Problem statement:</b>	3
<b>2. Application Design:</b>	3
<b>3. Pre-requisites:</b>	5
<b>4. Python Implementation:</b>	5
<b>5. Flask App:</b>	10
<b>6. Deployment to AWS:</b>	14

## Preface

This Presentation is intended to display a step by step guide for creating a machine learning model right from scratch and then deploying it to 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.

**Author: Oluwaseun Tope**

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.

# Machine Learning with Deployment to AWS Cloud Platform

## 1. The Problem statement:

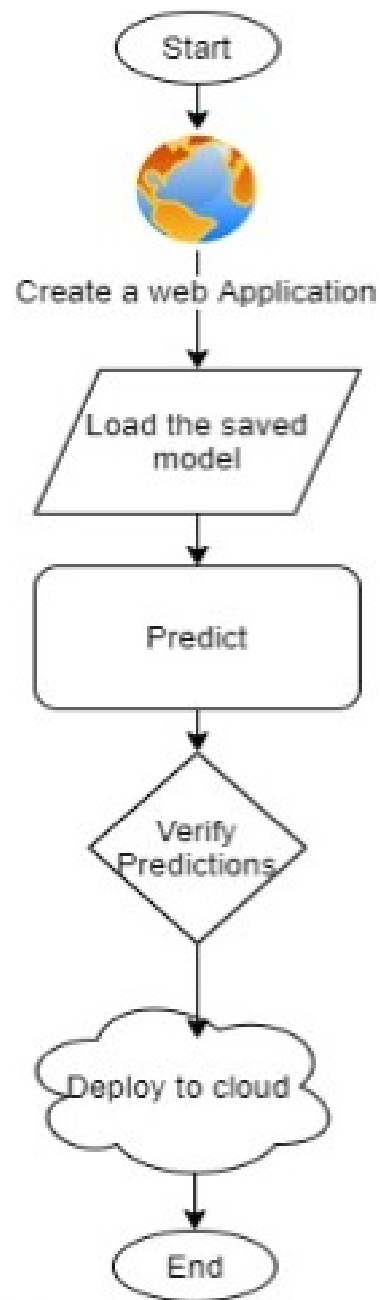
The goal here is to find the chance of admission of a candidate based on his/her GRE Score (out of 340), TOEFL Score (out of 120), rating of the University (out of 5) in which he/she is trying to get admission, Strength of the SOP (out of 5), strength of the Letter Of Recommendation (out of 5), CGPA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



