

Deployment of Machine Learning Model to Google Cloud Platform

Contents

1. The Problem statement:	3
2. Application Design:	3
3. Pre-requisites:	5
4. Python Implementation:	5
5. Flask App:	10
6. Deployment to G-cloud:	14

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 the Google Cloud Platform. 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 Google Cloud Platform

1. The Problem statement:

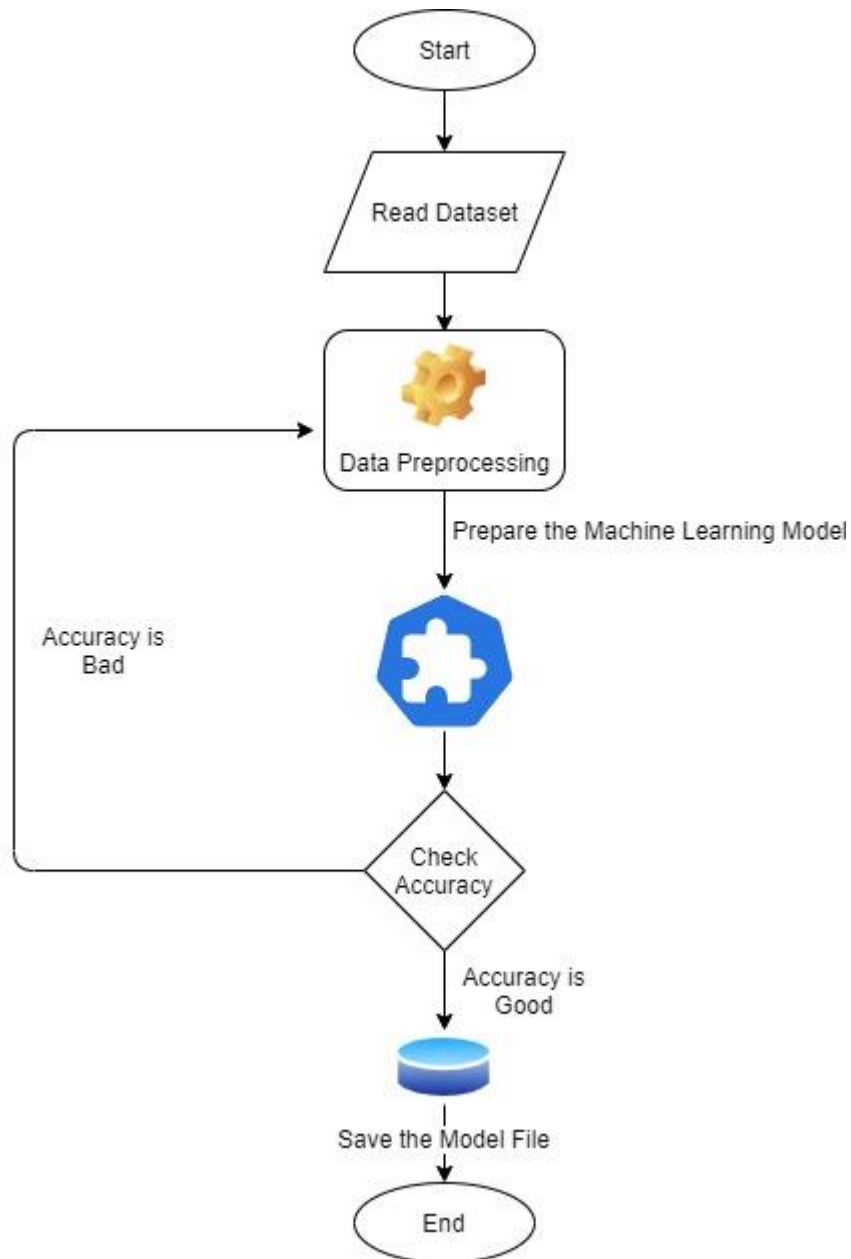
The goal here is to find the chance of admission of a candidate based on his/her GRE Score (out of 340), TOEFL Score (out of 120), rating of the University (out of 5) in which he/she is trying to get admission, Strength of the SOP (out of 5), strength of the Letter Of Recommendation (out of 5), CGPA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

