Data Science Internship

Week 4: Deployment of Flask

Name: Hira Fahim

Batch Code: LISUM19

Submission Date: 22nd March 2023

Submitted to: Data Glacier

Table of Contents

1. Introduction

An individual’s annual income results from several factors. Intuitively, it is influenced by the individual’s education level, age, gender, occupation, etc.

2. Agenda

The agenda is to build a Regression Model that is well trained to predict income based on attributes like gender, height, weight, occupation, education etc.

2.1. Dataset Information

The ‘dataset used’ is sourced from Kaggle. It comprises of 9 columns,131 rows, and 1179 total records. It has 5 discrete categorical columns and 4 continuous numeric columns. The Following Screenshot shows the data-frame of this dataset:

	Gender	Age	Height_cm	Weight_kg	Occupation	Education_Level	Marital_Status	Income_USD	Favorite_Color
0	male	32	175	70	Software Engineer	Master's Degree	Married	75000	Blue
1	male	25	182	85	Sales Representative	Bachelor's Degree	Single	45000	Green
2	female	41	160	62	Doctor	Doctorate Degree	Married	120000	Purple
3	male	38	178	79	Lawyer	Bachelor's Degree	Single	90000	Red
4	female	29	165	58	Graphic Designer	Associate's Degree	Single	35000	Yellow

Table 1 Dataset Information

2.2. Attribute Information

The collection is composed by the CSV file of dataset. Following table shows the attributes of the dataset:

Attributes	Details
Gender	Discrete categorical column
Age	Continuous numerical column
Height (cm)	Continuous numerical column
Weight (cm)	Continuous numerical column
Occupation	Discrete categorical column
education	Discrete categorical column
Marital status	Discrete categorical column
Income	Continuous numerical column
Favourite Colour	Discrete categorical column

Table 2 Attributes Information

2.3. Application workflow

Given Workflow shows Gradient Boosting Regressor model is used and Flask Framework for deployment. It represents the details of how the model works from user interface till the results.

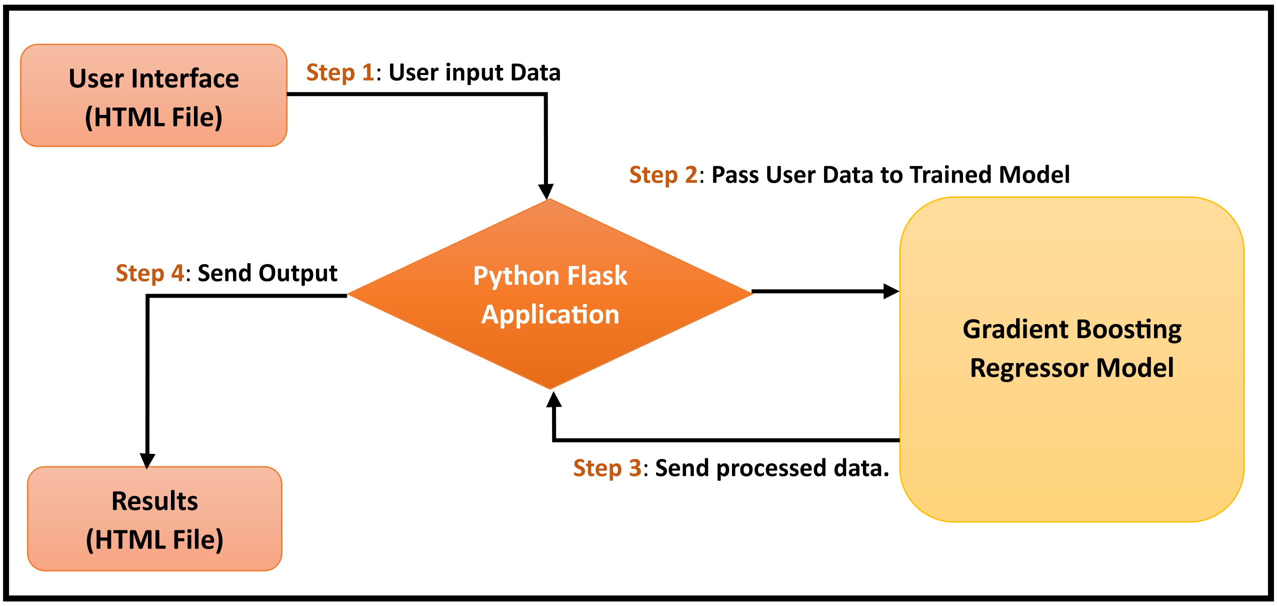


Fig 2.1 Application Framework

The machine learning model is built for Income Prediction based on input attributes, then creates an API for the model using flask Framework and python micro-framework for building web application. This API call used to predict results through HTTP requests.