### 3. Pre-requisites:

- Basic knowledge of flask framework.
- Any Python IDE installed(we are using PyCharm).
- An AWS 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 '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['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
```

Activate Windows  
Go to Settings to activate Windows.

**Author: Oluwaseun Tope**

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

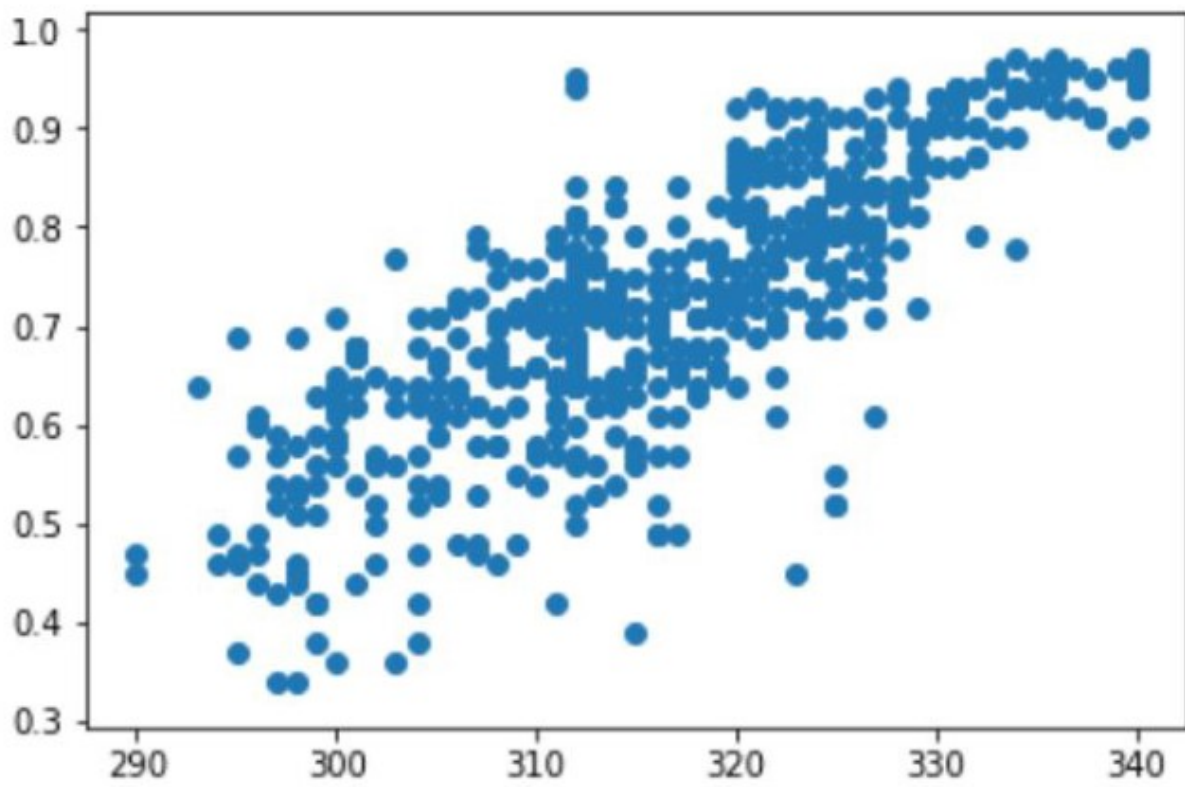
Generally, we'd use a scaler to transform data to the same scale. The scaling process is skipped. Once the features columns have been separated, the next is to plot the graphs among the features columns and the label column to see the relationship between them.

➤ A graph between GRE Score and Chance of Admission

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



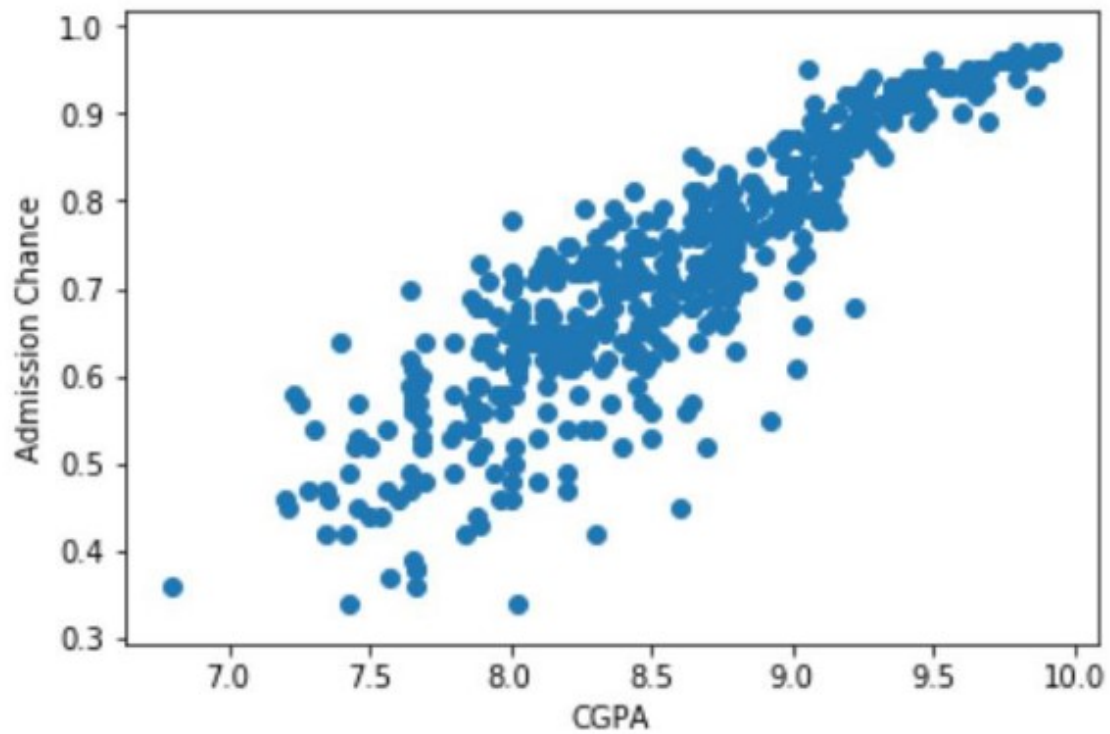
*Author: Oluwaseun Tope*



**Author: Oluwaseun Tope**

- A graph between CGPA and Chance of Admission

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

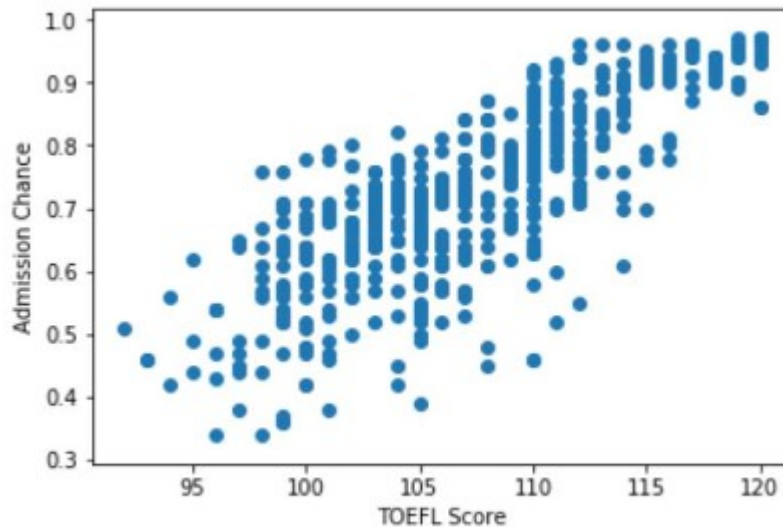




**Author: Oluwaseun Tope**

- A graph between TOEFL Score and Chance of Admission

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



- From the above graphs between the continuous feature variables and the label columns, it can be concluded that they exhibit a linear relationship amongst them. So, we'll use Linear regression for prediction.
- Once the Machine Learning Algorithm to use is determined, The next step is to 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)
```