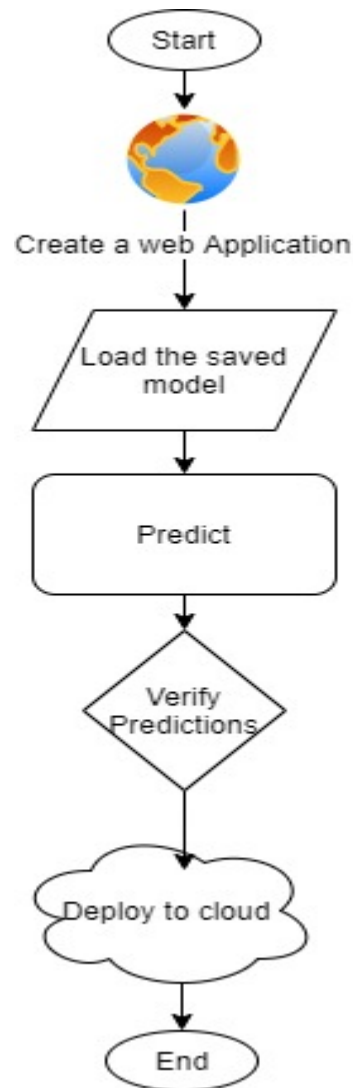
- **The Training Pipeline**

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 done 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 Google Cloud Platform 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

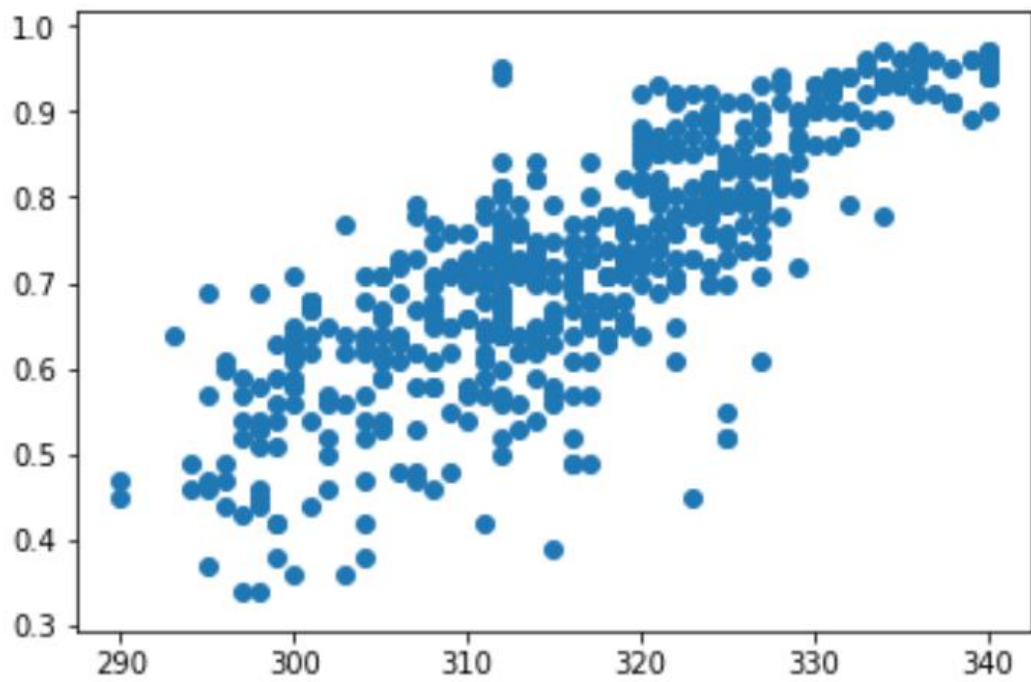
Generally, we'd use a scaler to transform data to the same scale. But as we are just at the beginning of the curriculum, we are skipping that. It'll be discussed in the forthcoming reading materials.

- 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.

