Data Glacier

Your Deep Learning Partner

Deploying a model using Flask

Name: Shruthi Madgi
Batch ID:LISUM17
Date:1/25/2023
Reviewer: Data Glacier

Table of Contents:

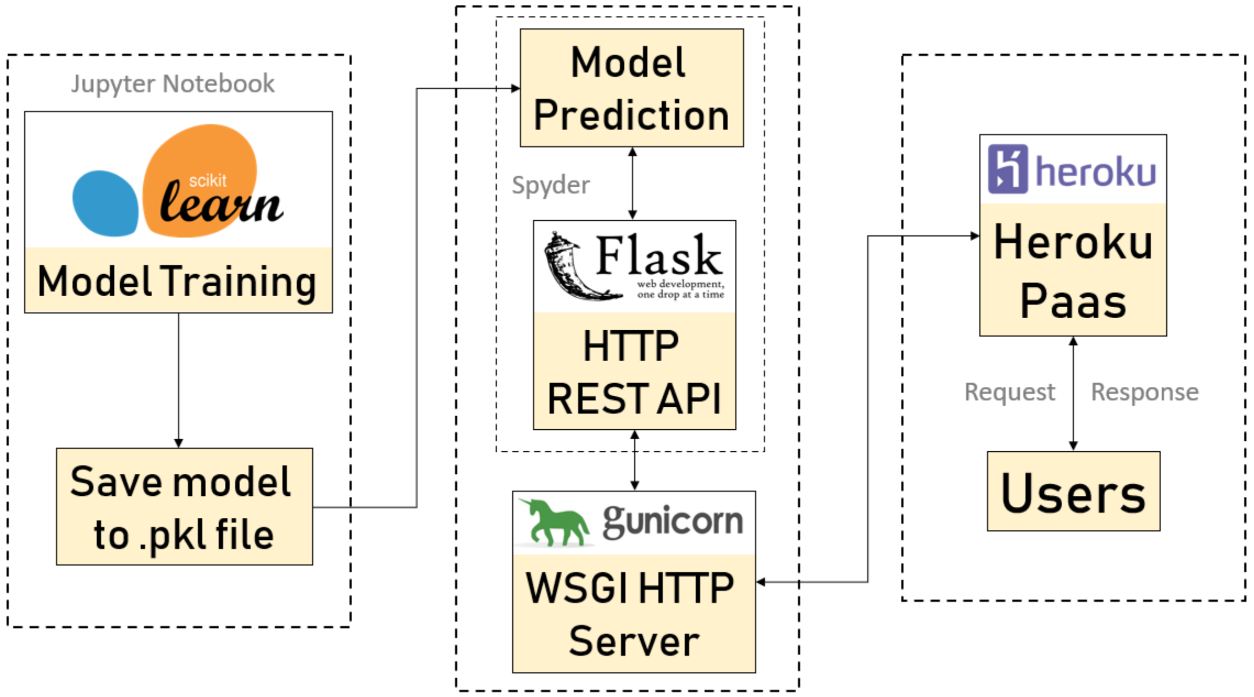
Data Glacier

Your Deep Learning Partner

Introduction

In this project, we are going to deploying a simple model on a local machine using the Flask Framework. As a demonstration, the model predicts the salary of a prospective, hirable employee.

The data for this model is small; yet trainable. The first part of the deployment is to create the model and then pickle it. Build an API for the model, using Flask, the Python micro-framework for building web applications. This API allows us to utilize predictive capabilities through HTTP requests..



Data Glacier

Flowchart: Life-cycle of Model Deployment

Data Information

The data was built from scratch so to as train the model easily and focus more on the Flask Deployment. Data contains just 9 entries.

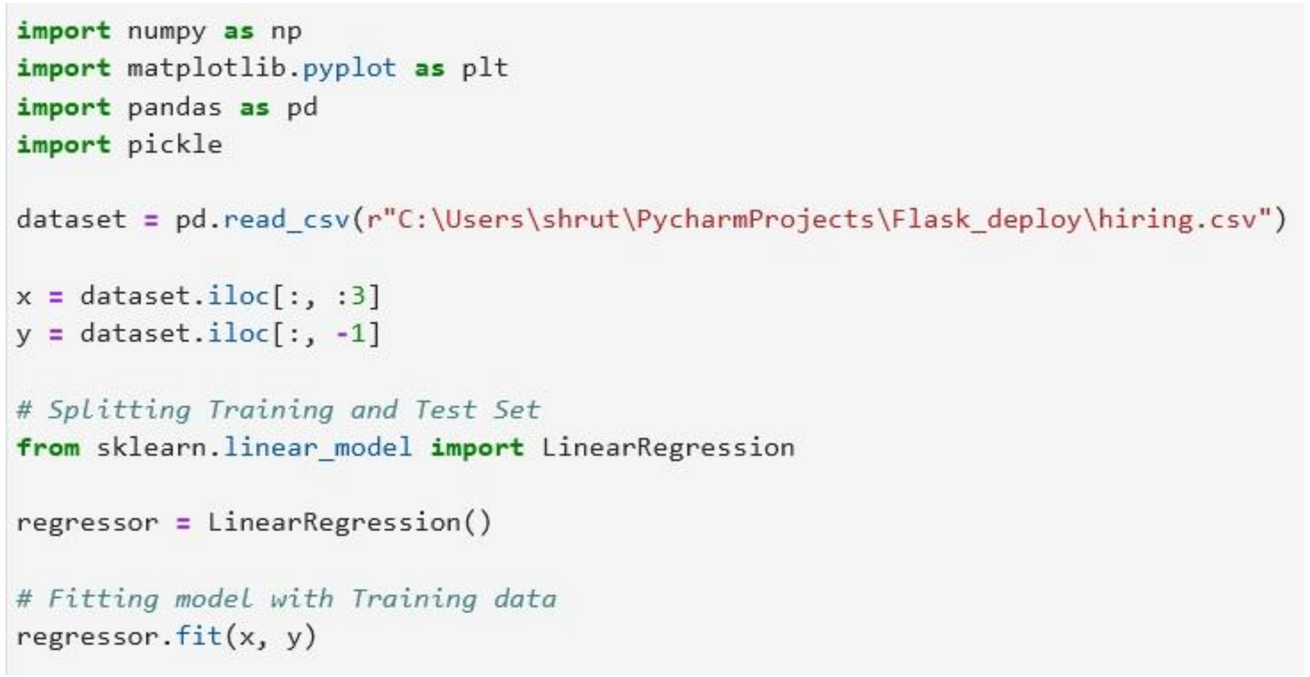
Attributes of the data are Experience, Test Score, Interview Score. Based on the given input, the results are returned, predicting salary for the employee.

