Data Glacier

Data Science Intern at Data Glacier

Week 5: Deployment on Flask
and Cloud (AWS)

Name: Nishchay Vaid

Batch Code: LISUM20

Date: 4 May 2023

Submitted to: Data Glacier

Table of Contents:

1. Introduction... 3
2. Data Information 4
   2.1. Attribute Information.. 4
3. Build Machine Learning Model 5
4. Turning Model into Flask Framework 6
   4.1. App.py 7
   4.2. Home.html 8
   4.3. Style.css 9
   4.4. Running Procedure... 10
5. Deploying Model on AWS
   5.1. Steps for deploying on AWS.. 10
   5.2. Output 13

1. Introduction

In this project, we are going to deploying machine learning model (Linear Regression) using the Flask Framework. We are trying to build a machine learning model that can predict the salary of an employee based on his/her years of experience, level of education, and performance score in an interview.

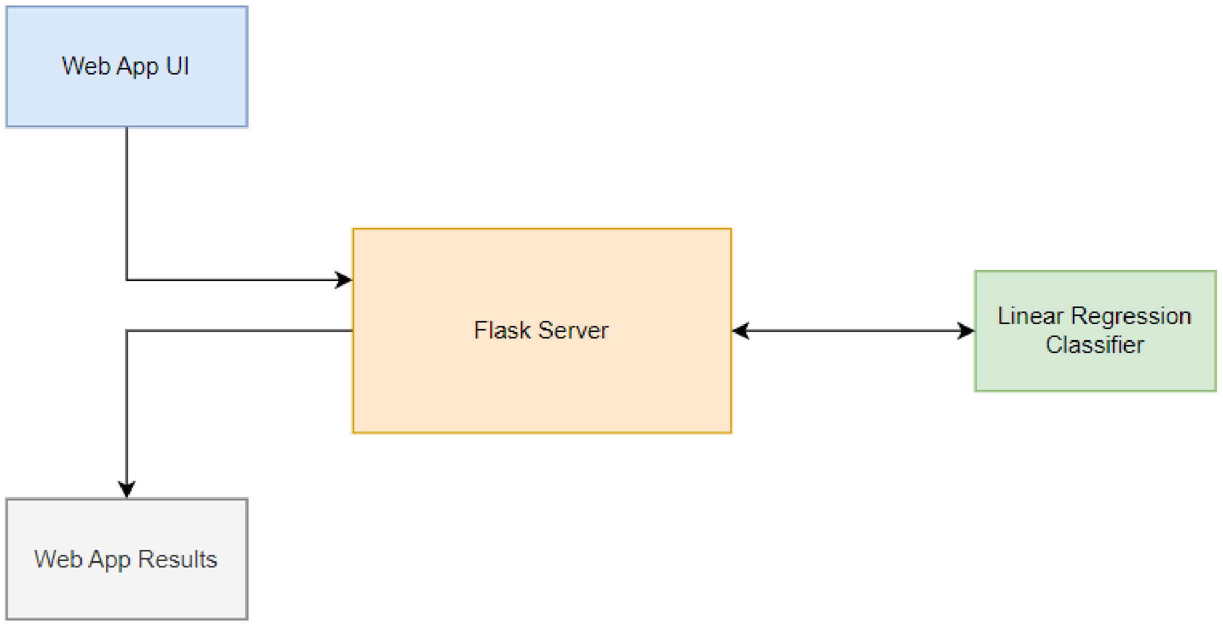


Figure 1.1: Application Workflow

Our main objective is to accomplish two tasks. First, we aim to construct a machine learning model that can be used to analyze the sentiment of YouTube comments. Second, we plan to develop an application programming interface (API) for the model using Flask, which is a lightweight web application framework written in Python. This API will enable us to utilize the predictive capabilities of the model by sending HTTP requests.

2. Data Information

Table 2.1: Dataset Information

experience	test_score	interview_score	salary
null	8	9	50000
null	8	6	45000
five	6	7	60000
two	10	10	65000
seven	9	6	70000
three	7	10	62000
ten	null	7	72000
eleven	7	8	80000

2.1 Attribute Information

The collection is composed of one CSV file per dataset, where each line has the following attributes:

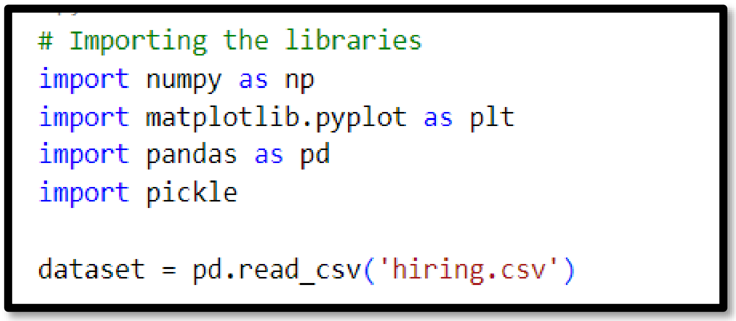
Table 2.2: Attribute Information

Attributes	Example (1 instance)
experience	two
test_score	10
interview_score	10
salary	65000