3. Building Machine Learning Model

3.1. Import Libraries

Import essential libraries for model building (income_pred. jpynb and train.py)

```
In [1]: import pandas as pd                 # For data manupulation using dataframes
        import numpy as np                  # For Statistical Analysis
        import seaborn as sns               # for statistical Data Visualisation
        import matplotlib. pyplot as plt    # For Data Visualisation
```

3.2. Import Dataset

Import dataset for model training and building.

```
In [2]: d=pd.read_csv("gender.csv")
        d.head()      # display columns and 5 rows of dataset
```

Out[2]:

	Gender	Age	Height_cm	Weight_kg	Occupation	Education_Level	Marital_Status	Income_USD	Favorite_Color
0	male	32	175	70	Software Engineer	Master's Degree	Married	75000	Blue
1	male	25	182	85	Sales Representative	Bachelor's Degree	Single	45000	Green
2	female	41	160	62	Doctor	Doctorate Degree	Married	120000	Purple
3	male	38	178	79	Lawyer	Bachelor's Degree	Single	90000	Red
4	female	29	165	58	Graphic Designer	Associate's Degree	Single	35000	Yellow

3.3. Dataset Details

Below are the details of dataset like number of rows and columns, number of non-null values, columns' names, data types of each column and description of dataset.

```
In [3]: d.shape # shape of the dataset(no. of rows and columns)
Out[3]: (131, 9)

In [5]: d.info()  # Information about datset

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 131 entries, 0 to 130
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Gender           131 non-null    object
 1   Age              131 non-null    int64 
 2   Height_cm        131 non-null    int64 
 3   Weight_kg        131 non-null    int64 
 4   Occupation       131 non-null    object
 5   Education_Level  131 non-null    object
 6   Marital_Status   131 non-null    object
 7   Income_USD       131 non-null    int64 
 8   Favorite_Color   131 non-null    object
dtypes: int64(4), object(5)
memory usage: 9.3+ KB

In [6]: d.describe()  #Description about dataset
Out[6]:
```

	Age	Height_cm	Weight_kg	Income_USD
count	131.000000	131.000000	131.000000	131.000000
mean	34.564885	173.198473	71.458015	93206.106870
std	5.984723	8.045467	12.648052	74045.382919
min	24.000000	160.000000	50.000000	30000.000000
25%	29.000000	166.000000	60.000000	55000.000000
50%	34.000000	175.000000	75.000000	75000.000000
75%	39.000000	180.500000	83.000000	100000.000000
max	52.000000	190.000000	94.000000	500000.000000

3.4. Dataset Pre-processing

There are no null values in the data set.

```
In [7]: d.isnull().sum() #Check total null values
Out[7]: Gender             0
        Age                0
        Height_cm          0
        Weight_kg          0
        Occupation         0
        Education_Level    0
        Marital_Status     0
        Income_USD         0
        Favorite_Color     0
        dtype: int64
```

Using iloc function we drop the unnecessary column like Favourite_Color. Also, remove duplicate rows and rows with all null value. Final dataset has 128 rows.

```
In [8]: #Drop Unnecessary columns
        d=d.iloc[:,:-1]
        d
```

Out[8]:

	Gender	Age	Height_cm	Weight_kg	Occupation	Education_Level	Marital_Status	Income_USD
0	male	32	175	70	Software Engineer	Master's Degree	Married	75000
1	male	25	182	85	Sales Representative	Bachelor's Degree	Single	45000
2	female	41	160	62	Doctor	Doctorate Degree	Married	120000
3	male	38	178	79	Lawyer	Bachelor's Degree	Single	90000
4	female	29	165	58	Graphic Designer	Associate's Degree	Single	35000
...	...	...	...	...	...	...	...	...
126	female	32	170	64	Nurse	Associate's Degree	Single	60000
127	male	38	176	79	Project Manager	Bachelor's Degree	Married	90000
128	female	27	162	55	Graphic Designer	Associate's Degree	Single	55000
129	male	33	175	77	Sales Representative	Bachelor's Degree	Married	80000
130	female	29	164	57	Software Developer	Bachelor's Degree	Single	65000

131 rows × 8 columns

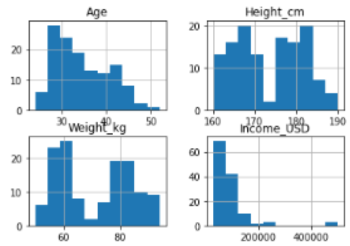
```
In [9]: # Drop duplicate rows and row with null values
        d=d.dropna(how="all")
        d=d.drop_duplicates()
        d.shape

Out[9]: (128, 8)
```

Here problem-values are treated and defined as NaN. Also, changed the datatype of categorical columns from 'object' to category.

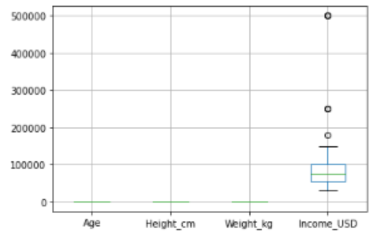
```
In [10]: #invalid parsing will be set as NaN
         d['Age']=pd.to_numeric(d['Age'],errors='coerce')
         d['Height_cm']=pd.to_numeric(d['Height_cm'],errors='coerce')
         d['Weight_kg']=pd.to_numeric(d['Weight_kg'],errors='coerce')
         d['Income_USD']=pd.to_numeric(d['Income_USD'],errors='coerce')
         #Datatype of categorical columns define as category
         d['Gender']=d['Gender'].astype('category')
         d['Occupation']=d['Occupation'].astype('category')
         d['Education_Level']=d['Education_Level'].astype('category')
         d['Marital_Status']=d['Marital_Status'].astype('category')

In [11]: d.info()  # Dataset information after data cleaning and feature selection

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 128 entries, 0 to 130
         Data columns (total 8 columns):
          #   Column           Non-Null Count  Dtype
         ---  ------           --------------  -----
          0   Gender           128 non-null    category
          1   Age              128 non-null    int64
          2   Height_cm        128 non-null    int64
          3   Weight_kg        128 non-null    int64
          4   Occupation       128 non-null    category
          5   Education_Level  128 non-null    category
          6   Marital_Status   128 non-null    category
          7   Income_USD       128 non-null    int64
         dtypes: category(4), int64(4)
         memory usage: 6.7 KB
```

Visualise the continuous columns to check variation and outliers in the dataset. These outliers show the natural variation, and they should be left in the dataset.

```
In [12]: d.hist()
Out[12]: array([[<AxesSubplot:title={'center':'Age'}>,
                 <AxesSubplot:title={'center':'Height_cm'}>],
                [<AxesSubplot:title={'center':'Weight_kg'}>,
                 <AxesSubplot:title={'center':'Income_USD'}>]], dtype=object)
```




```
In [13]: d.boxplot()
Out[13]: <AxesSubplot:>
```