Location: https://github.com/shru0405/DataGlacierInternship/blob/main/Week_4/hiring.csv

Even though the data is small, it is still trained. Linear regression is used.

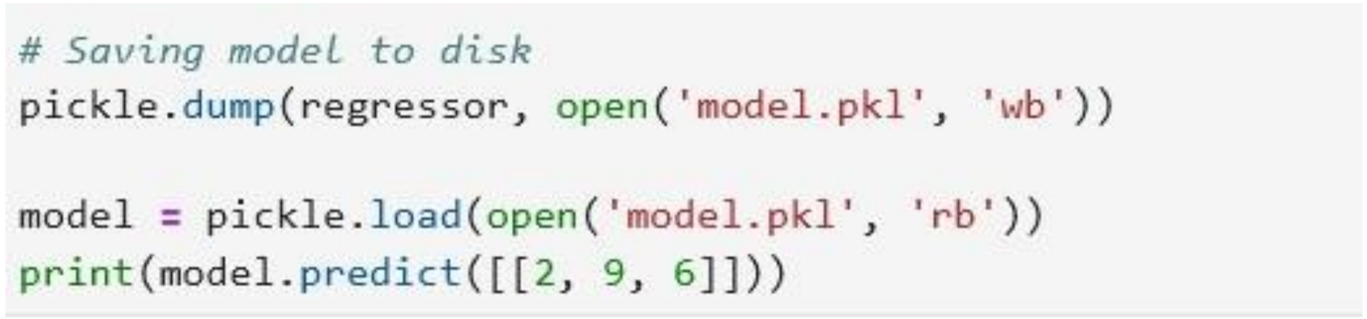
Build and Save Model

After data preprocessing, we implement Linear regression to train the data to predict the salary based on experience and interview.



```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pickle

dataset = pd.read_csv(r"C:\Users\shrut\PycharmProjects\Flask_deploy\hiring.csv")

x = dataset.iloc[:, :3]
y = dataset.iloc[:, -1]

# Splitting Training and Test Set
from sklearn.linear_model import LinearRegression

regressor = LinearRegression()

# Fitting model with Training data
regressor.fit(x, y)
```

Model is saved using Pickle.

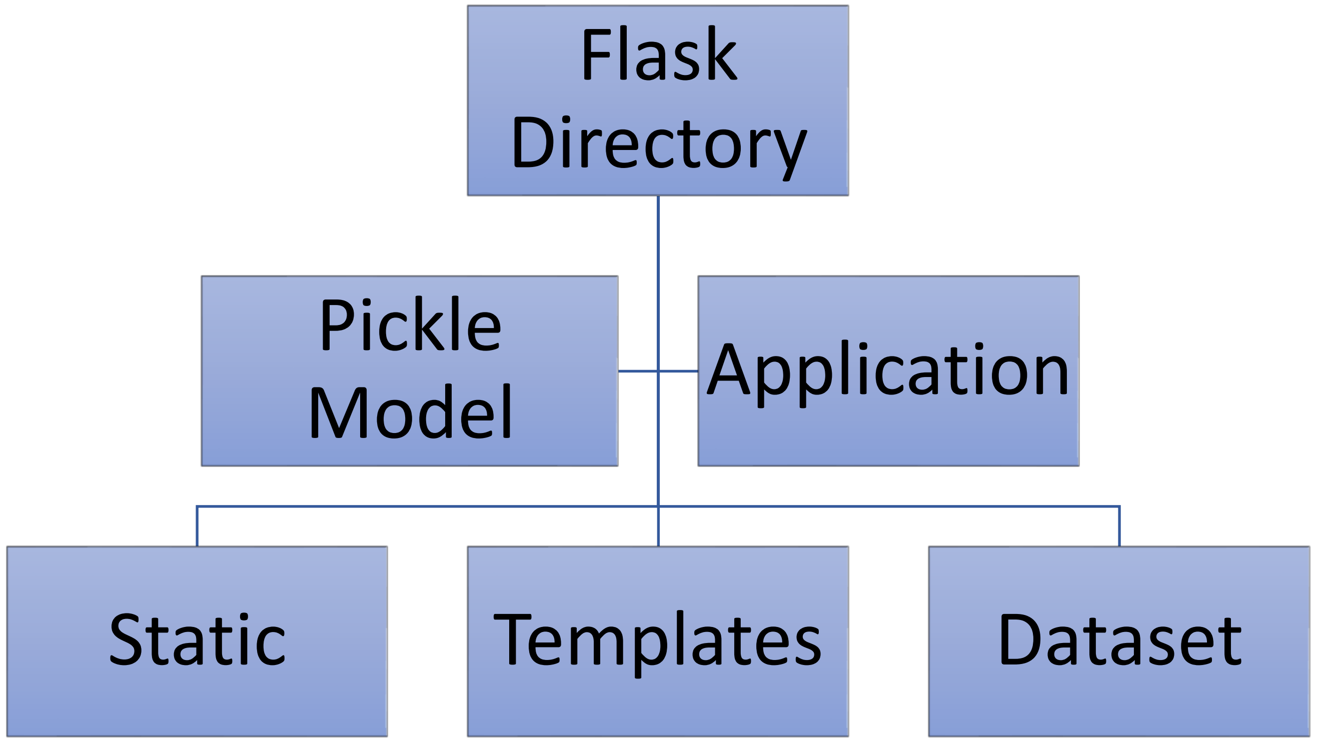


```python
# Saving model to disk
pickle.dump(regressor, open('model.pkl', 'wb'))

model = pickle.load(open('model.pkl', 'rb'))
print(model.predict([[2, 9, 6]]))
```