3. Building a Model

3.1 Import Required Libraries and Dataset

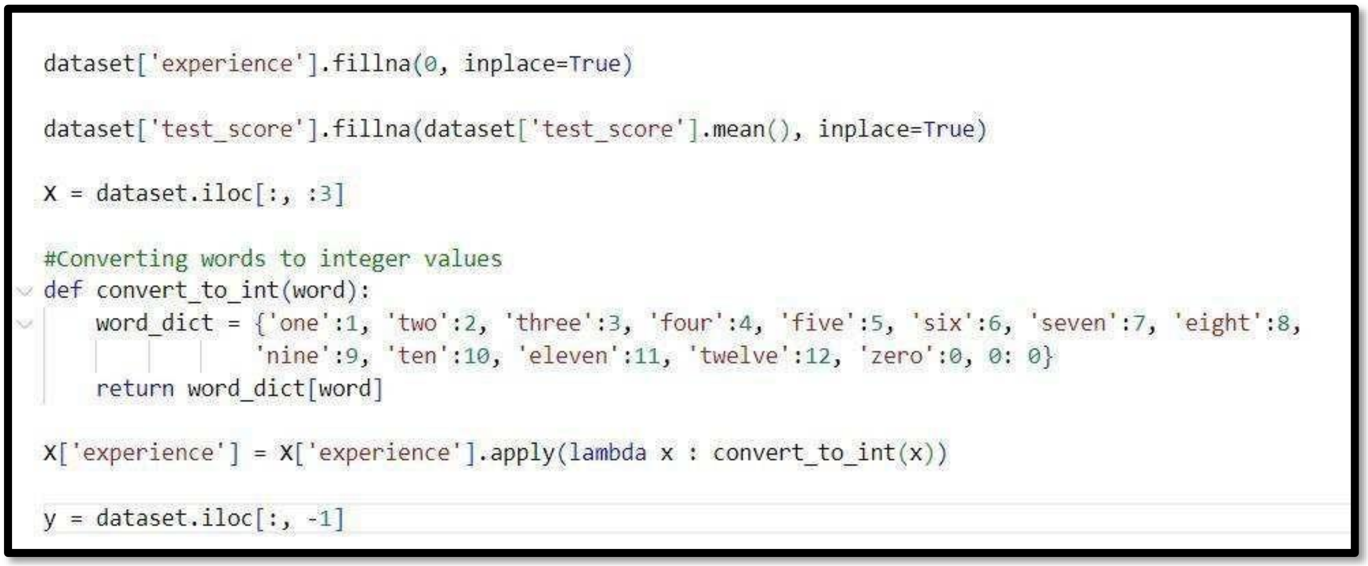
The code below imports necessary libraries such as NumPy, Matplotlib, Pandas, and Pickle. Then it reads a CSV file 'hiring.csv' and loads the data into a pandas dataframe called 'dataset'.



```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pickle

dataset = pd.read_csv('hiring.csv')
```

3.2 Data Preprocessing

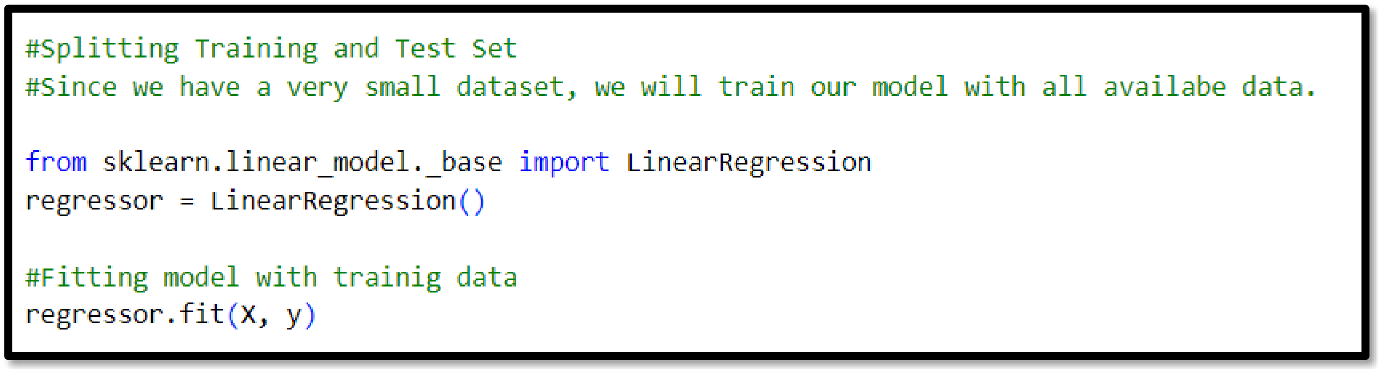
The code performs data preprocessing on a dataset containing information about job applicants. It fills missing values in the 'experience' column with zeros and missing values in the 'test_score' column with the mean score. The 'experience' column is then converted from string to integer values using a dictionary and a conversion function. Finally, the input and output variables for the machine learning model are created, with the input variables being the first three columns of the dataset and the output variable being the last column.



```python
dataset['experience'].fillna(0, inplace=True)

dataset['test_score'].fillna(dataset['test_score'].mean(), inplace=True)

X = dataset.iloc[:, :3]

#Converting words to integer values
def convert_to_int(word):
    word_dict = {'one':1, 'two':2, 'three':3, 'four':4, 'five':5, 'six':6, 'seven':7, 'eight':8,
                'nine':9, 'ten':10, 'eleven':11, 'twelve':12, 'zero':0, 0: 0}
    return word_dict[word]

X['experience'] = X['experience'].apply(lambda x : convert_to_int(x))

y = dataset.iloc[:, -1]
```

