# 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

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.
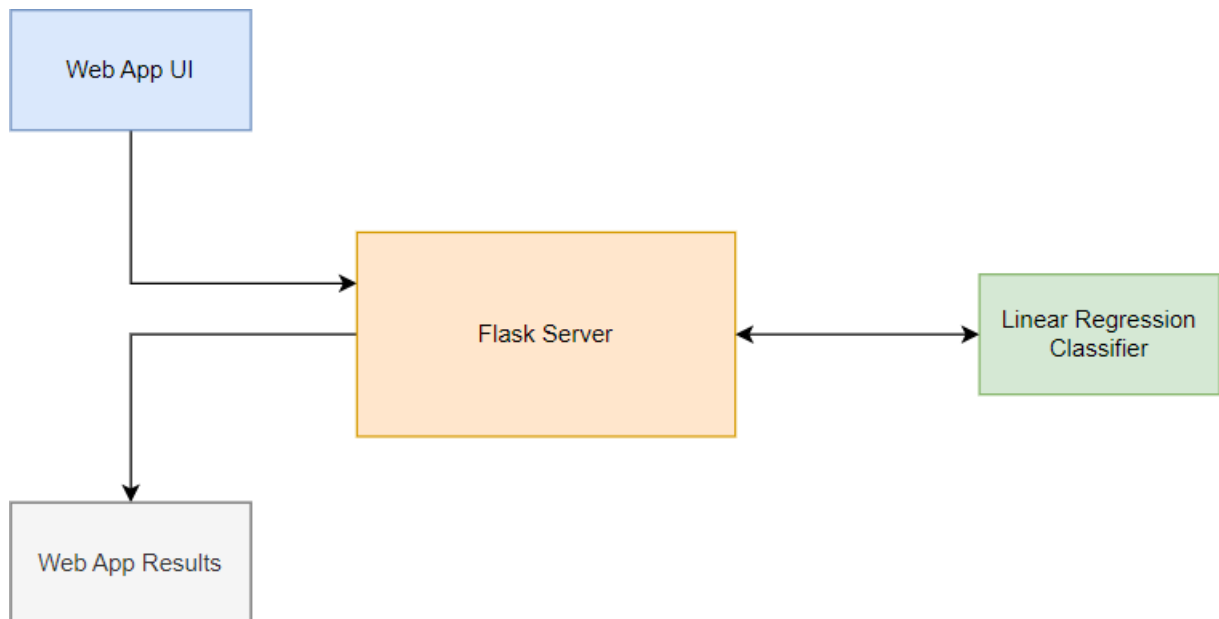


Figure 1.1: Application Workflow

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'.

```python
# 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.

```
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.

```
#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