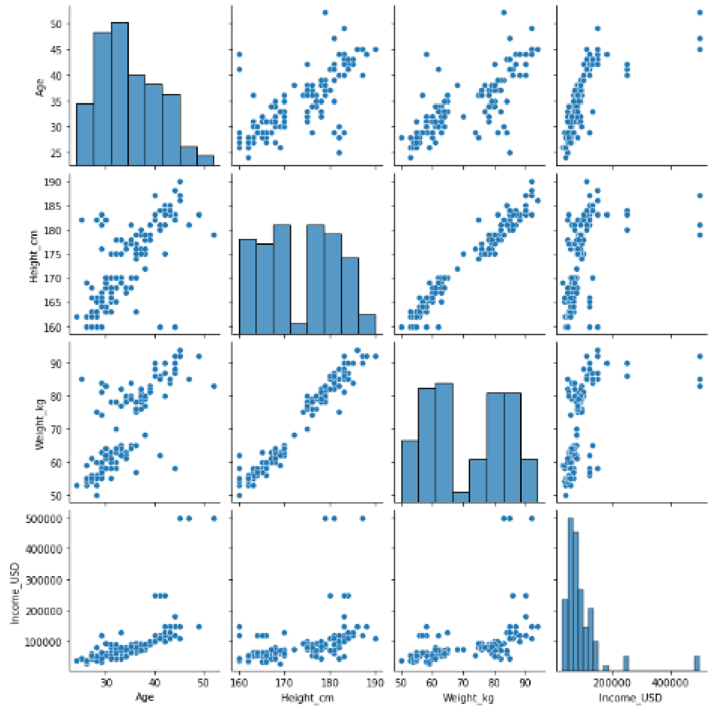


Below are the lists of categories in each discrete column along with their value counts.

```
In [15]: d['Gender'].value_counts() # Numbers of records of each category of discrete column
Out[15]: male      65
         female    63
         Name: Gender, dtype: int64

In [16]: d['Occupation'].value_counts()
Out[16]: Teacher                 12
         Writer                   9
         Lawyer                   9
         Nurse                    9
         Graphic Designer         9
         Marketing Specialist     9
         Project Manager          8
         Doctor                   8
         Engineer                 8
         Sales Representative     7
         IT Manager               7
         Architect                7
         Analyst                  6
         CEO                      6
         Accountant               6
         Business Analyst         5
         Business Consultant      1
         Software Developer       1
         Software Engineer        1
         Name: Occupation, dtype: int64

In [17]: d['Education_Level'].value_counts()
Out[17]: Bachelor's Degree     77
         Master's Degree       19
         Associate's Degree    18
         Doctorate Degree      14
         Name: Education_Level, dtype: int64

In [18]: d['Marital_Status'].value_counts()
Out[18]: Married     62
         Single      59
         Divorced     5
         Widowed      2
         Name: Marital_Status, dtype: int64
```

The pairplot and heatmap represent the correlation between continuous columns. Weight and height have linear relation.

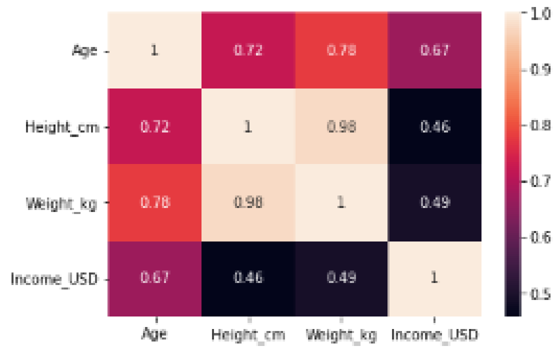
```
In [19]: sns.pairplot(d)    #visuals representation of Correlation between all continuous columns.
Out[19]: <seaborn.axisgrid.PairGrid at 0x18135ada7c0>
```




```
In [20]: d.corr()
Out[20]:
```

	Age	Height_cm	Weight_kg	Income_USD
Age	1.000000	0.720806	0.779204	0.667324
Height_cm	0.720806	1.000000	0.975158	0.457235
Weight_kg	0.779204	0.975158	1.000000	0.489154
Income_USD	0.667324	0.457235	0.489154	1.000000

```
In [21]: sns.heatmap(d.corr(),annot=True)
Out[21]: <AxesSubplot:>
```



For model building, convert all categorical columns to numerical columns.

In [119]:
```
# Encoding of discrete columns
from sklearn import preprocessing
le=preprocessing.LabelEncoder()
d['Gender']=le.fit_transform(d['Gender'])
d['Occupation']=le.fit_transform(d['Occupation'])
d['Education_Level']=le.fit_transform(d['Education_Level'])
d['Marital_Status']=le.fit_transform(d['Marital_Status'])
d.head()
```

Out[119]:

	Gender	Age	Height_cm	Weight_kg	Occupation	Education_Level	Marital_Status	Income_USD
0	1	32	175	70	16	3	1	75000
1	1	25	182	85	14	1	2	45000
2	0	41	160	62	6	2	1	120000
3	1	38	178	79	10	1	2	90000
4	0	29	165	58	8	0	2	35000

3.5. Model Building

Import train_test_split and divide the dataset into input variables and output variable then split the input and output into train and test sets (20% test and 80% train).