Turning Model into Web Application

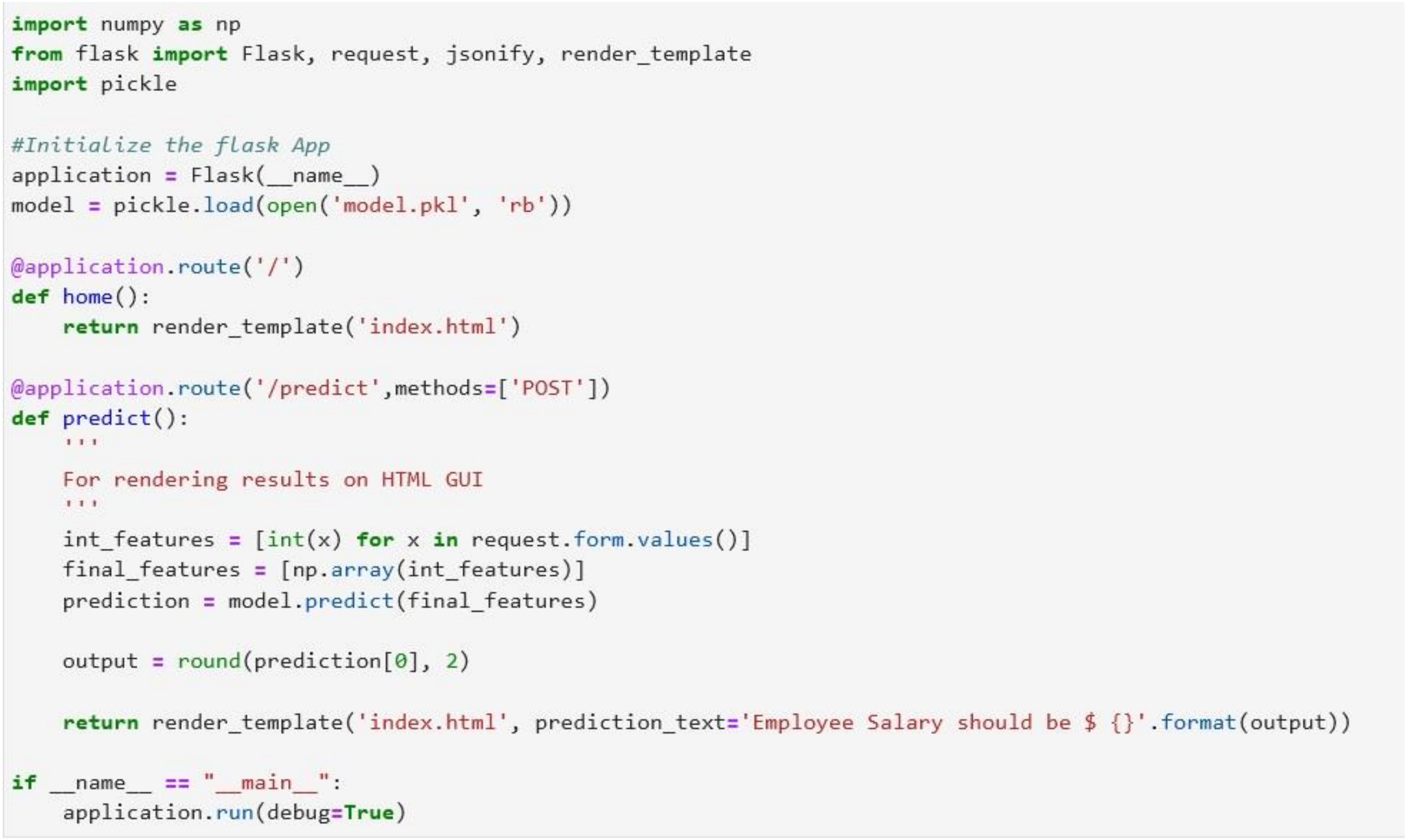
Using Flask Framework, a web application is developed that consists of a simple web page with a form field that lets us enter the values. After execution, it is rendered on a new web page, in this case on a local machine.