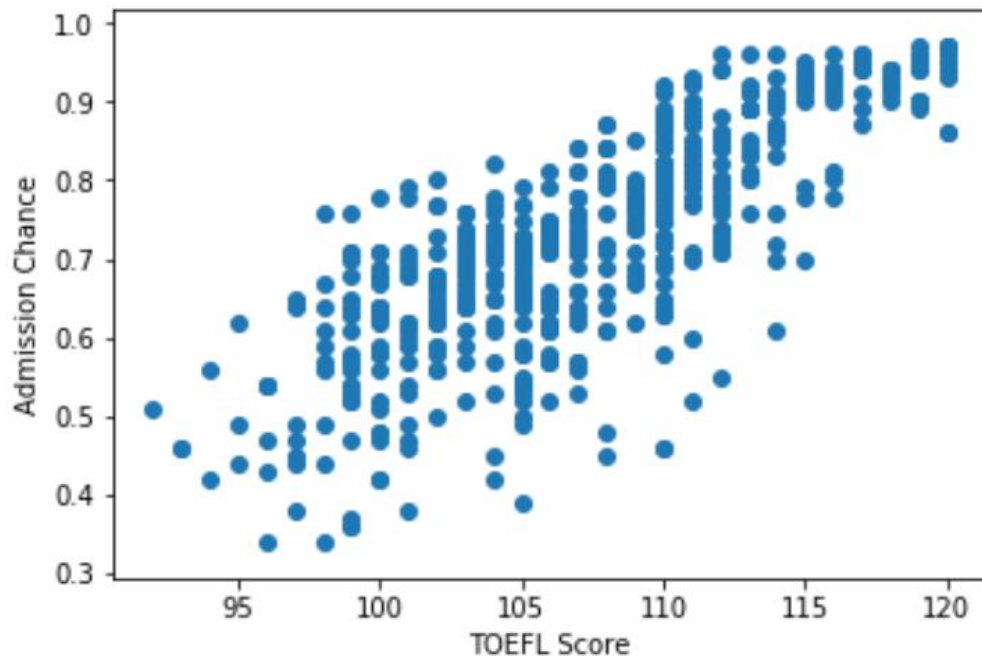
- A graph between GRE Score and Chance of Admission

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

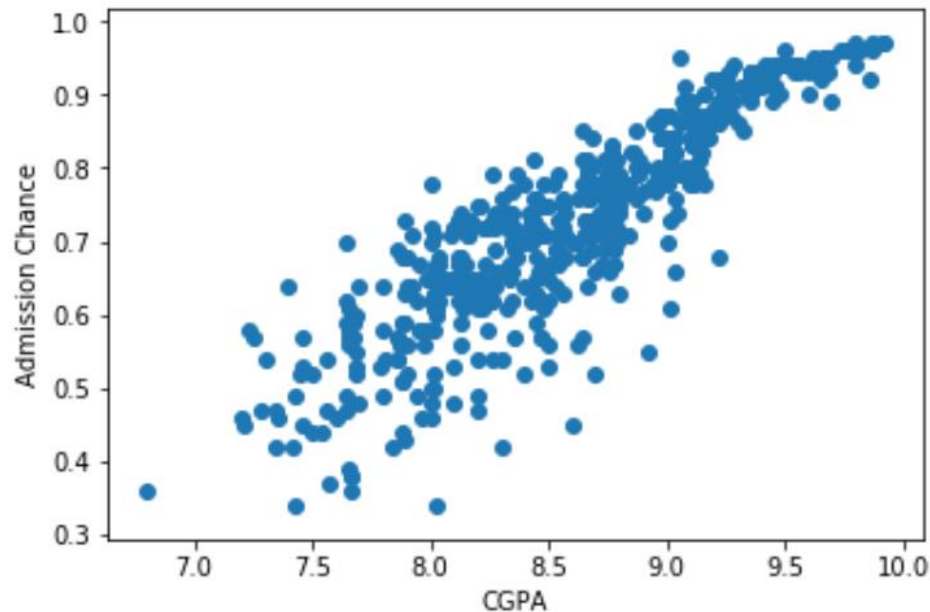
- A graph between TOEFL Score and Chance of Admission

```
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])
```