In [120]:
```
from sklearn.model_selection import train_test_split   #splitting dataset into train and test dataset
```

In [121]:
```
X=d.iloc[:,:-1] # Independent variables
Y=d.iloc[:,-1]  #Out variables
```

In [135]:
```
X_train,X_test,Y_train,Y_test =train_test_split(X,Y, test_size=0.2,random_state=40)
# Splitting dataset into 20% test dataset and 80% train dataset
```

In [136]:
```
X_train.shape
```

Out[136]: (102, 7)

In [137]:
```
X_test.shape
```

Out[137]: (26, 7)

After data pre-processing, a machine learning model is created to predict the Income. For this purpose, Gradient Boosting Regressor algorithm is used from ensemble. scikit-learn. After importing and initialize GradientBoosingRegressor model the dataset is being fitted for training using reg.

```
In [176]: from sklearn.ensemble import GradientBoostingRegressor #imnport XGboost regressor from sklearn

In [210]: params ={"n_estimators": 300,
                   "max_depth":4,
                   "min_samples_split":5,
                   "learning_rate": 0.01,
                   "loss":"squared_error",}
          reg=GradientBoostingRegressor(**params)
          reg.fit(X_train,Y_train)

Out[210]: GradientBoostingRegressor(learning_rate=0.01, max_depth=4, min_samples_split=5,
                                    n_estimators=300)

In [216]: y_pred=reg.predict(X_test)

In [211]: print ("Accuracy of All dataset: ", (reg.score(X,Y)))
          print ("Accuracy of Train dataset: " ,(reg.score(X_train,Y_train)))
          print ("Accuracy of Test dataset: " ,(reg.score(X_test,Y_test)))

          Accuracy of All dataset:  0.9582452703209905
          Accuracy of Train dataset:  0.9785637228842583
          Accuracy of Test dataset:  0.9025308667502843

In [220]: from sklearn import metrics

In [224]: print('MAE:', metrics.mean_absolute_error(Y_test,y_pred))
          print('MSE:', metrics.mean_squared_error(Y_test,y_pred))
          print('RMSE:', np.sqrt(metrics.mean_squared_error(Y_test, y_pred)))

          MAE: 13056.13447918422
          MSE: 713399079.1315732
          RMSE: 26709.531615727992
```

3.6. Save the Model

Last step is saving the model using pickle.

```
In [212]: #model.py
          import pickle # its used for seriealizing.
          pickle.dump(reg, open('model.pkl','wb'))
          model = pickle.load(open('model.pkl','rb'))
```

4. Deployment of model into flask framework

A web application is develop that consists of a two-web pages, one with a form field that lets us enter input values. After submitting the input to the web application, it will redirect it on a result page which gives us the predicted income. Following is the directory structure of all files used for application.

4.1. App.py

The app.py file contains the source code including the ML code for prediction and will be execute by the Python interpreter to run the Flask web application.

```
import numpy as np
from flask import Flask, request, render_template
from sklearn.metrics import DistanceMetric
import pickle

#Create an app object using the Flask class.
app = Flask(__name__)

#Load the trained model. (Pickle file)
model = pickle.load(open('models/model.pkl', 'rb'))

#Define the route to be home.
#The decorator below links the relative route of the URL to the function it is decorating.
#Here, home function is with '/', our root directory.
#Running the app sends us to index.html.
#Note that render_template means it looks for the file in the templates folder.

#use the route() decorator to tell Flask what URL should trigger our function.
@app.route('/')
def home():
  return render_template('index.html')

#You can use the methods argument of the route() decorator to handle different HTTP methods.
#GET: A GET message is send, and the server returns data
#POST: Used to send HTML form data to the server.
#Add Post method to the decorator to allow for form submission.
#Redirect to /predict page with the output
@app.route('/predict',methods=['POST'])
def predict():

    int_features = [int(x) for x in request.form.values()] #Convert string inputs to float.
    features = [np.array(int_features)]  #Convert to the form [[a, b,c]] for input to the model
    prediction = model.predict(features)  # features Must be in the form [[a, b,c]]

    output = round(prediction[0], 2)

    return render_template('results.html', prediction_text="Income : $ {} ".format(output))


#When the Python interpreter reads a source file, it first defines a few special variables.
#For now, we care about the __name__ variable.
#If we execute our code in the main program, like in our case here, it assigns
# __main__ as the name (__name__).
#So if we want to run our code right here, we can check if __name__ == __main__
#if so, execute it here.
#If we import this file (module) to another file then __name__ == app (which is the name of this python file).

if __name__ == "__main__":
    app.run()
```