To implement the framework on a local machine, Flask will look for the static HTML in the code. Ensure that all the HTML files are saved under one main directory. The directory and sub directory hierarchy will look something like this:



Appl.py

The Appl.py file contains the main code that will be executed by the Python interpreter to run the Flask web application.



```python
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle

#Initialize the flask App
application = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))

@application.route('/')
def home():
    return render_template('index.html')

@application.route('/predict',methods=['POST'])
def predict():
    '''
    For rendering results on HTML GUI
    '''
    int_features = [int(x) for x in request.form.values()]
    final_features = [np.array(int_features)]
    prediction = model.predict(final_features)

    output = round(prediction[0], 2)

    return render_template('index.html', prediction_text='Employee Salary should be $ {}'.format(output))

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

Data Glacier

- The application is run as a single module and initialized a new Flask instance with the argument __*name*__ to let Flask know that it can find the HTML template folder (templates) in the same directory where it is located.

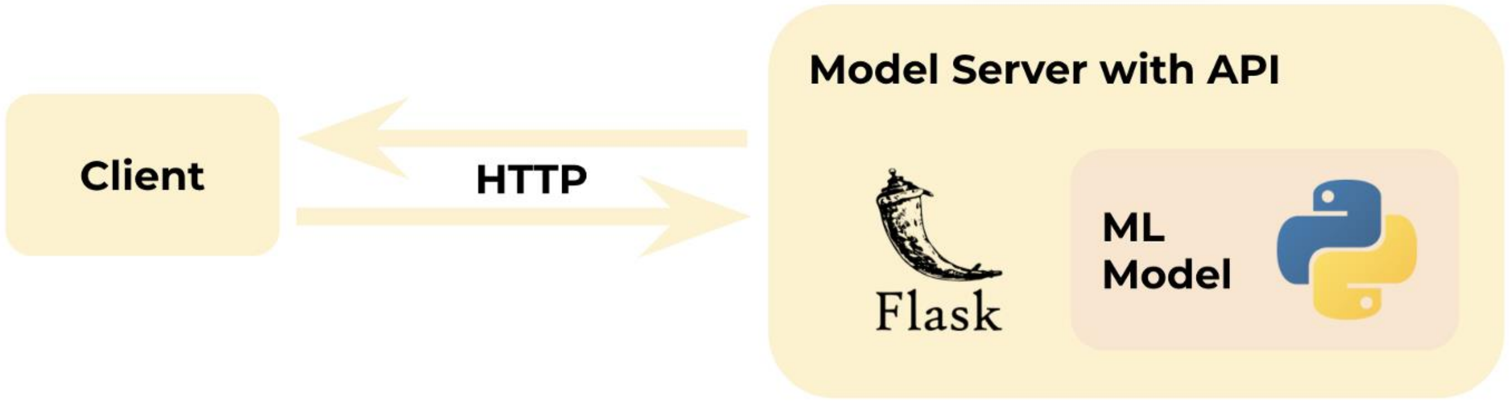
- The route decorator *(@app.route('/'))* is used to specify the URL that should trigger the execution of the home function.

- Home function simply rendered the index HTML file, which is located in the templates folder.

- Inside the predict function, the dataset is accessed, pre-processed. Prediction is made and then the model is stored. The new value is entered by the user and use our model to make a prediction for its label.

- The POST method is used to transport the form data to the server in the message body. By setting the debug=True argument inside the *application. Run* method, the Flask's debugger is activated.

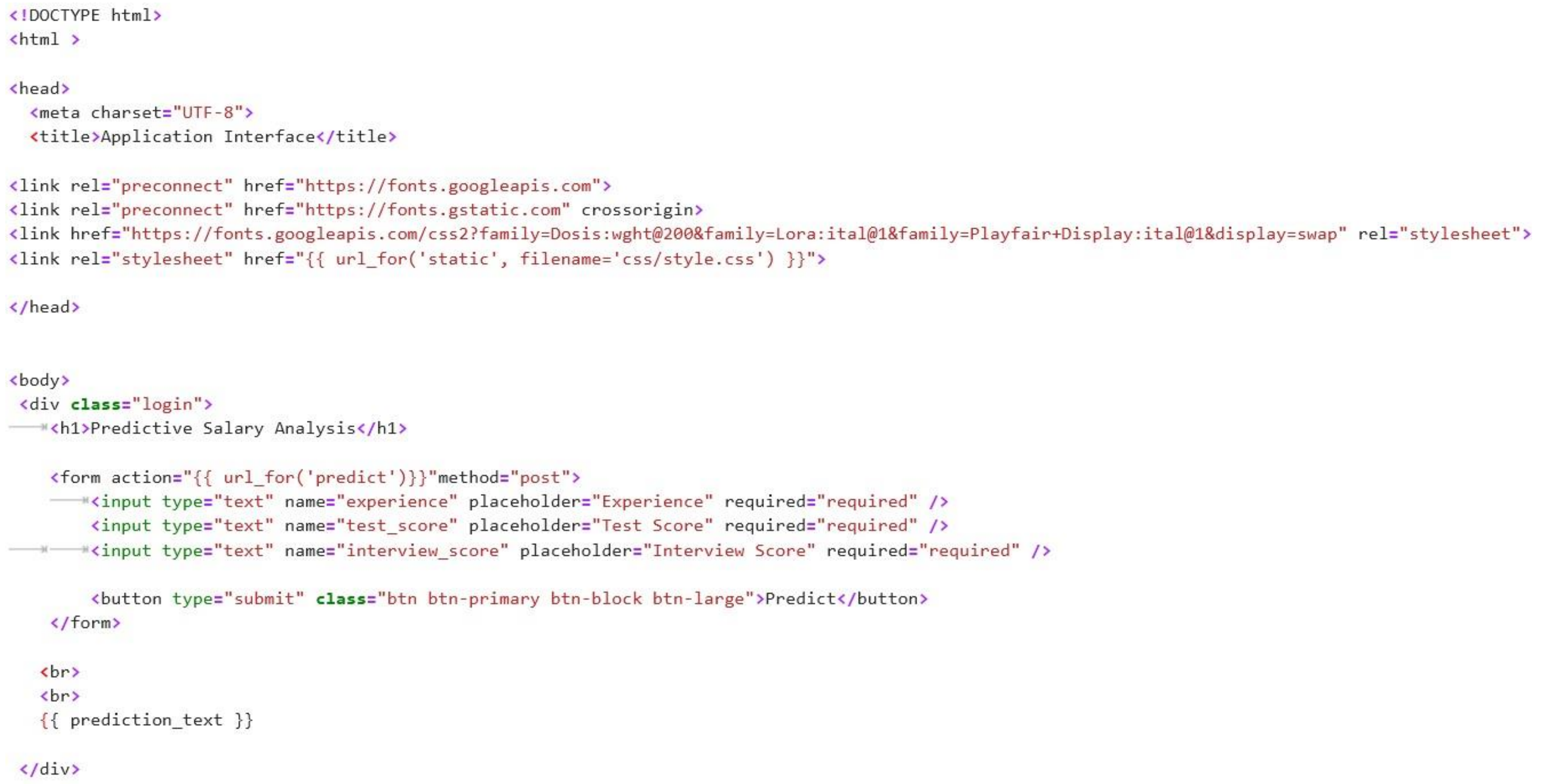
- The run function is used to only run the application on the server when this script is directly executed by the Python interpreter, which is done by using the if statement with __*name*__ == '__*main*__'.



Data Glacier

Index.html

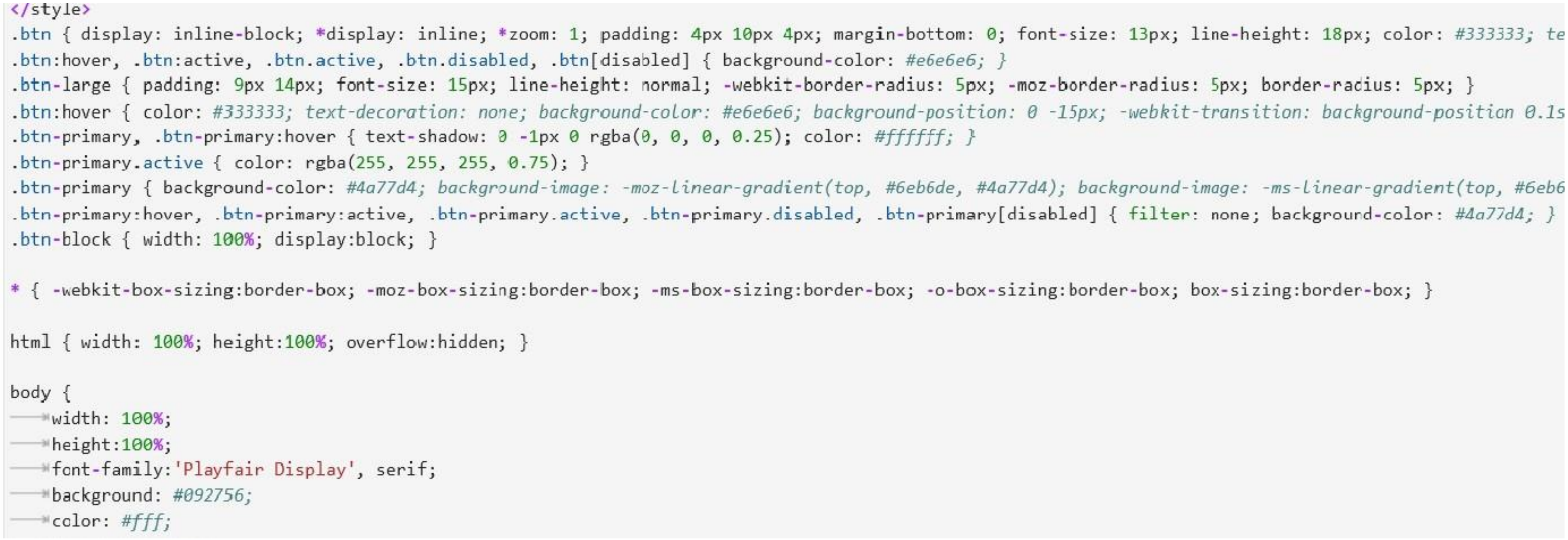
Index.html is an HTML file stored in template folder which will render a text form where a user can enter the values and get the prediction.