- The next step is to 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)
```

**Author: Oluwaseun Tope**

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.

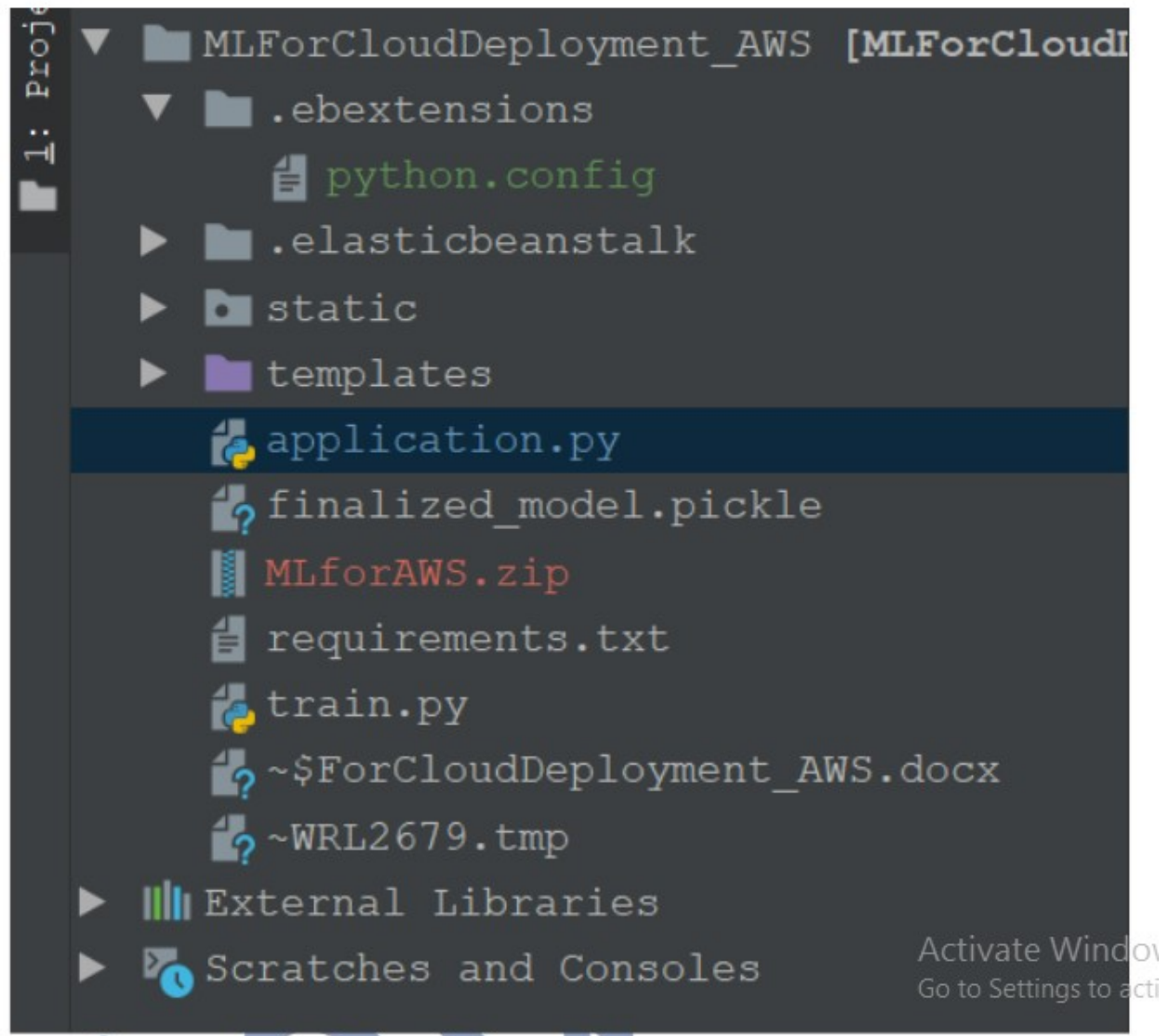
## 5. Flask App:

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 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 %}
```