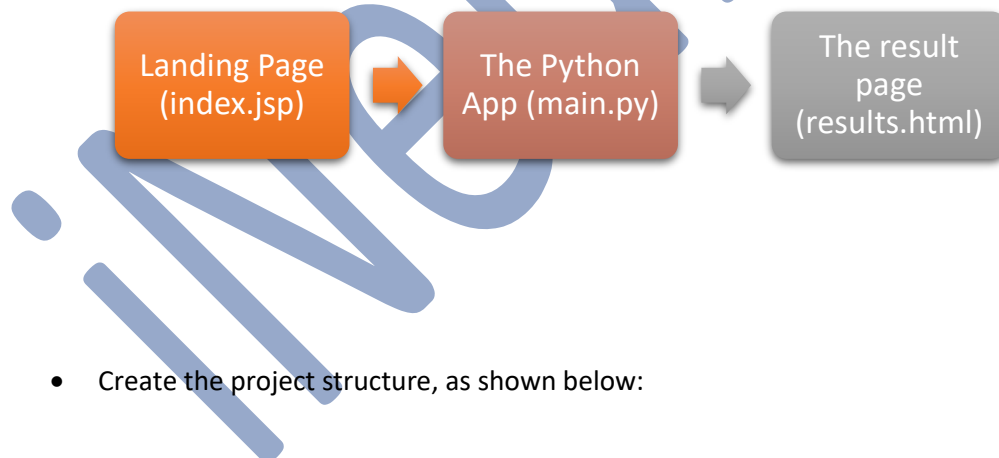
With the given input, our model predicts that the chance of admission is 99.57 per cent.

Now, the model is ready for cloud deployment.

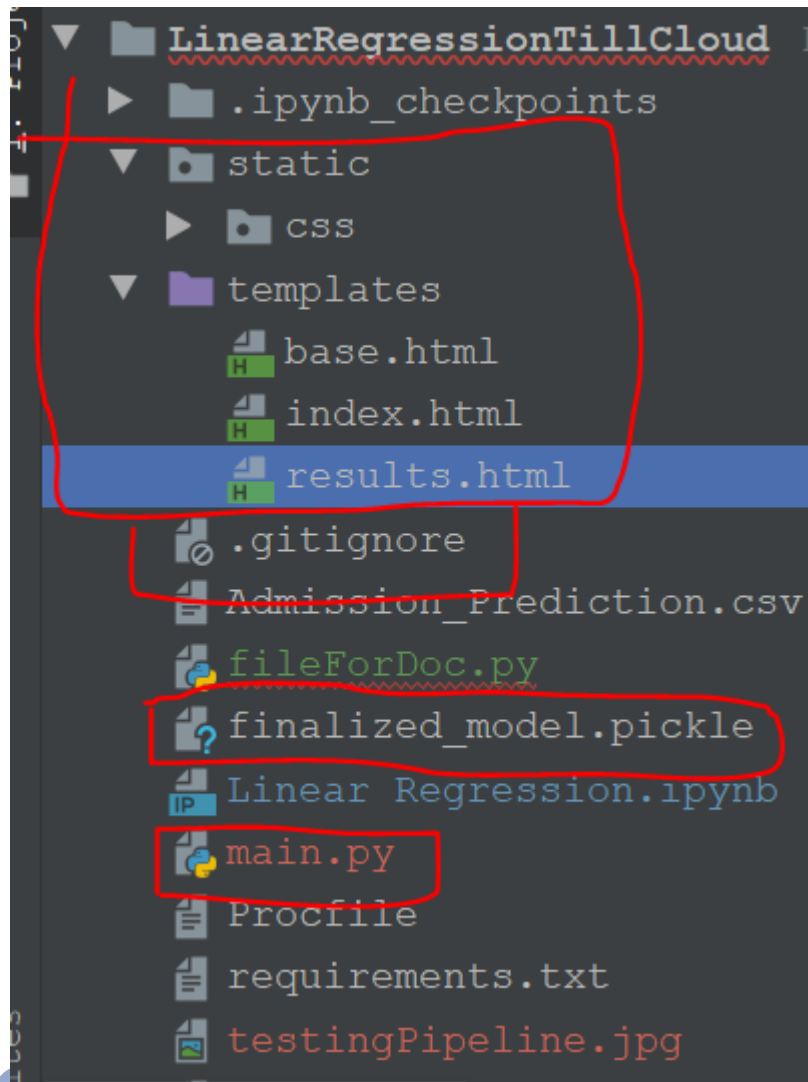
5. Flask App:

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 marked files and folders and put the saved model file in the same folder as your main.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">
 </form>
 </div>
</div>
{% endblock %}
```