```
<!DOCTYPE html>
<html >

<head>
  <meta charset="UTF-8">
  <title>Application Interface</title>

<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Dosis:wght@200&family=Lora:ital@1&family=Playfair+Display:ital@1&display=swap" rel="stylesheet">
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">

</head>


<body>
 <div class="login">
    <h1>Predictive Salary Analysis</h1>

    <form action="{{ url_for('predict')}}"method="post">
        <input type="text" name="experience" placeholder="Experience" required="required" />
        <input type="text" name="test_score" placeholder="Test Score" required="required" />
        <input type="text" name="interview_score" placeholder="Interview Score" required="required" />

        <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
    </form>

   <br>
   <br>
   {{ prediction_text }}

 </div>
```

Style.css

In the header section of index.html, Style.css file is loaded. CSS determines the look and feel of HTML documents.
Style.css has to be saved in a sub-directory called static, which is the default directory where Flask looks for static files such as CSS.



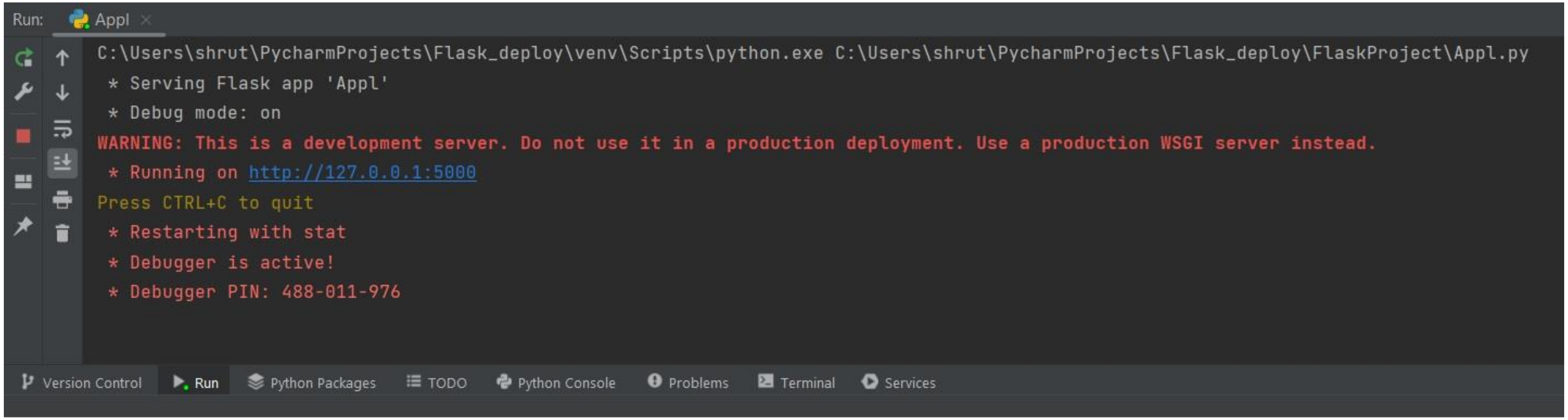
```
</style>
.btn { display: inline-block; *display: inline; *zoom: 1; padding: 4px 10px 4px; margin-bottom: 0; font-size: 13px; line-height: 18px; color: #333333; te
.btn:hover, .btn:active, .btn.active, .btn.disabled, .btn[disabled] { background-color: #e6e6e6; }
.btn-large { padding: 9px 14px; font-size: 15px; line-height: normal; -webkit-border-radius: 5px; -moz-border-radius: 5px; border-radius: 5px; }
.btn:hover { color: #333333; text-decoration: none; background-color: #e6e6e6; background-position: 0 -15px; -webkit-transition: background-position 0.1s
.btn-primary, .btn-primary:hover { text-shadow: 0 -1px 0 rgba(0, 0, 0, 0.25); color: #ffffff; }
.btn-primary.active { color: rgba(255, 255, 255, 0.75); }
.btn-primary { background-color: #4a77d4; background-image: -moz-linear-gradient(top, #6eb6de, #4a77d4); background-image: -ms-linear-gradient(top, #6eb6
.btn-primary:hover, .btn-primary:active, .btn-primary.active, .btn-primary.disabled, .btn-primary[disabled] { filter: none; background-color: #4a77d4; }
.btn-block { width: 100%; display:block; }

* { -webkit-box-sizing:border-box; -moz-box-sizing:border-box; -ms-box-sizing:border-box; -o-box-sizing:border-box; box-sizing:border-box; }

html { width: 100%; height:100%; overflow:hidden; }

body {
	width: 100%;
	height:100%;
	font-family:'Playfair Display', serif;
	background: #092756;
	color: #fff;
```

Data Glacier

Your Deep Learning Partner

Execution and Result

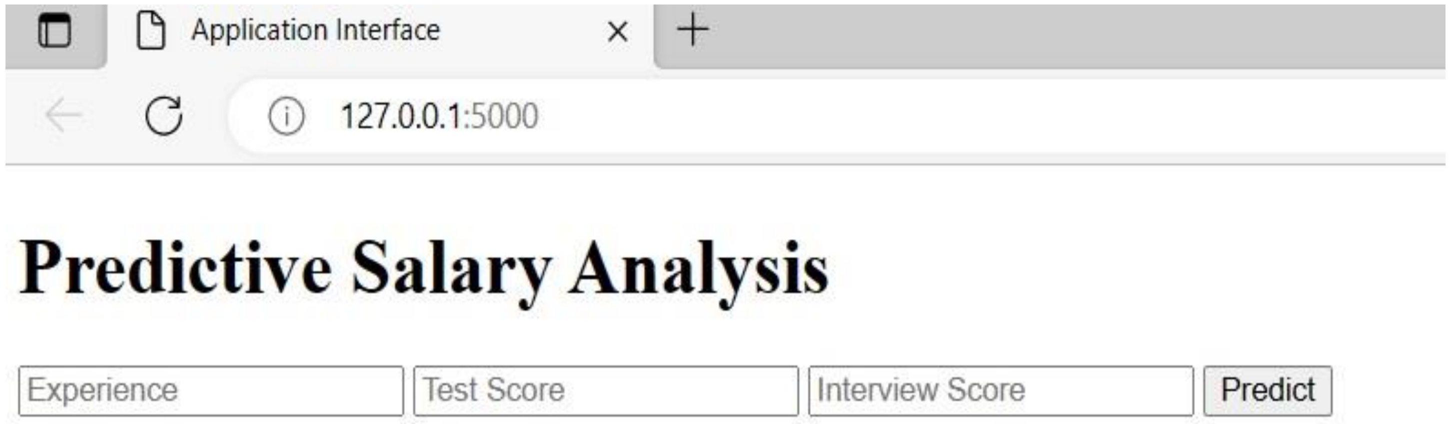
Run the API Appl.py either in the terminal or the IDE.



Run: Appl