3.3 Build a Model

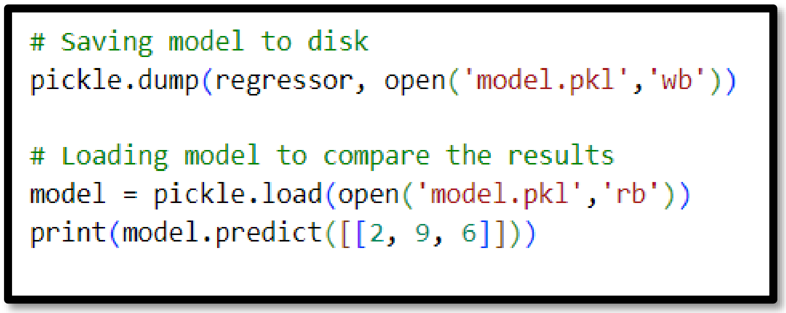
The code below splits the dataset into training and testing sets, but since the dataset is very small, the model will be trained on all available data. It then imports the LinearRegression class from the Scikit-learn library and creates an instance of it. Finally, the model is fit to the training data using the fit() method.



```python
#Splitting Training and Test Set
#Since we have a very small dataset, we will train our model with all availabe data.

from sklearn.linear_model._base import LinearRegression
regressor = LinearRegression()

#Fitting model with trainig data
regressor.fit(X, y)
```

3.4 Save the Model

After that we save our model using pickle



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

# Loading model to compare the results
model = pickle.load(open('model.pkl','rb'))
print(model.predict([[2, 9, 6]]))
```

4. Turning Model into Web Application

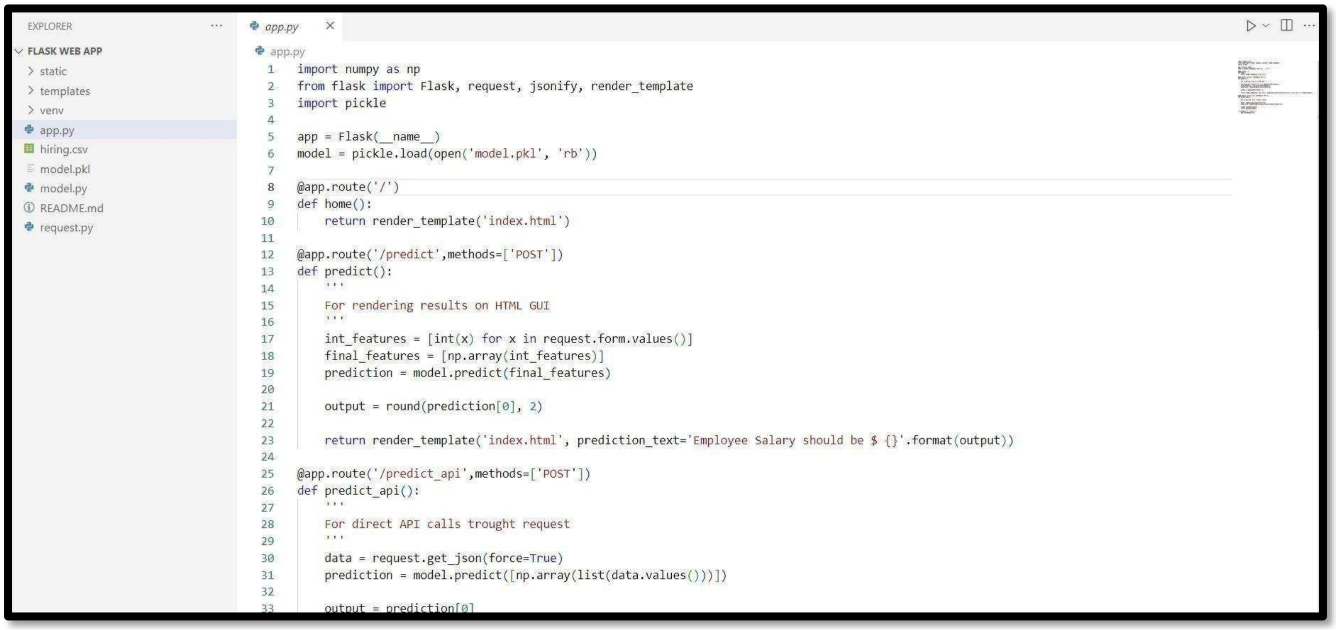
Table 3.1: Application Folder File Directory