```

 <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 %}

```

- main.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>

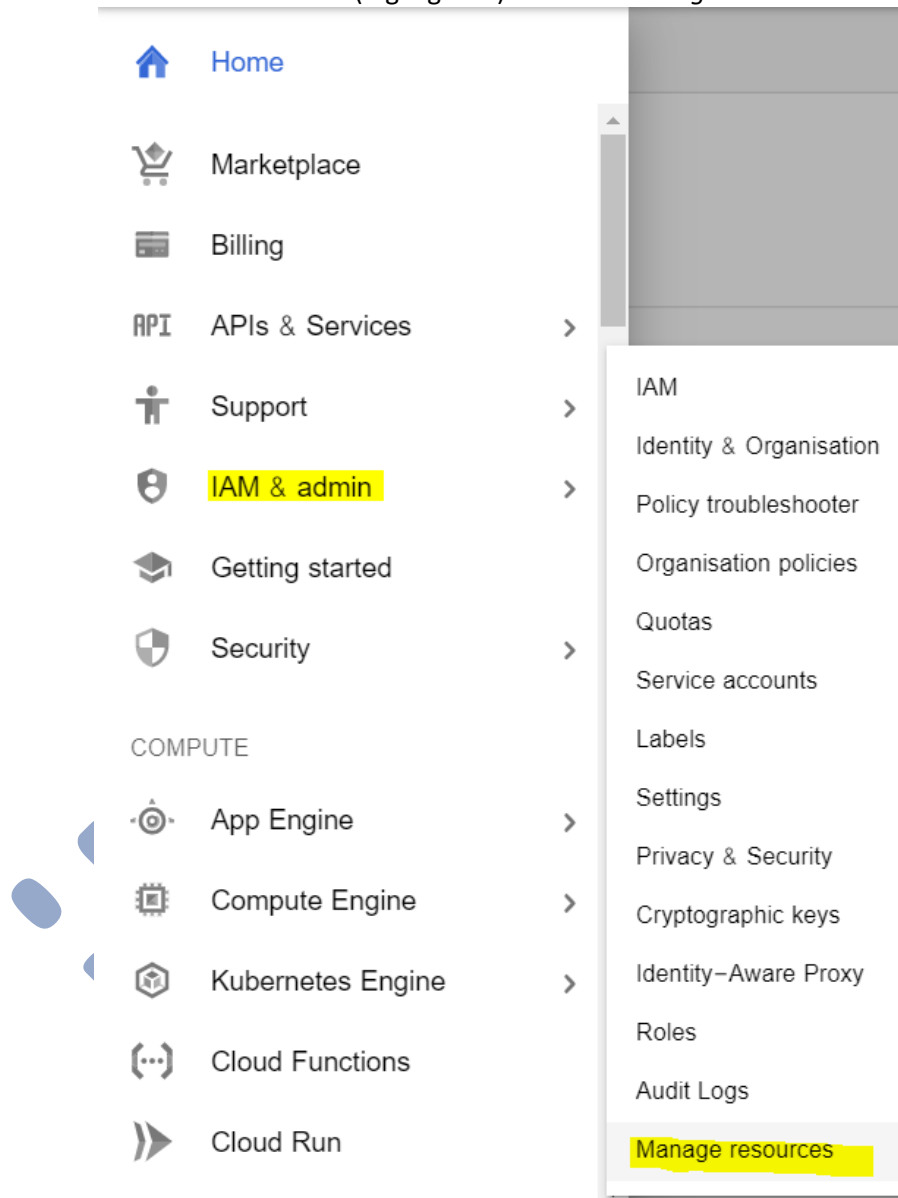
 </body>