```
C:\Users\shrut\PycharmProjects\Flask_deploy\venv\Scripts\python.exe C:\Users\shrut\PycharmProjects\Flask_deploy\FlaskProject\Appl.py
 * Serving Flask app 'Appl'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 488-011-976
```

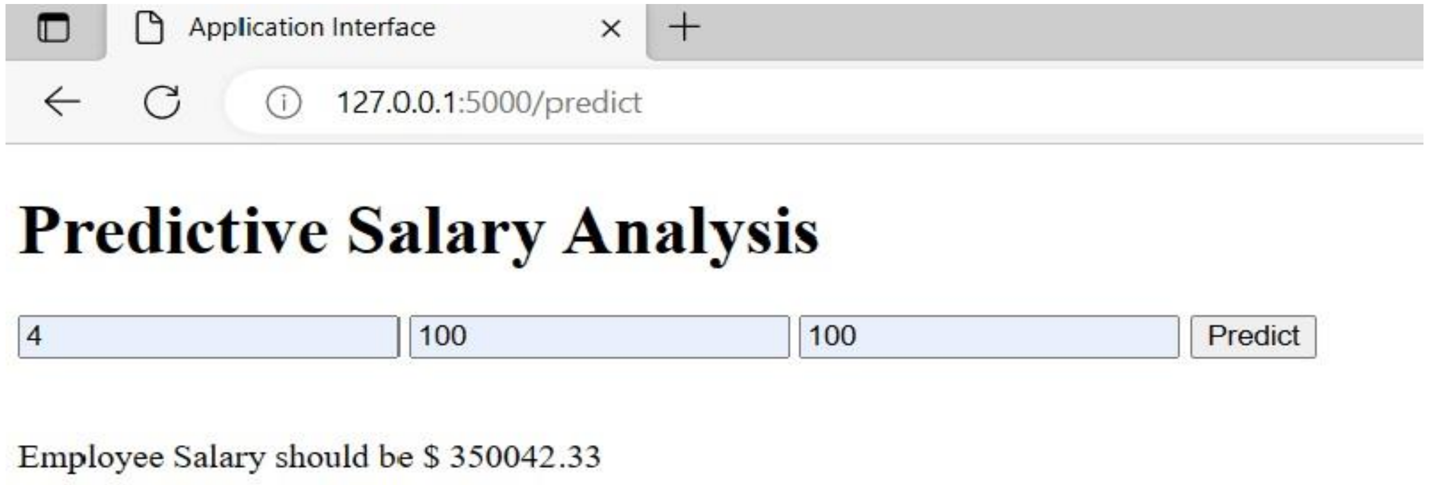
Since the Flask is deployed on the local host, navigate to the browser by clicking on the generated link.


Application Interface

127.0.0.1:5000

Predictive Salary Analysis

If all the steps are executed correctly, this is how the end result should look like. Entering the values will predict the salary.


Application Interface

127.0.0.1:5000/predict

Predictive Salary Analysis

4	100	100	Predict

Employee Salary should be $ 350042.33

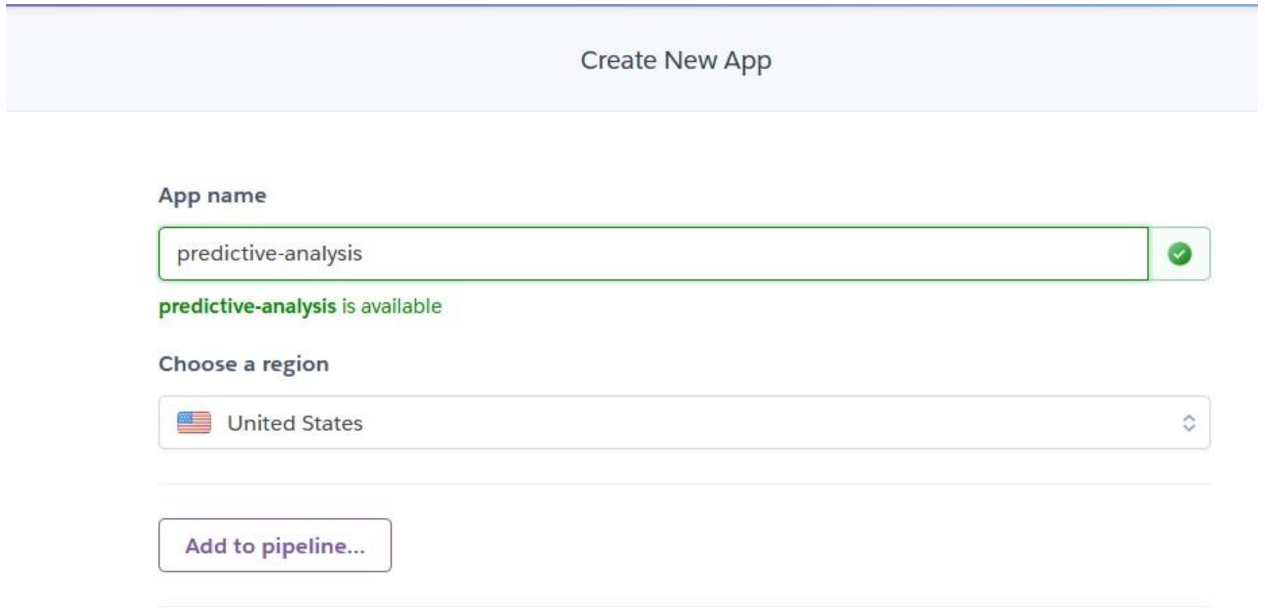
Model deployment using Heroku

To deploy a model on Heroku, there are two important files that need to be in the root directory of Github:

1. **Requirements.txt** ,which lists all the requirements of your Flask for Heroku to read. Requirements is generated by using   pipreqs or pip freeze command.
2.**Procfile**, which is a file without any extension. Heroku reads Procfile and web command in it to know what server to run for. The Procfile must contain web or worker command. Procfile is case sensitive.