```
app.py
template/
              index.html
static/css/
        style.css

model.pkl

hiring.csv

request.py
```

4.1 App.py

The app.py file contains the code that will be executed by the Python interpreter to run the Flask web application. The code creates a web application using Flask, a Python web application framework, and a machine learning model to predict employee salary.

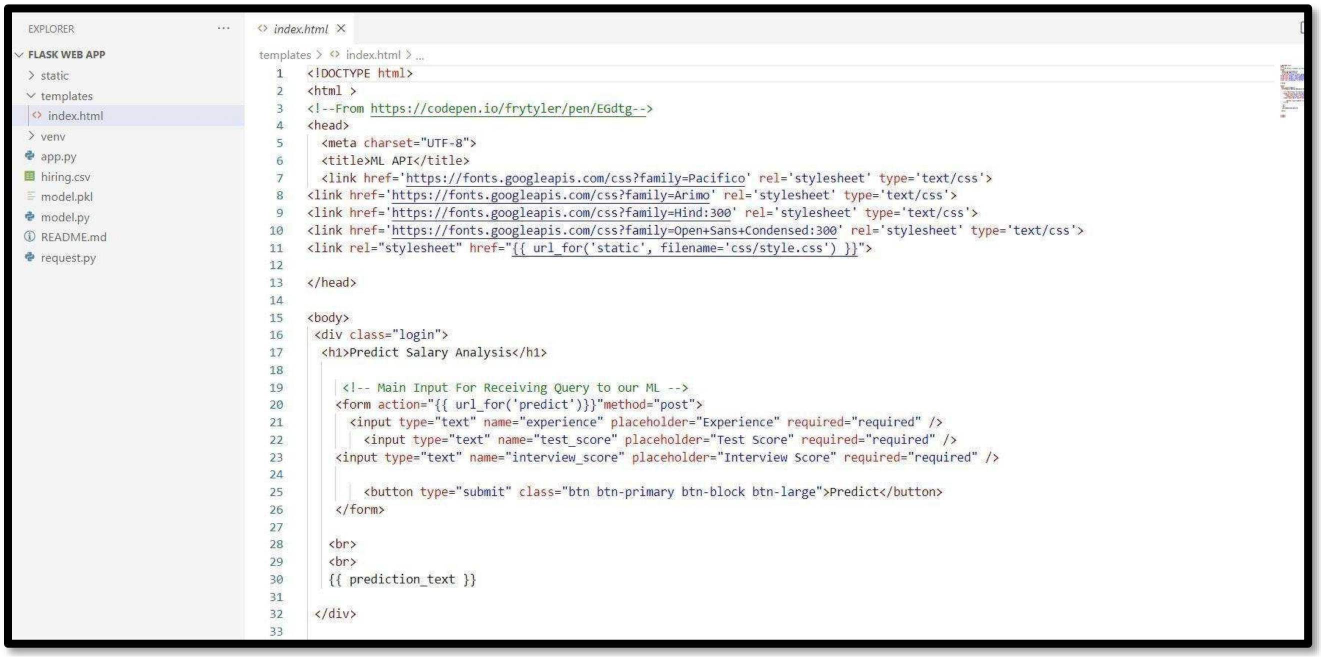
The model is loaded from a saved file, and two routes are defined to handle HTTP requests. The '/predict' route accepts data submitted through an HTML form and returns a predicted salary value. The '/predict_api' route accepts data submitted in JSON format and returns a salary prediction directly. The app.py file also includes a home route that renders an HTML template for the user to submit data. The output from the model prediction is rendered on the same page.



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

app = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))

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

@app.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))

@app.route('/predict_api',methods=['POST'])
def predict_api():
    '''
    For direct API calls trought request
    '''
    data = request.get_json(force=True)
    prediction = model.predict([np.array(list(data.values()))])

    output = prediction[0]
```

4.2 index.html

The following are the contents of the index.html file that will render a text form where a user can enter his values.



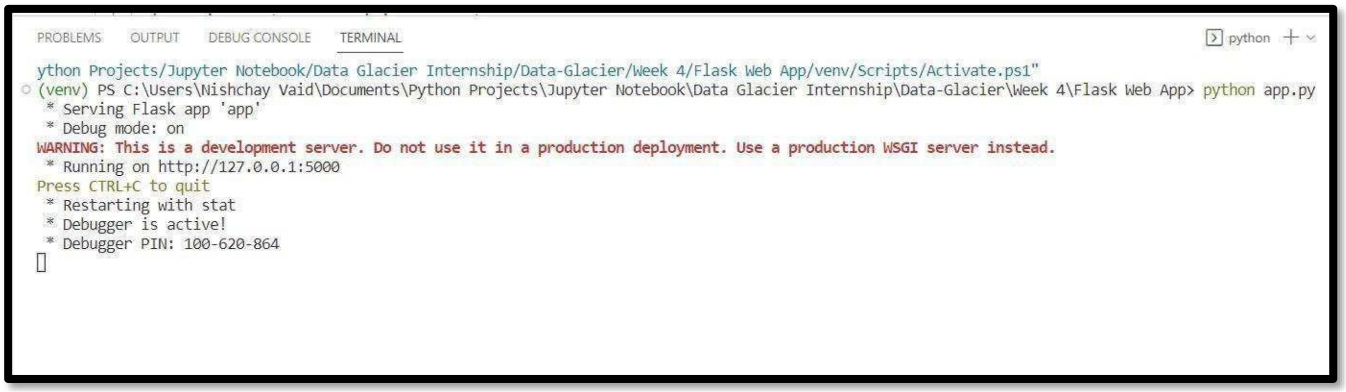
```html
<!DOCTYPE html>
<html >
<!--From https://codepen.io/frytyler/pen/EGdtg-->
<head>
  <meta charset="UTF-8">
  <title>ML API</title>
  <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">

</head>

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

     <!-- Main Input For Receiving Query to our ML -->
    <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>

```

4.3 Style.css

The index.html file includes a header section where the styles.css file is loaded. This file contains the styling information for the HTML document, and it should be saved in the 'static' sub-directory. Flask automatically looks for static files, such as CSS, in this directory.