```
# 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]]))
```

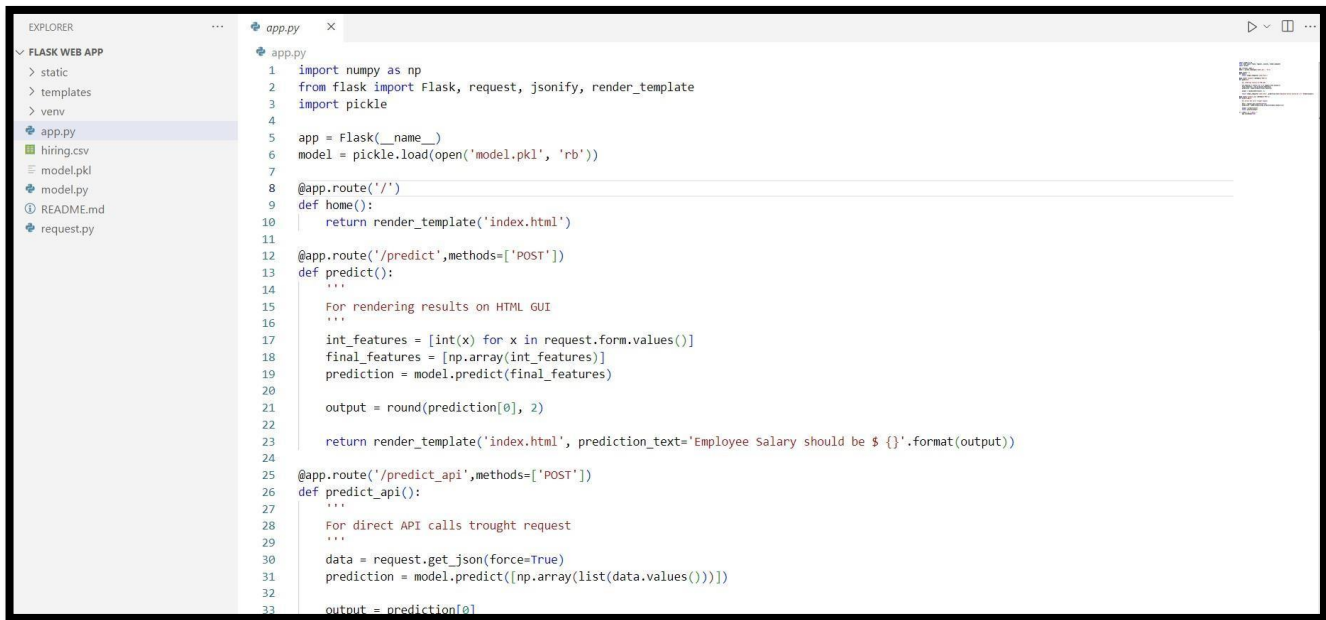## 4. Turning Model into Web Application

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.
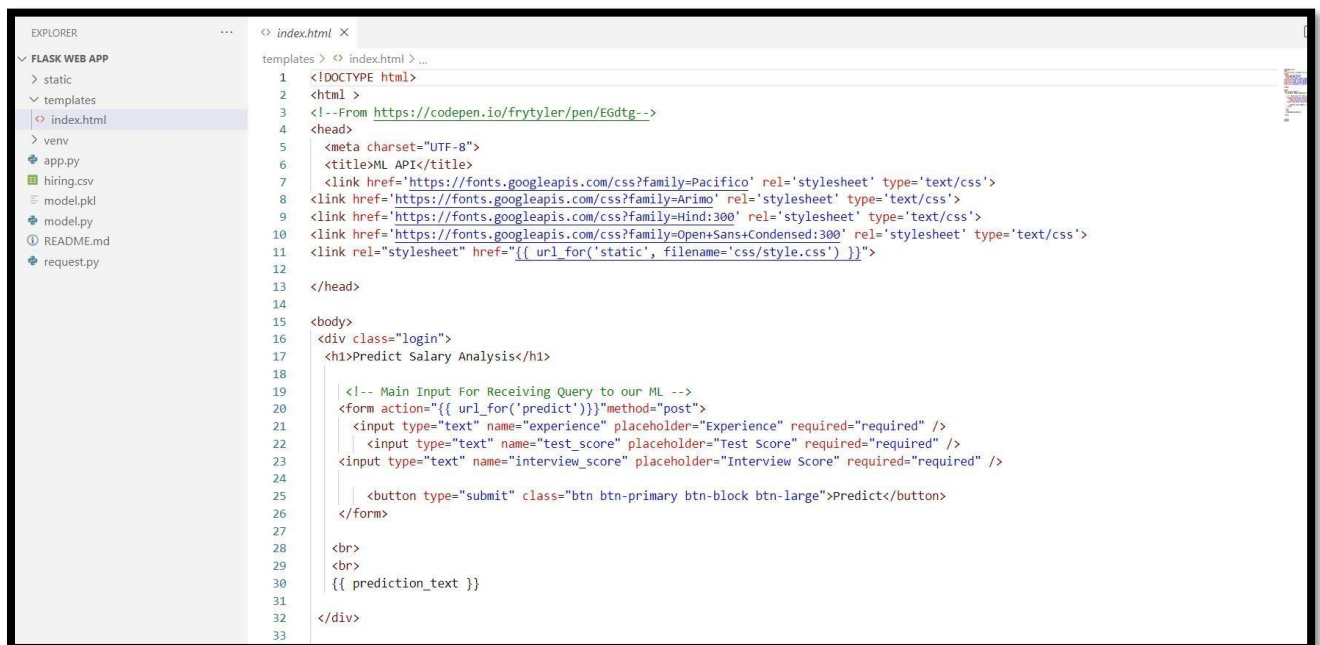
The model is loaded from a saved file, and two routes are defined to handle HTTP requests. The '/predict' route accepts data submitted through an HTML form and returns a predicted salary value. The '/predict_api' route accepts data submitted in JSON format and returns a salary prediction directly. The 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.