▪ Application will run as a single module; thus, a new Flask instance is initialized 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.

▪ Next, the route decorator is used (@app. route ('/')) to specify the URL that should trigger the execution of the home function. Home function simply rendered the index.html HTML file, which is in the templates folder.

▪ Predict function has the data set, it pre-processes the input, and make predictions, and then store the model. The input is entered by the user and uses the 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.

▪ The run function is used to only run the application on the server when this script is directly executed by the Python interpreter, which we ensured using the if statement with __name__ == '__main__'.

4.2. Index.html

The Index.html file will render a text form where a user enter the details of required fields.

```html
<!DOCTYPE html>
<html >
<head>
  <meta charset="UTF-8">

  <!-- Make it compatible to mobile devices -->
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>INCOME PREDICTION</title>
</head>

<body style="background: #FFA07A;">
 <div class="login">
    <h1 style="color:Black;">INCOME PREDICTION</h1>

     <!-- Action is where the data is sent. In our case, predict page.
     If action is omitted, it assumed to be the current page -->
    <form action="{{ url_for('predict')}}"method="post">
        <label for="Gender">Gender:</label>
        <input type="text" name="Gender" placeholder="1= Male or 0= Female" required="required" /><p><br>
        <label for="Age">Age:</label>
        <input type="text" name="Age" placeholder="Enter the Age" required="required" /><p><br>
        <label for="Height">Height:</label>
        <input type="text" name="Height" placeholder="Height in cm" required="required" /><p><br>
        <label for="Weight">Weight:</label>
        <input type="text" name="Weight" placeholder="Weight in kg" required="required" /><p><br>
        <label for="Occupation">Occupation:</label>
        <input type="text" name="Occupation" placeholder="Select option from 0 to 18" required="required" /><p><br>
        <label for="Education">Education:</label>
        <input type="text" name="Education" placeholder="Select option from 0 to 3" required="required" /><p><br>
        <label for="Marital status">Marital Status:</label>
        <input type="text" name="Marital Status" placeholder="Select option from 0 to 3" required="required" /><p><br>

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

<h8 style="color:white;">List of Occupations:</h8><br>
<h8 style="color:white;">0:CEO, 1:Accountant, 2:IT Manager, 3:Business Analyst,
4:Software Developer, 5:Analyst, 6:Doctor,</h8><br>
<h8 style="color:white;">7:Project Manager, 8:Nurse, 9:Architect,
10:Lawyer, 11:Graphic Designer, 12:Marketing Specialist,</h8> <br>
<h8 style="color:white;">13:Engineer, 14:Sales Representative,
15:Software Engineer, 16:Business Consultant, 17:Teacher, 18:Writer </h8><br></br>
<h8 style="color:white;">Level of Education:</h8>
<h8 style="color:white;">0:Associates, 1:Bachelors, 2:Doctorate, 3:Masters</h8><br></br>
<h8 style="color:white;">Level of Education:</h8>
<h8 style="color:white;">0:Divorced, 1:Married, 2:Single, 3:Window</h8><br>
 </div>

</body>
</html>
```

4.3. Results.html

Result.html file will be rendered via the render_template ('results.html', prediction_text="Income: $ {} ".format(output)), which is inside the predict function of app.py script to display the output as per the input submitted by the user.