```

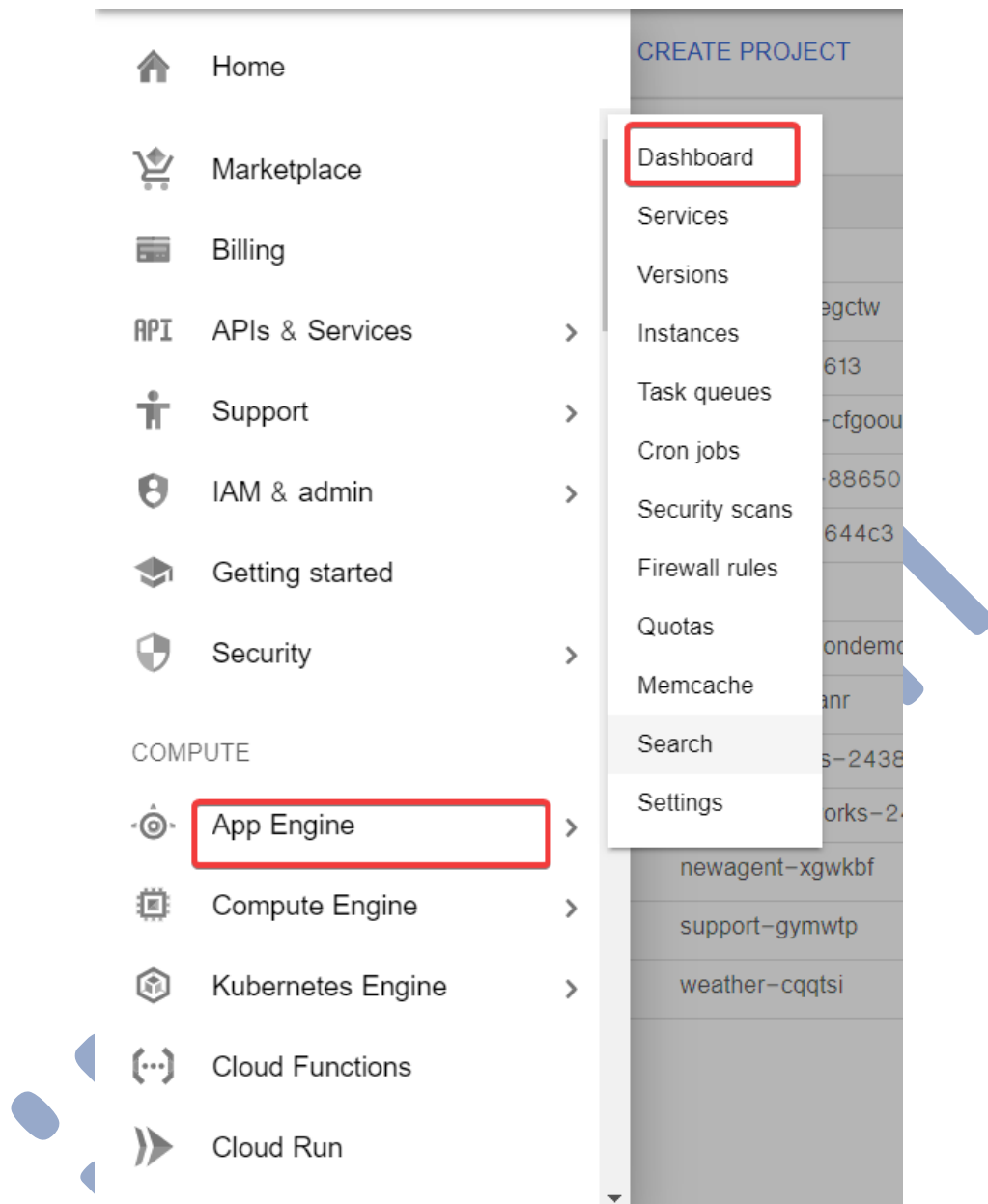
&lt;/html&gt;

## 6. Deployment to G-cloud:

- Go to <https://cloud.google.com/> and create an account if already haven't created one. Then go to the console of your account.
- Go to *IAM and admin* (highlighted) and click *manage resources*.



- Click *CREATE PROJECT* to create a new project for deployment.
- Once the project gets created, select *App Engine* and select *Dashboard*.



- Go to <https://dl.google.com/dl/cloudsdk/channels/rapid/GoogleCloudSDKInstaller.exe> to download the google cloud SDK to your machine.
- Click *Start Tutorial* on the screen and select Python app and click start.










## Hello World

New to App Engine? Start with a simple 'Hello World' app to learn the essentials.

Time: 10 minutes

[START TUTORIAL](#)

 Node.js
  PHP
  Python
  Go

 Java™
  Ruby
  .NET

### Deploy via command line

With the Google Cloud SDK, you can use the CLI to easily create and deploy your app:

```
$ gcloud app deploy
```

[DOWNLOAD THE SDK](#)

## App Engine quickstart

### Introduction

This tutorial shows you how to deploy a sample [Python](#) application to App Engine using the `gcloud` command.