```
EXPLORER                  ···    ✿ app.py  ✕
∨ FLASK WEB APP                   ✿ app.py
  > static                    1    import numpy as np
  > templates                 2    from flask import Flask, request, jsonify, render_template
  > venv                      3    import pickle
  ✿ app.py                    4
  ▣ hiring.csv                5    app = Flask(__name__)
  ≡ model.pkl                 6    model = pickle.load(open('model.pkl', 'rb'))
  ✿ model.py                  7
  ① README.md                 8    @app.route('/')
  ✿ request.py                9    def home():
                             10        return render_template('index.html')
                             11
                             12    @app.route('/predict',methods=['POST'])
                             13    def predict():
                             14        '''
                             15        For rendering results on HTML GUI
                             16        '''
                             17        int_features = [int(x) for x in request.form.values()]
                             18        final_features = [np.array(int_features)]
                             19        prediction = model.predict(final_features)
                             20
                             21        output = round(prediction[0], 2)
                             22
                             23        return render_template('index.html', prediction_text='Employee Salary should be $ {}'.format(output))
                             24
                             25    @app.route('/predict_api',methods=['POST'])
                             26    def predict_api():
                             27        '''
                             28        For direct API calls trought request
                             29        '''
                             30        data = request.get_json(force=True)
                             31        prediction = model.predict([np.array(list(data.values()))])
                             32
                             33        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.
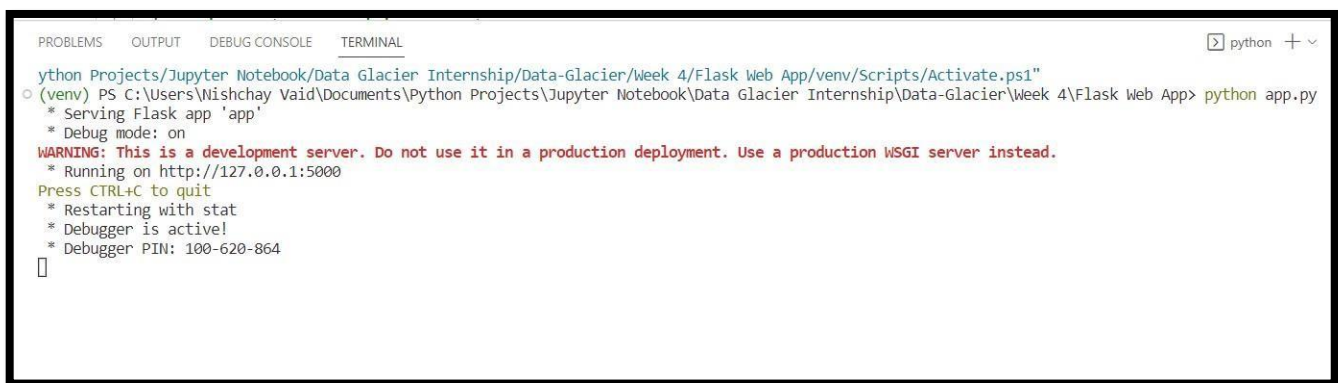
```
EXPLORER                  ···    <> index.html  ✕
∨ FLASK WEB APP                   templates > <> index.html > ...
  > static                    1    <!DOCTYPE html>
  ∨ templates                 2    <html >
    <> index.html             3    <!--From https://codepen.io/frytyler/pen/EGdtg-->
  > venv                      4    <head>
  ✿ app.py                    5      <meta charset="UTF-8">
  ▣ hiring.csv                6      <title>ML API</title>
  ≡ model.pkl                 7      <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>
  ✿ model.py                  8    <link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
  ① README.md                 9    <link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>
  ✿ request.py               10    <link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>
                             11    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
                             12
                             13    </head>
                             14
                             15    <body>
                             16    <div class="login">
                             17      <h1>Predict Salary Analysis</h1>
                             18
                             19        <!-- Main Input For Receiving Query to our ML -->
                             20        <form action="{{ url_for('predict')}}"method="post">
                             21          <input type="text" name="experience" placeholder="Experience" required="required" />
                             22          <input type="text" name="test_score" placeholder="Test Score" required="required" />
                             23        <input type="text" name="interview_score" placeholder="Interview Score" required="required" />
                             24
                             25          <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
                             26      </form>
                             27
                             28      <br>
                             29      <br>
                             30      {{ prediction_text }}
                             31
                             32    </div>
                             33
```