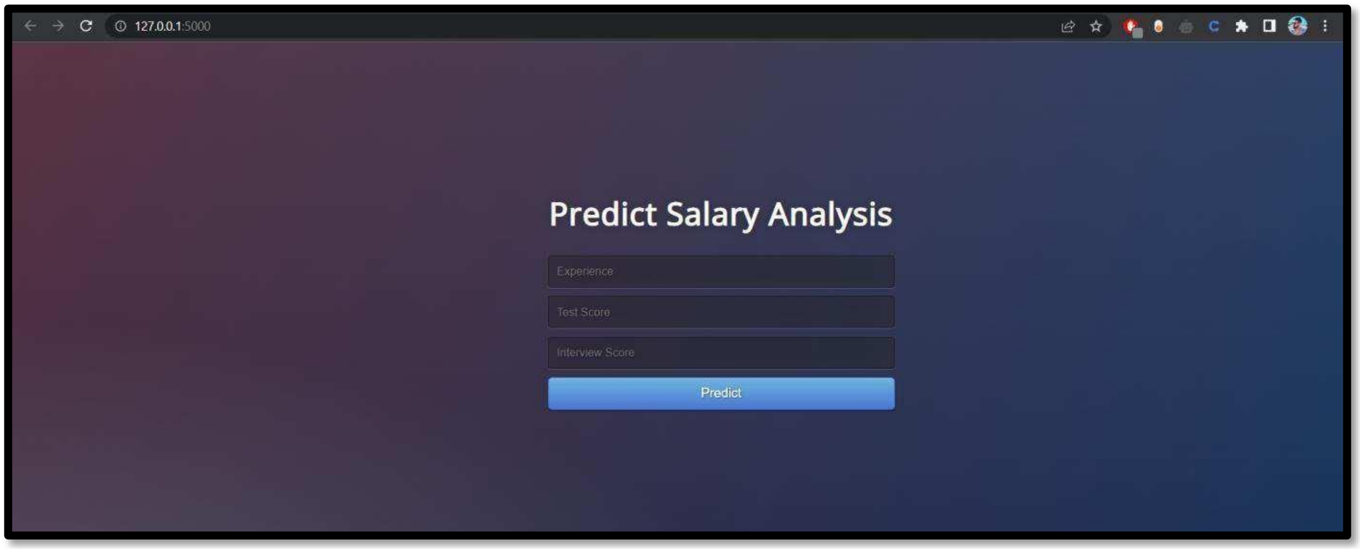
4.4 Running Procedure

Now, to start the Flask server we use the terminal and type in the command python app.py

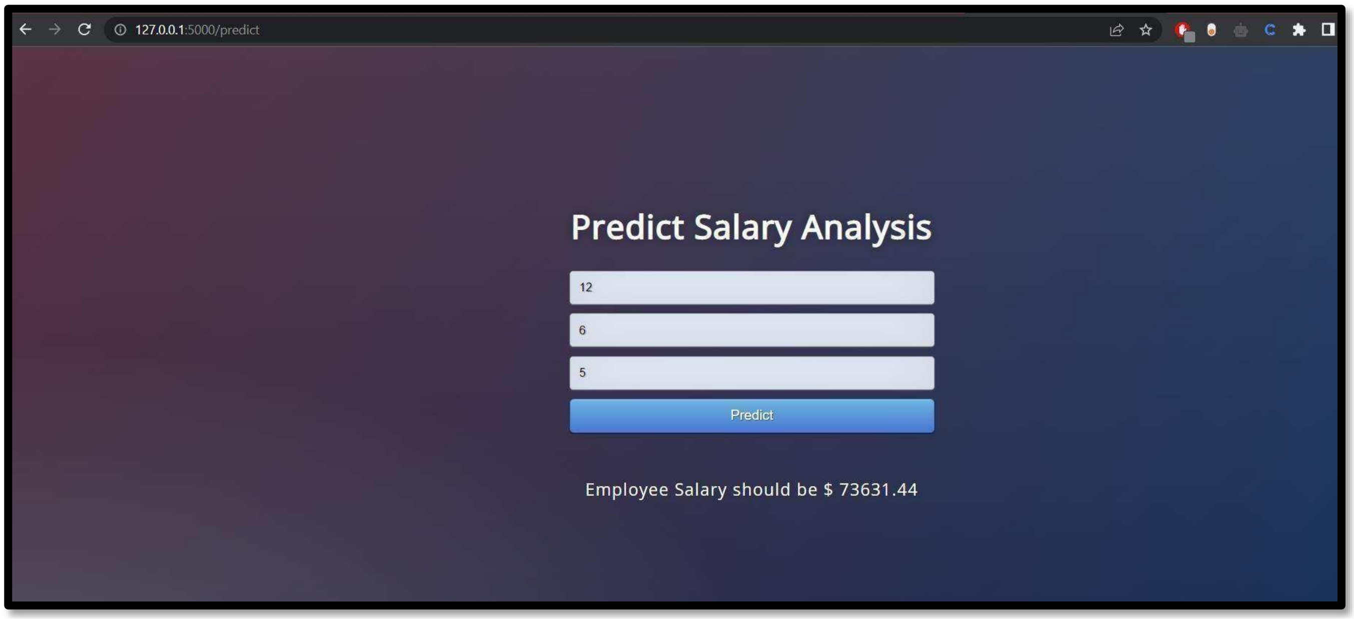


```
ython Projects/Jupyter Notebook/Data Glacier Internship/Data-Glacier/Week 4/Flask Web App/venv/Scripts/Activate.ps1"
(venv) PS C:\Users\Nishchay Vaid\Documents\Python Projects\Jupyter Notebook\Data Glacier Internship\Data-Glacier\Week 4\Flask Web App> python app.py
 * Serving Flask app 'app'
 * Debug 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: 100-620-864
```

After running the web application, we can open a web browser and enter the URL 'http://127.0.0.1:5000/'. This should display a basic website with a simple layout and design.