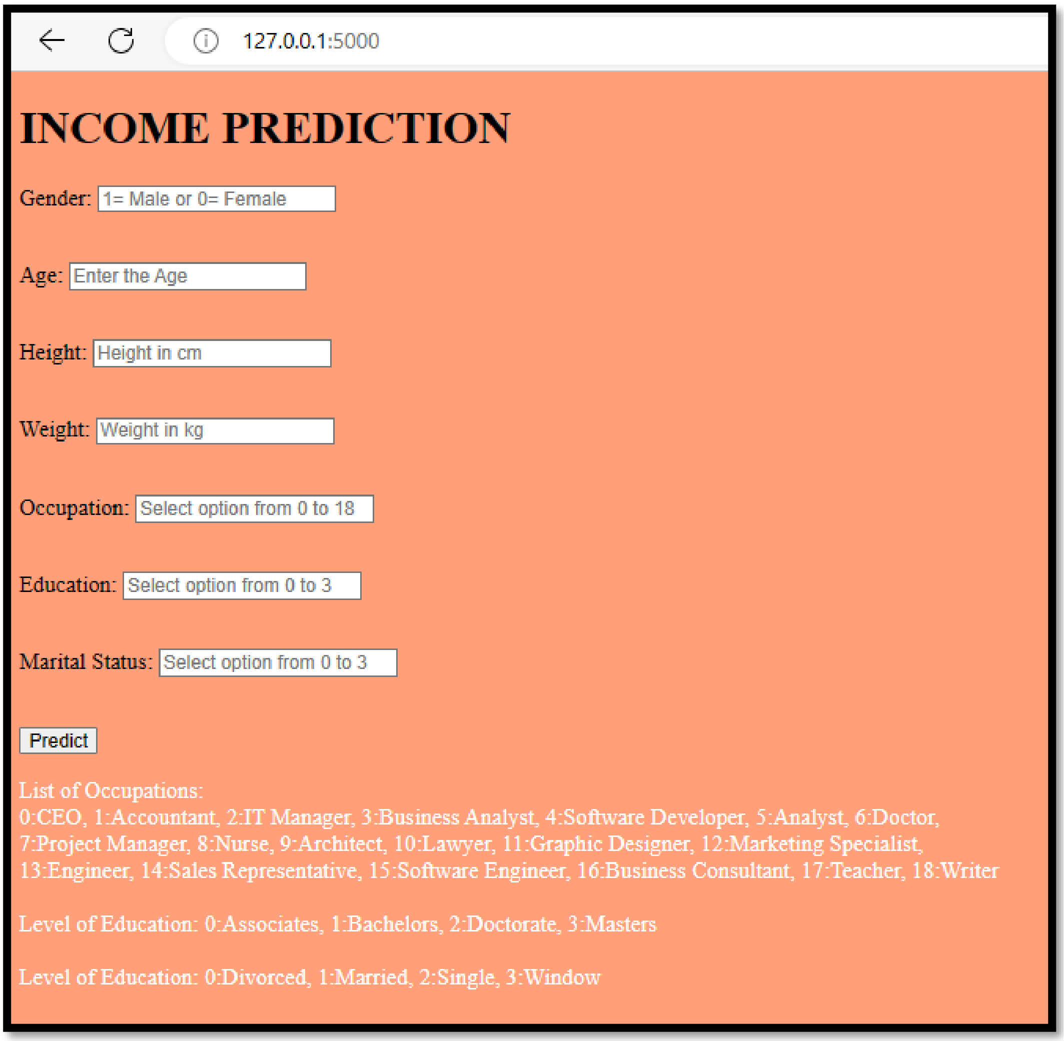
```html
<!DOCTYPE html>
<html >
<head>
  <meta charset="UTF-8">

  <!-- Make it compatible to mobile devices -->
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>INCOME PREDICTION</title>

</head>

<body style="background: #FFA07A;">
 <div class="login">
    <h1 style="color:Black;">INCOME PREDICTION</h1><br></br>
    <h4 style="color:Black;">{{prediction_text }}</h4>
```