Here are the steps that you will be taking:

- **Create a project**  
Projects bundle code, VMs and other resources together for easier development and monitoring.
- **Build and run your 'Hello World!' app**  
You will learn how to run your app using Cloud Shell, directly in your browser. At the end, you'll deploy your app to the web using the `gcloud` command.
- **After the tutorial...**  
Your app will be real and you'll be able to experiment with it after you deploy, or you can remove it and start afresh.

'Python' and the Python logos are trademarks or registered trademarks of the Python Software Foundation.

[Start](#)


- Check whether the correct project name is displayed and then click next.

## Project setup

GCP organises resources into projects, which collect all of the related resources for a single application in one place.

Begin by creating a new project or selecting an existing project for this tutorial.

Select a project, or [create a new one](#)

 **LinearRegressionDemo**

For details, see [Creating a project](#).

Previous      Step 1 of 8      [Next](#)

- Create a file 'app.yaml' and put 'runtime: python37' in that file.

- Create a 'requirements.txt' file by opening the command prompt/anaconda prompt, navigate to the project folder and enter the command 'pip freeze > requirements.txt'. It is recommended to use separate environments for different projects.
- Your python application file should be called 'main.py'. It is a GCP specific requirement.
- Open command prompt window, navigate to the project folder and enter the command *gcloud init* to initialise the gcloud context.
- It asks you to select from the list of available projects.

```
You are logged in as: [viratsagar26@gmail.com].

Pick cloud project to use:
[1] clgdfdemo-segctw
[2] cloudml-246613
[3] collegedemo-cfgoou
[4] exemplary-works-246613
[5] faq-goilum
[6] fir-functions-88650
[7] health-care-644c3
[8] linearregressiondemo
[9] lowesbot-kr'jam
[10] newagent-xgwkbk
[11] support-gymwtp
[12] teak-environs-243805
[13] weather-cqqtqi
[14] Create a new project

Please enter numeric choice or text value (must exactly match list item):
```

- Once the project name is selected, enter the command *gcloud app deploy app.yaml --project <project name>*.
- After executing the above command, GCP will ask you to enter the region for your application. Choose the appropriate one.

```
l:\datascience\iNeuron\LinearRegressionTillCloud>gcloud app deploy app.yaml --project linearregressiondemo
You are creating an app for project [linearregressiondemo].
WARNING: Creating an App Engine application for a project is irreversible and the region
cannot be changed. More information about regions is at
<https://cloud.google.com/appengine/docs/locations>.

Please choose the region where you want your App Engine application
located:

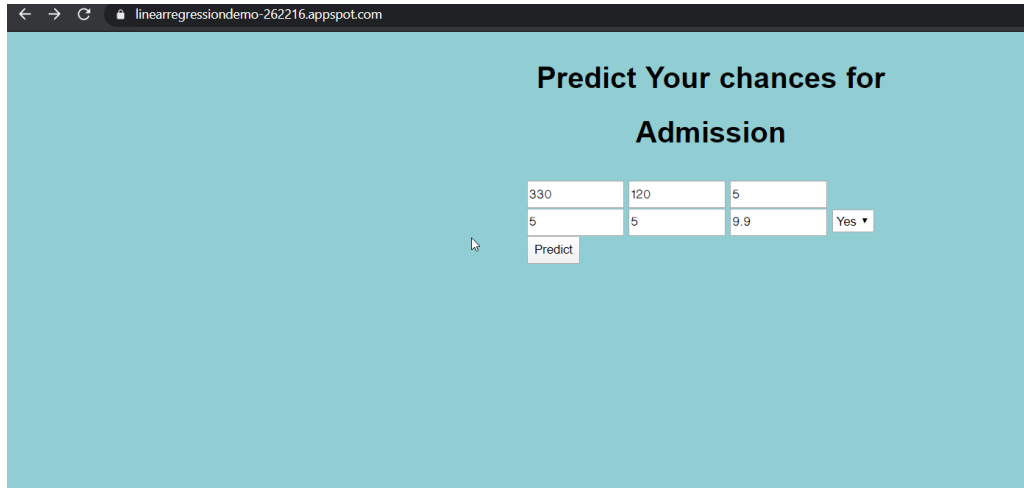
[1] asia-east2 (supports standard and flexible)
[2] asia-northeast1 (supports standard and flexible)
[3] asia-northeast2 (supports standard and flexible)
[4] asia-south1 (supports standard and flexible)
[5] australia-southeast1 (supports standard and flexible)
[6] europe-west (supports standard and flexible)
[7] europe-west2 (supports standard and flexible)
[8] europe-west3 (supports standard and flexible)
[9] europe-west6 (supports standard and flexible)
[10] northamerica-northeast1 (supports standard and flexible)
[11] southamerica-east1 (supports standard and flexible)
[12] us-central (supports standard and flexible)
[13] us-east1 (supports standard and flexible)
[14] us-east4 (supports standard and flexible)
[15] us-west2 (supports standard and flexible)
[16] cancel

Please enter your numeric choice: 4

Creating App Engine application in project [linearregressiondemo] and region [asia-south1]....done.
```