Now we enter input in the comments form and get the predicted employee salary

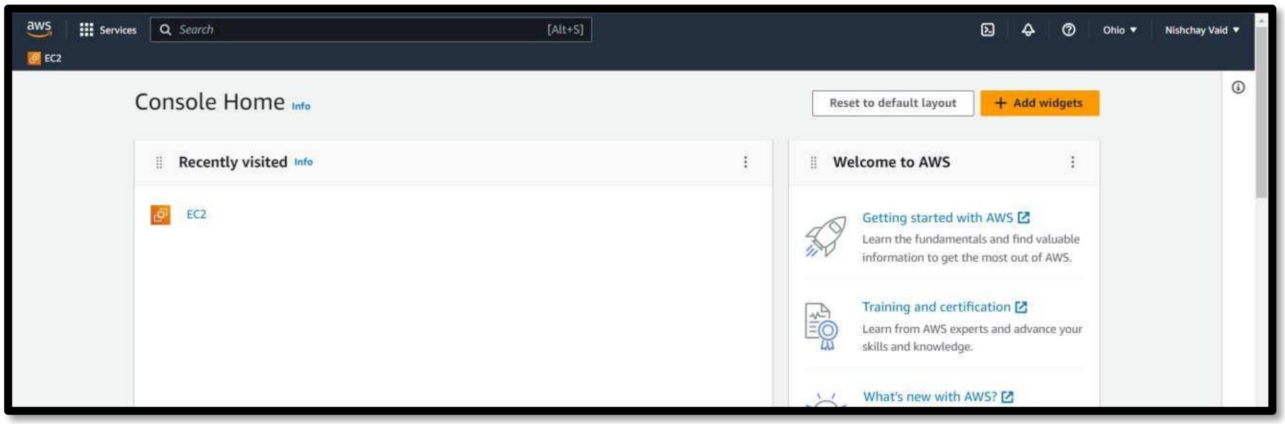


Employee Salary should be $ 73631.44

5. Deploying Model on AWS

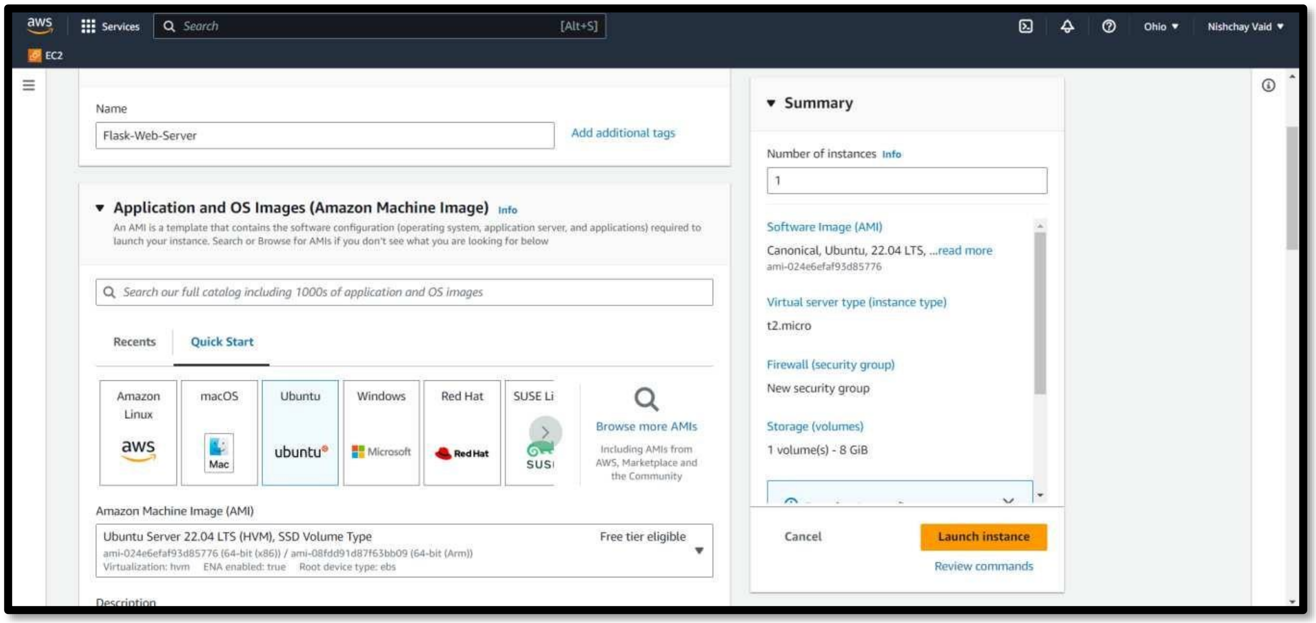
5.1 Steps for deploying on AWS

Create a new **EC2 Instance on AWS**. This will serve as the deployment instance on which we will upload our flask web app.



Learn the fundamentals and find valuable information to get the most out of AWS.

Learn from AWS experts and advance your skills and knowledge.



Name
Flask-Web-Server

Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Amazon Machine Image (AMI)
Ubuntu Server 22.04 LTS (HVM), SSD Volume Type — Free tier eligible
ami-024e6efaf93d85776 (64-bit (x86)) / ami-08fdd91d87f63bb09 (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

Description

Summary

Number of instances Info
1

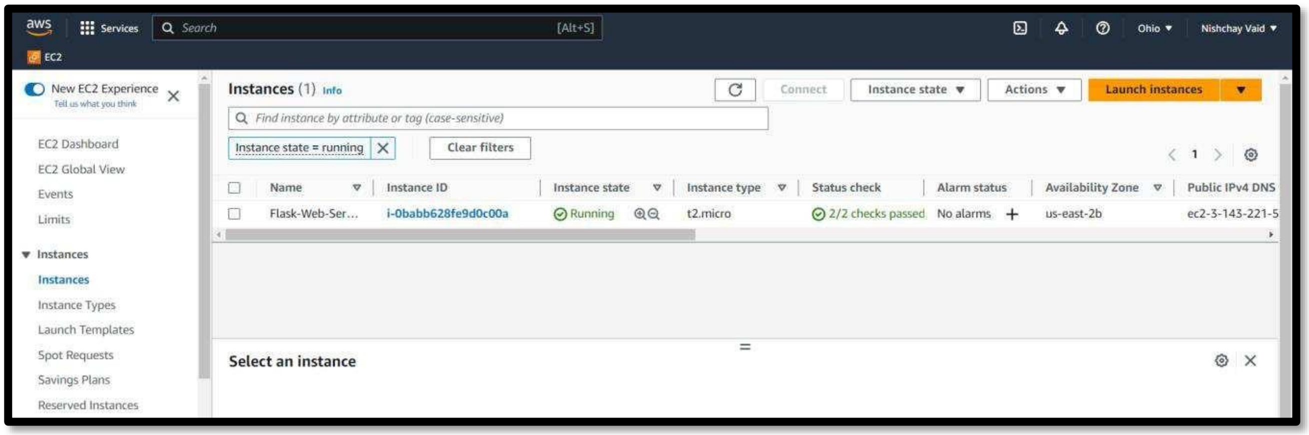
Software Image (AMI)
Canonical, Ubuntu, 22.04 LTS, ...read more
ami-024e6efaf93d85776