4.4. Development Server

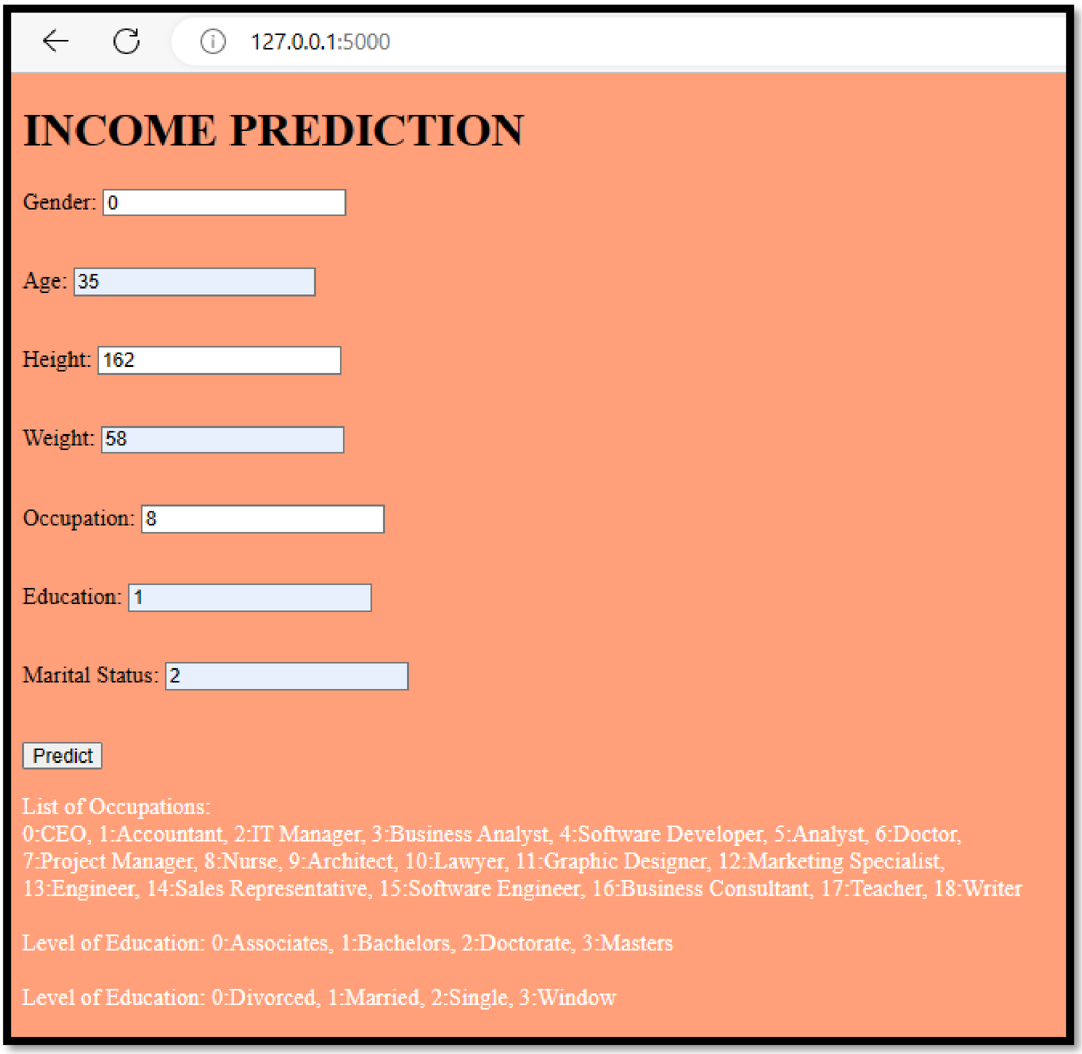
Following is the URL generate by 'app.py.'

```
In [1]: runfile('C:/Users/hiras/Documents/DataScienceDG/Data Glacier/
week4/app.py', wdir='C:/Users/hiras/Documents/DataScienceDG/Data Glacier/
week4')
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production
deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Now open a web browser and navigate to http://127.0.0.1:5000/ following is output of Index.html.



Fill in the required fields (Select categorical fields as per their respective number code given below) and click the Predict button.



The following Output is displayed via Results.html