## 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.
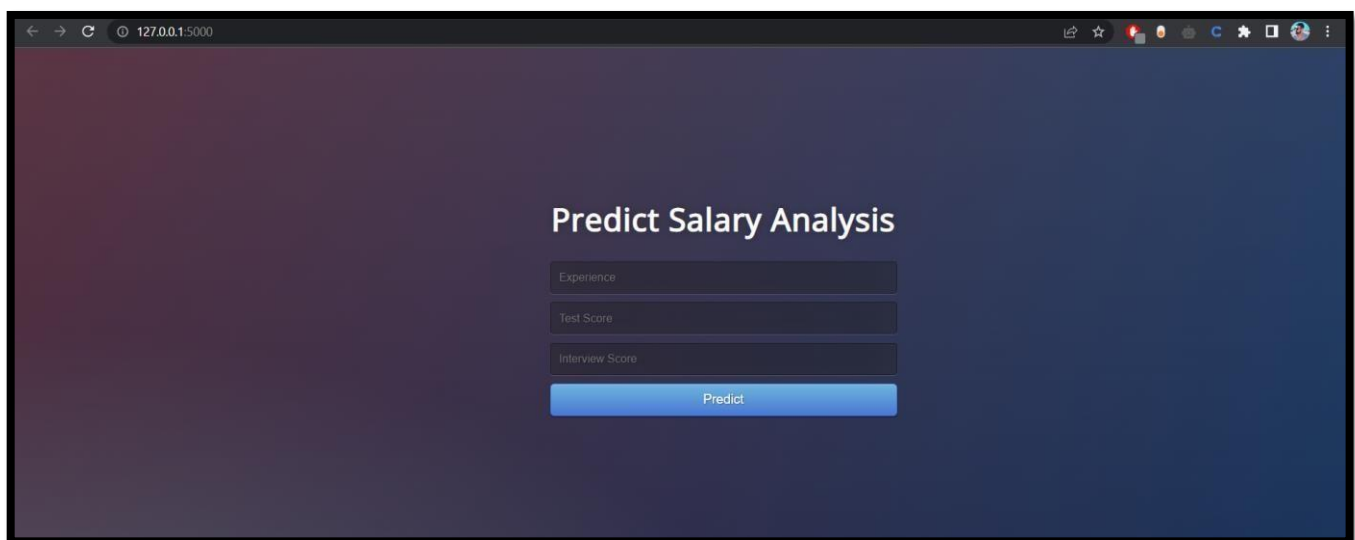
## 4.4 Running Procedure

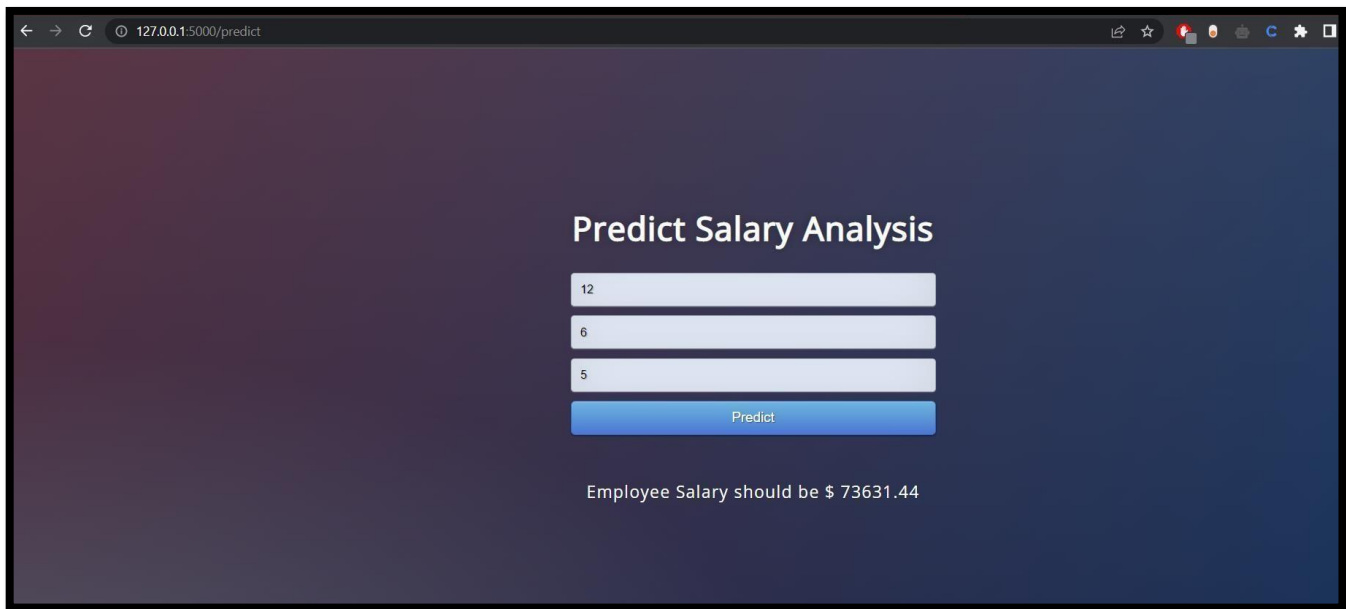Now, to start the Flask server we use the terminal and type in the command python app.py