Virtual server type (instance type)
t2.micro

Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 8 GiB

Once configured, the instance should be up and running as shown below:



Instances (1) Info

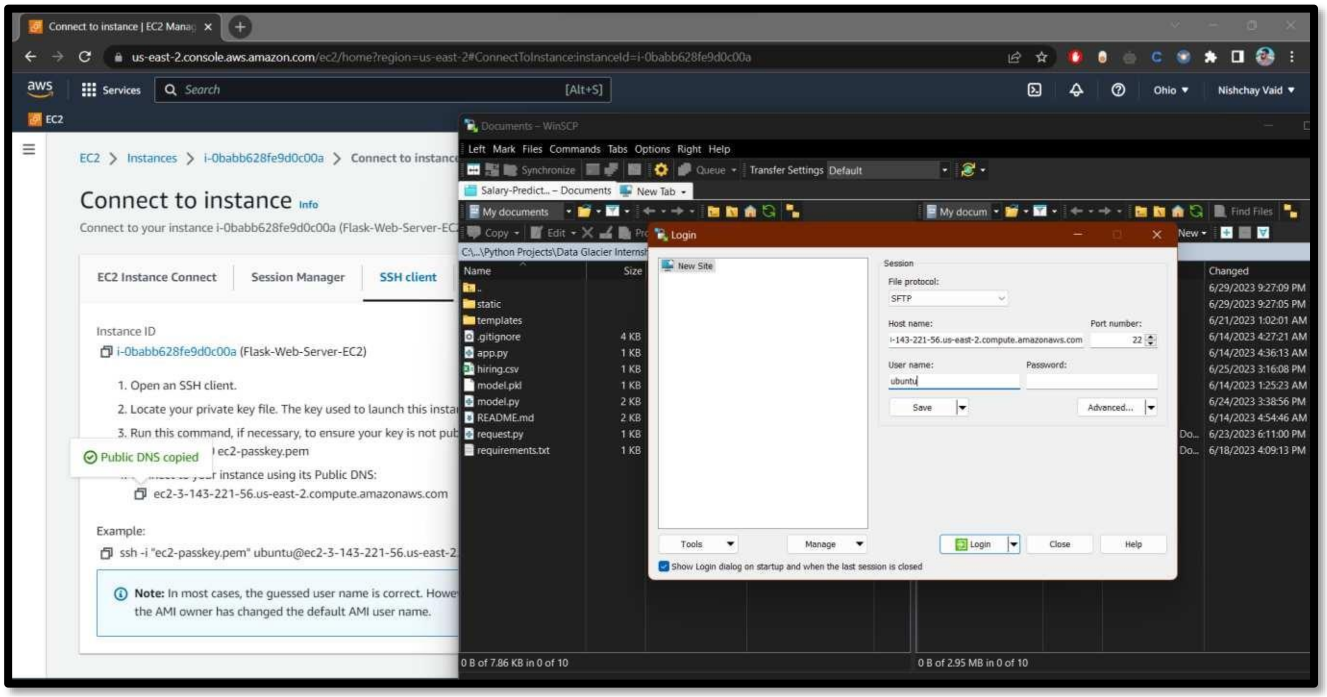
Find instance by attribute or tag (case-sensitive)

Instance state = running

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
Flask-Web-Ser...	i-0babb628fe9d0c00a	Running	t2.micro	2/2 checks passed	No alarms	us-east-2b	ec2-3-143-221-5

Now, we need to communicate with this EC2 Instance, so download WinSCP for FTP management and PuTTy for accessing the command line interface (CLI).

After downloading the above mentioned softwares, use the SSH key and domain name to access the EC2 instance we just created.



Connect to instance

Connect to your instance i-0babb628fe9d0c00a (Flask-Web-Server-EC

EC2 Instance Connect | Session Manager | SSH client

Instance ID
i-0babb628fe9d0c00a (Flask-Web-Server-EC2)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this insta
3. Run this command, if necessary, to ensure your key is not pub
ec2-passkey.pem
4. Connect to your instance using its Public DNS:
ec2-3-143-221-56.us-east-2.compute.amazonaws.com