Once the files are uploaded on to the GitHub repository, link the repository to the Github account. There are other ways to deploy the app on the web, but linking the GitHub is the easiest.

1. After sign up on Heroku.com then click on Create new app and name it.



Create New App

App name

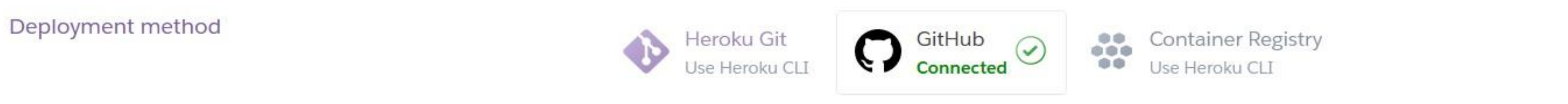
predictive-analysis

predictive-analysis is available

Choose a region

United States

Connect the repository where the files are saved.

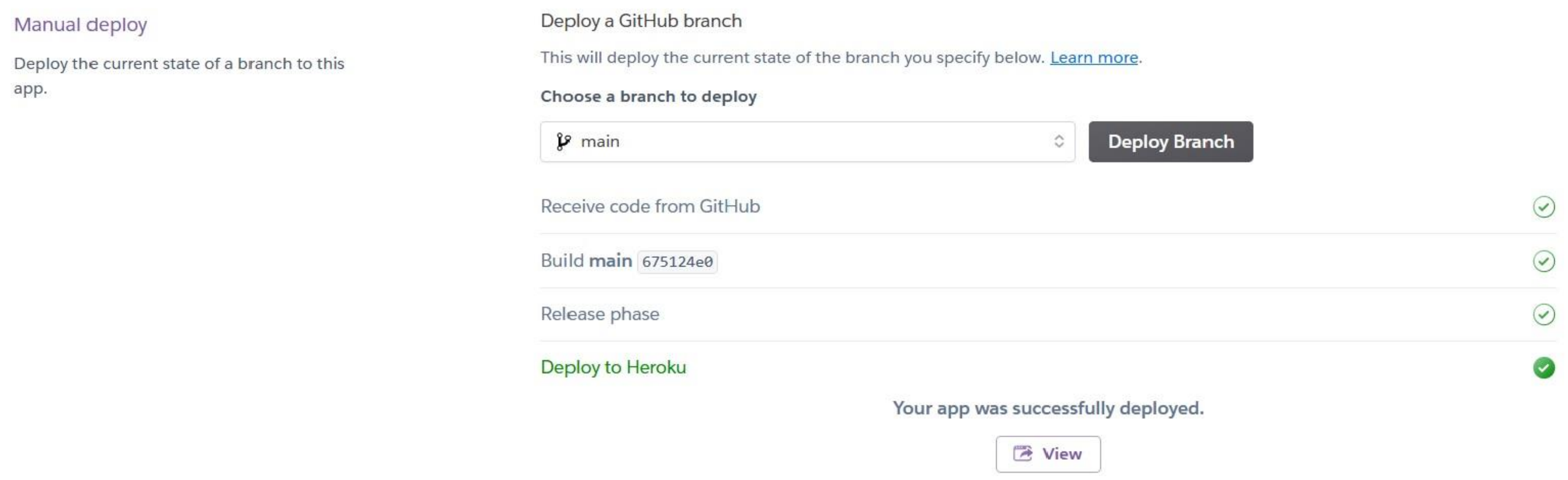

Deployment method

Heroku Git
Use Heroku CLI

GitHub
Connected

Container Registry
Use Heroku CLI

Deploy the Branch. After few minutes, if everything is executed properly, the app will be deployed.


Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. Learn more.

Choose a branch to deploy

main

Deploy Branch

Receive code from GitHub ✓

Build main 675124e0 ✓

Release phase ✓

Deploy to Heroku ✓

Your app was successfully deployed.

View

App is deployed at :https://predictive-analysis.herokuapp.com/

NOTE: As of Nov’22, Heroku is no longer free.