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
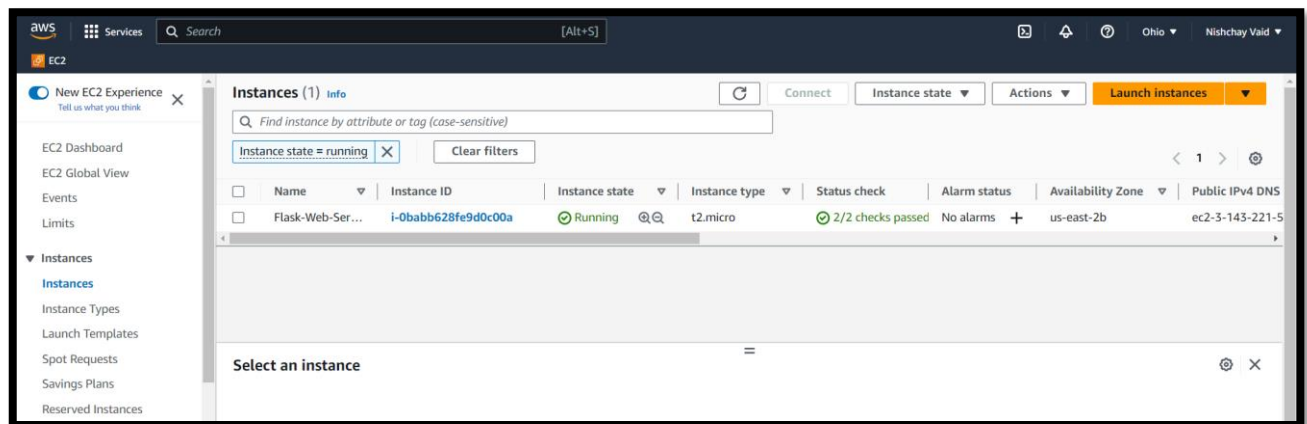