- application.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
```

**Author: Oluwaseun Tope**

```
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>
```

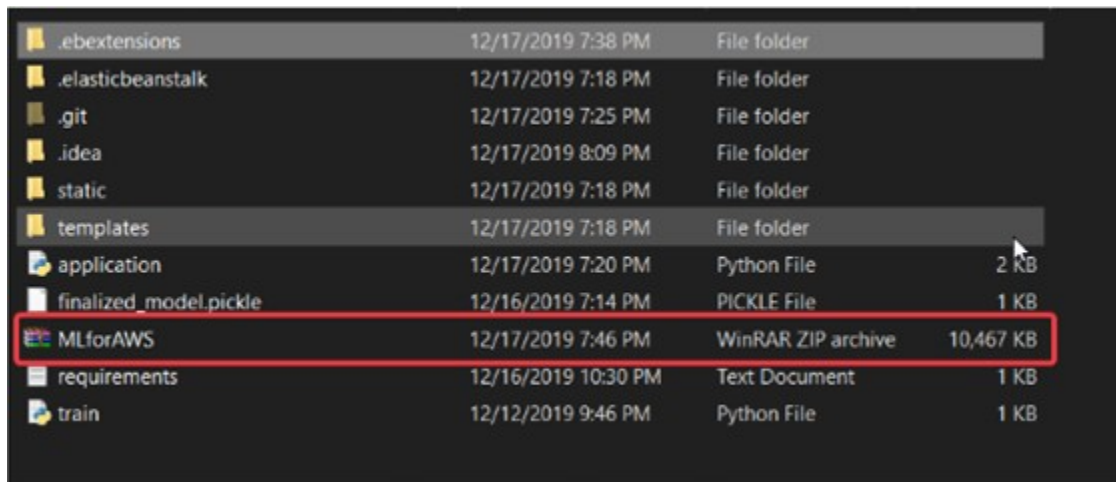
- Python.config

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

## 6. Deployment to AWS:

### 6.1 Points to consider before deployment

- The python application file should be named application.py
- Create a requirements.txt using pip freeze > requirements.txt from the project folder
- Create a folder '.ebextensions' and create a file 'python.config' inside it. Make sure to populate the content of python.config, as shown above.
- Create the zip file from the project folder itself.



.ebextensions	12/17/2019 7:38 PM	File folder	
.elasticbeanstalk	12/17/2019 7:18 PM	File folder	
.git	12/17/2019 7:25 PM	File folder	
.idea	12/17/2019 8:09 PM	File folder	
static	12/17/2019 7:18 PM	File folder	
templates	12/17/2019 7:18 PM	File folder	
application	12/17/2019 7:20 PM	Python File	2 KB
finalized_model.pickle	12/16/2019 7:14 PM	PICKLE File	1 KB
MLforAWS	12/17/2019 7:46 PM	WinRAR ZIP archive	10,467 KB
requirements	12/16/2019 10:30 PM	Text Document	1 KB
train	12/12/2019 9:46 PM	Python File	1 KB

### 6.2 Deployment Process

- An AWS account
- Go to the console and go to the 'Build a web app' section and click it.

**Author: Oluwaseun Tope**

## Build a solution

Get started with simple wizards and automated workflows.

### Launch a virtual machine

With EC2

2–3 minutes



### Build a web app

With Elastic Beanstalk

6 minutes



### Build using virtual servers

With Lightsail

1–2 minutes



### Register a domain

With Route 53

3 minutes



### Connect an IoT device

With AWS IoT

5 minutes



### Start migrating to AWS

With CloudEndure Migration

1–2 minutes



► See more

- Give the name of the application, give platform as python, and select the option to upload your code.

## Create a web app

Create a new application and environment with a sample application or your own code. By creating an environment, you allow AWS Elastic Beanstalk to manage AWS resources and permissions on your behalf. [Learn more](#)

### Application information

Application name

LRTtoAWS

Up to 100 Unicode characters, not including forward slash (/).

☐ Application tags

### Base configuration

Platform

Python

Choose [Configure more options](#) for more platform configuration options.

Application code

☐ Sample application

Get started right away with sample code.

☒ Upload your code

Upload a source bundle from your computer or copy one from Amazon S3.

ZIP or WAR

☐ Application code tags

## Final Result:

**Author: Oluwaseun Tope**

### Predict Your chances for Admission

GRE Score	TOEFL Score	University Rating	
SOP Score	LOR Score	CGPA	Yes ▾
<input type="button" value="Predict"/>			