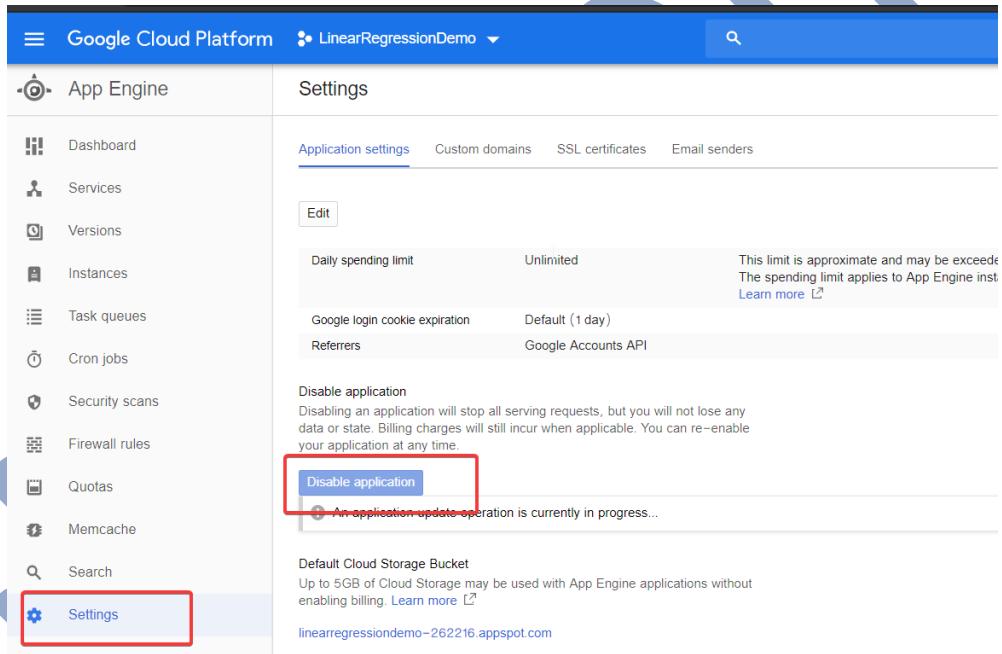
- GCP will ask for the services to be deployed. Enter 'y' to deploy the services.

- And then it will give you the link for your app, and the deployed app looks like:



The screenshot shows a web browser window with the address bar displaying `linearregressiondemo-262216.appspot.com`. The page has a light blue background and a title "Predict Your chances for Admission". Below the title, there is a form with four input fields arranged in a 2x2 grid. The first row contains the values 330, 120, and 5. The second row contains the values 5, 5, and 9.9. To the right of these fields is a dropdown menu currently set to "Yes". Below the form is a "Predict" button.

- To save money, go to settings and disable your app.



The screenshot shows the Google Cloud Platform console interface. The left sidebar contains a navigation menu with options like Dashboard, Services, Versions, Instances, Task queues, Cron jobs, Security scans, Firewall rules, Quotas, Memcache, Search, and Settings. The 'Settings' option is highlighted with a red box. The main content area is titled 'Settings' and has tabs for 'Application settings', 'Custom domains', 'SSL certificates', and 'Email senders'. The 'Application settings' tab is active. It shows an 'Edit' button and a table of settings. The 'Daily spending limit' is set to 'Unlimited'. Below this, there is a section titled 'Disable application' with a description and a 'Disable application' button, which is also highlighted with a red box. A message below the button states 'An application update operation is currently in progress...'. At the bottom, there is a section for 'Default Cloud Storage Bucket' and the application URL `linearregressiondemo-262216.appspot.com`.

# Thank You!