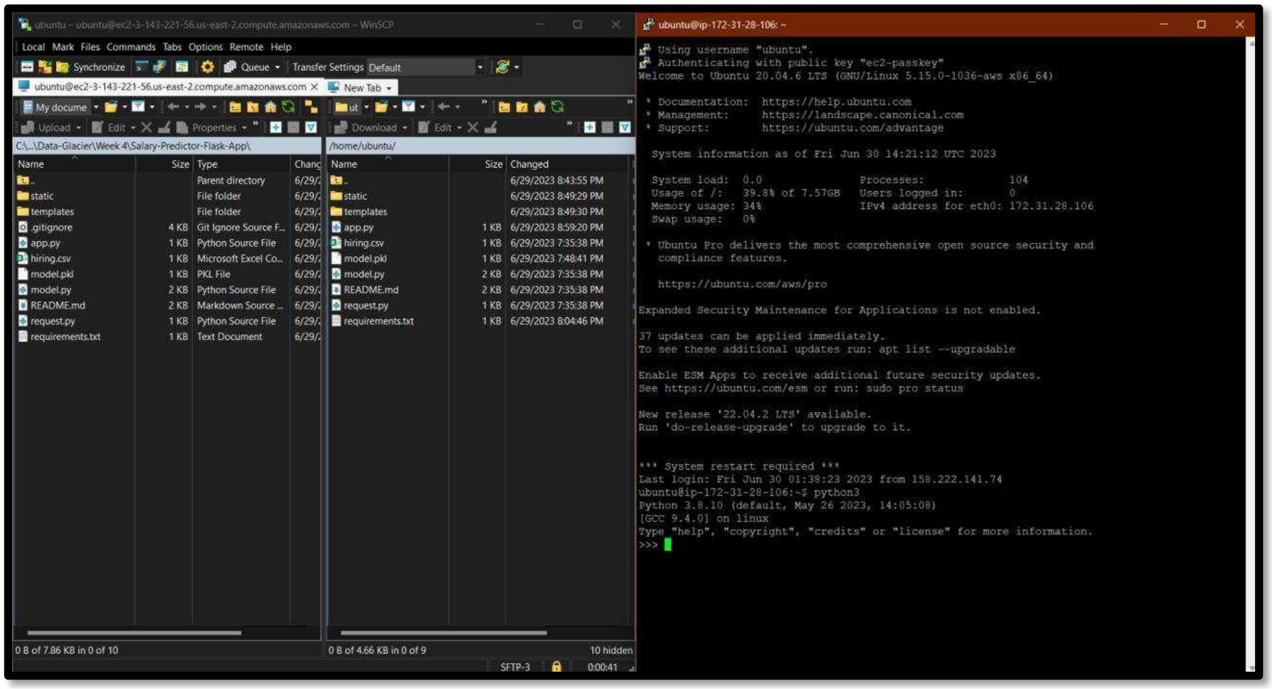
Example:
ssh -i "ec2-passkey.pem" ubuntu@ec2-3-143-221-56.us-east-2.

Note: In most cases, the guessed user name is correct. Howe
the AMI owner has changed the default AMI user name.

Login

File protocol: SFTP
Host name: i-143-221-56.us-east-2.compute.amazonaws.com
Port number: 22
User name: ubuntu
Password:

Once logged into the EC2 instance using WinSCP, also open the PuTTy shell.
Now upload the flask web app from local machine to the AWS EC2 machine.
Please find both of them opened side by side in the screenshot below :

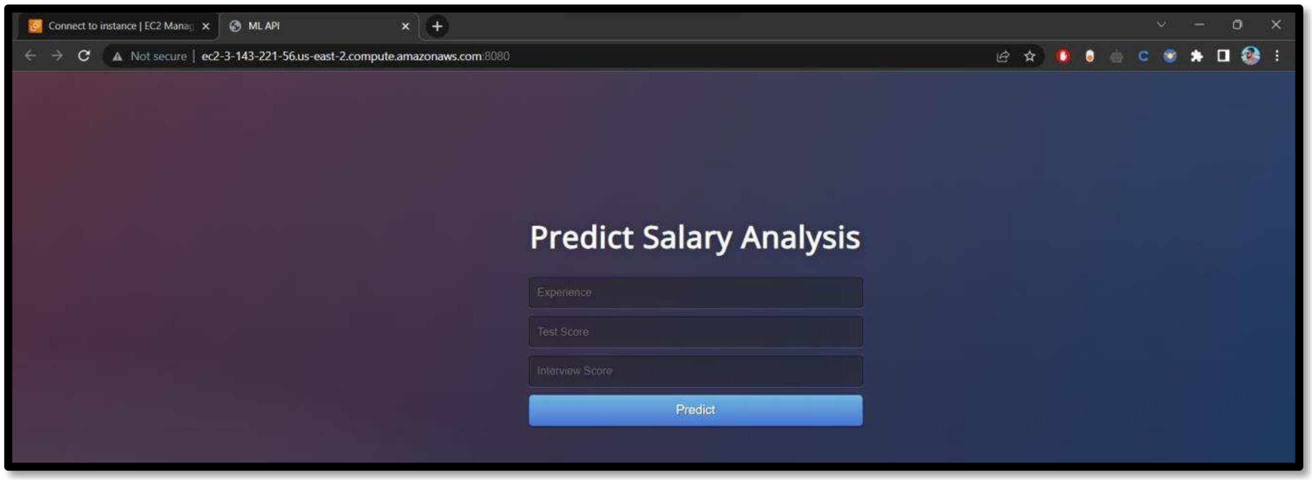


```
Using username "ubuntu".
Authenticating with public key "ec2-passkey"
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-1036-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Fri Jun 30 14:21:12 UTC 2023

  System load:  0.0               Processes:             104
  Usage of /:   39.8% of 7.57GB   Users logged in:       0
  Memory usage: 34%               IPv4 address for eth0: 172.31.28.106
  Swap usage:   0%

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.

   https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

37 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

New release '22.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


*** System restart required ***
Last login: Fri Jun 30 01:38:23 2023 from 158.222.141.74
ubuntu@ip-172-31-28-106:~$ python3
Python 3.8.10 (default, May 26 2023, 14:05:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Now run the flask app using PuTTy terminal to turn on the flask server. Once it starts, our application is now live on the EC2 Instance.

5.2 Output

http://ec2-3-143-221-56.us-east-2.compute.amazonaws.com:8080/



Connect to instance | EC2 Manag
ML API
Not secure | ec2-3-143-221-56.us-east-2.compute.amazonaws.com:8080
Predict Salary Analysis
Experience
Test Score
Interview Score
Predict