# 5. Deploying Model on AWS

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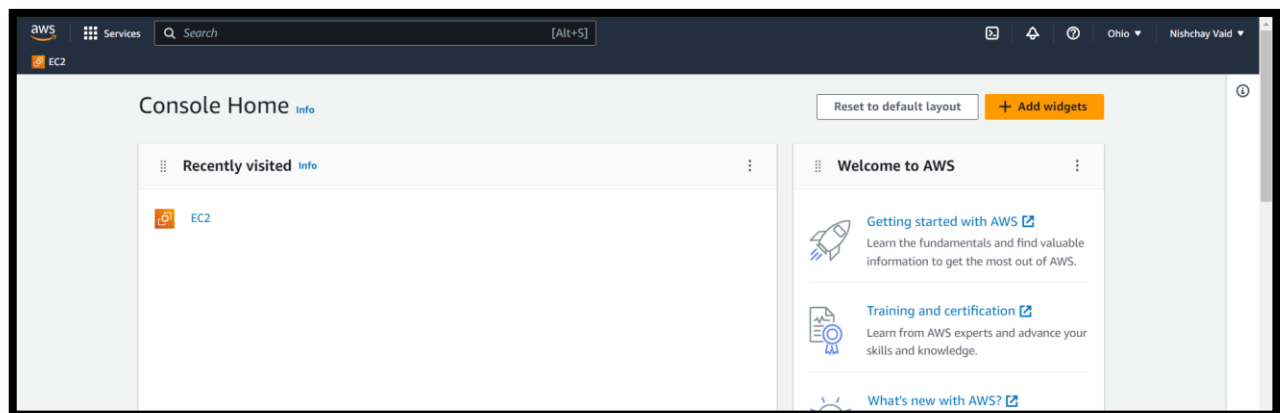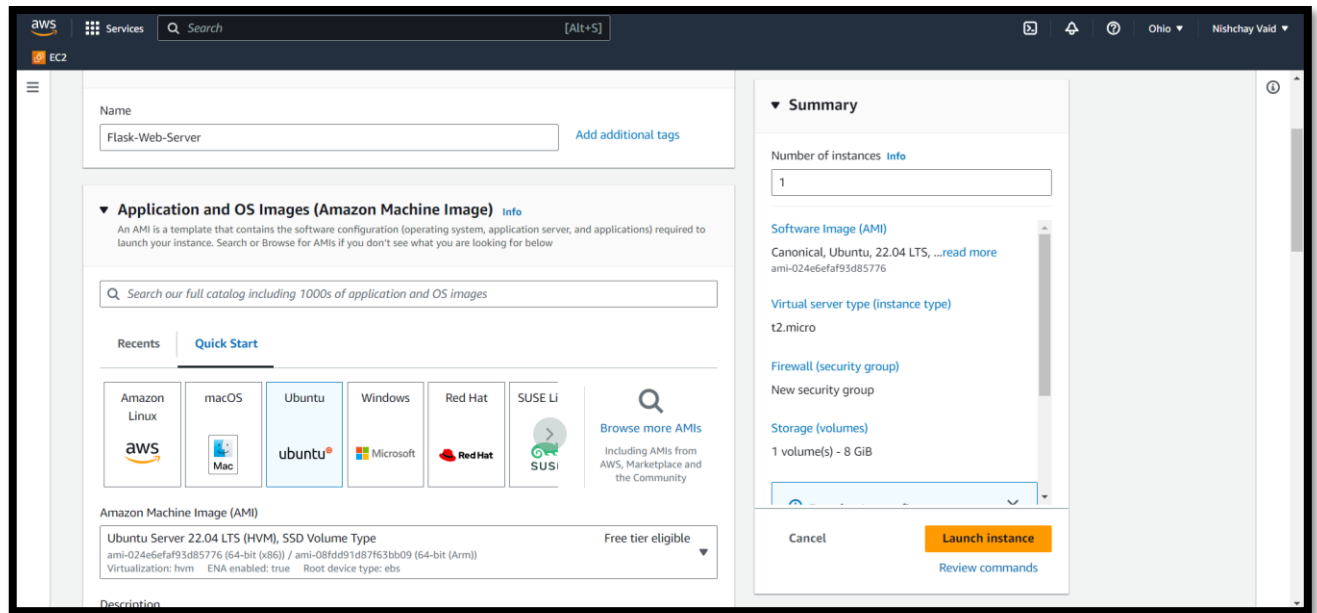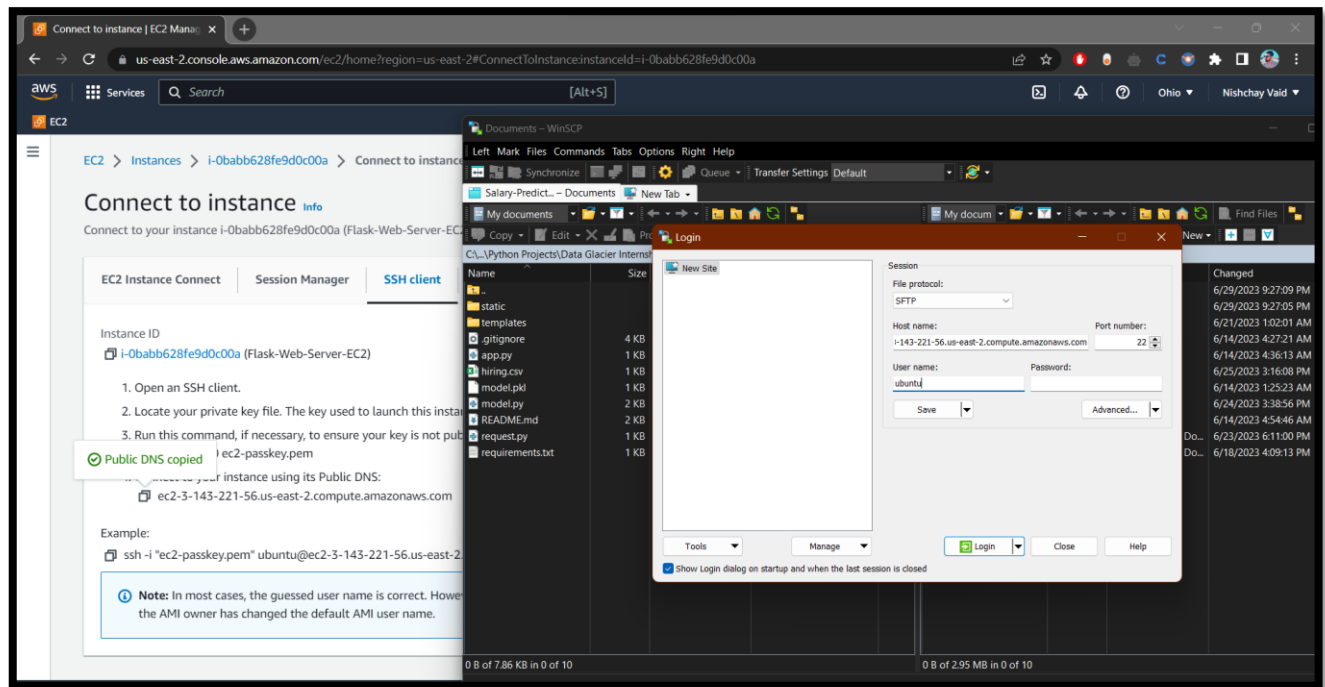
Once configured, the instance should be up and running as shown below:



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.



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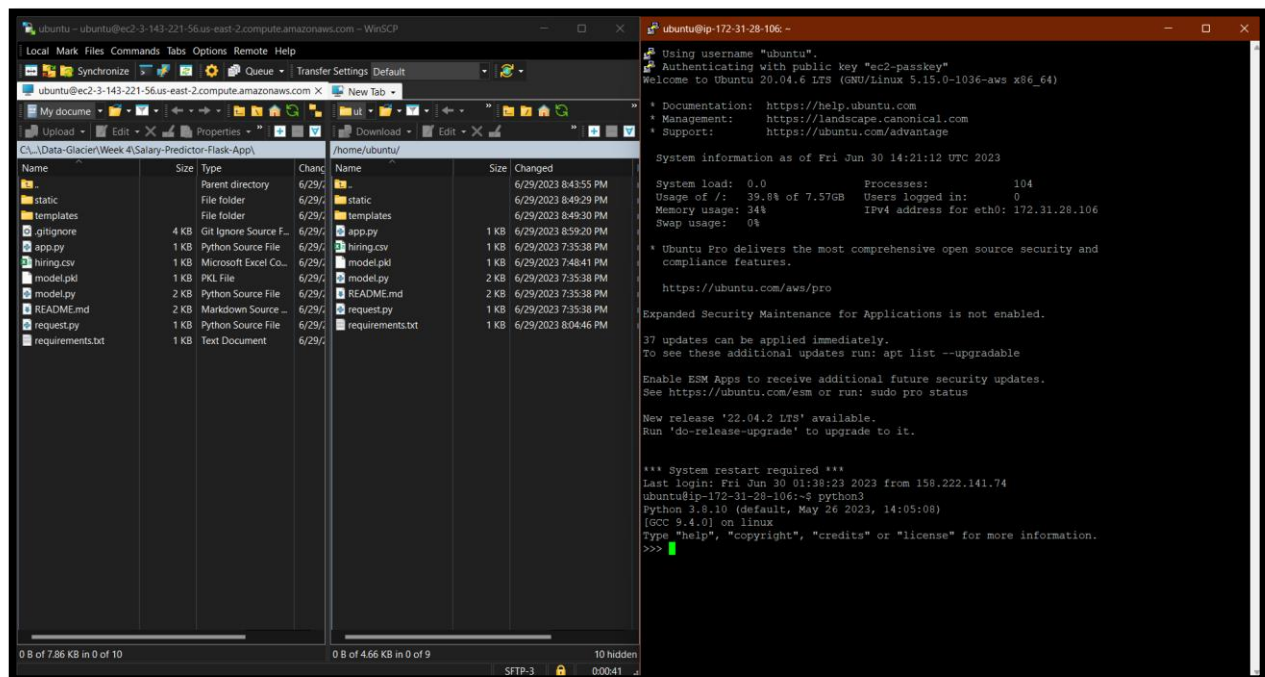
Now run the flask app using PuTTy terminal to turn on the flask server. Once it starts, our application is now live on the EC2 Instance.

## 5.2 Output