Name: UJUNWA FATIMA
Batch code: LISUM18

Submission date: 18 FEBRUARY, 2023
Submitted to: DATA GLACIER

INTRODUCTION

In this project, we are concerned about the deployment of a linear regression model which predicts the employee salary based on experience, test score and interview score. This will be accomplished with *Flask* - a micro web framework written in Python to allow for the building of web applications. We will first build the model and then deploy it to *flask*.

MODEL BUILDING

DATA INFORMATION

Our data set is an employee data set with 8 rows and 3 columns. The features include experience, *test_score* and *interview_score* while the target variable is the *salary*. Based on the features, we are required to predict the salary of the individual. This is depicted in the table shown below.

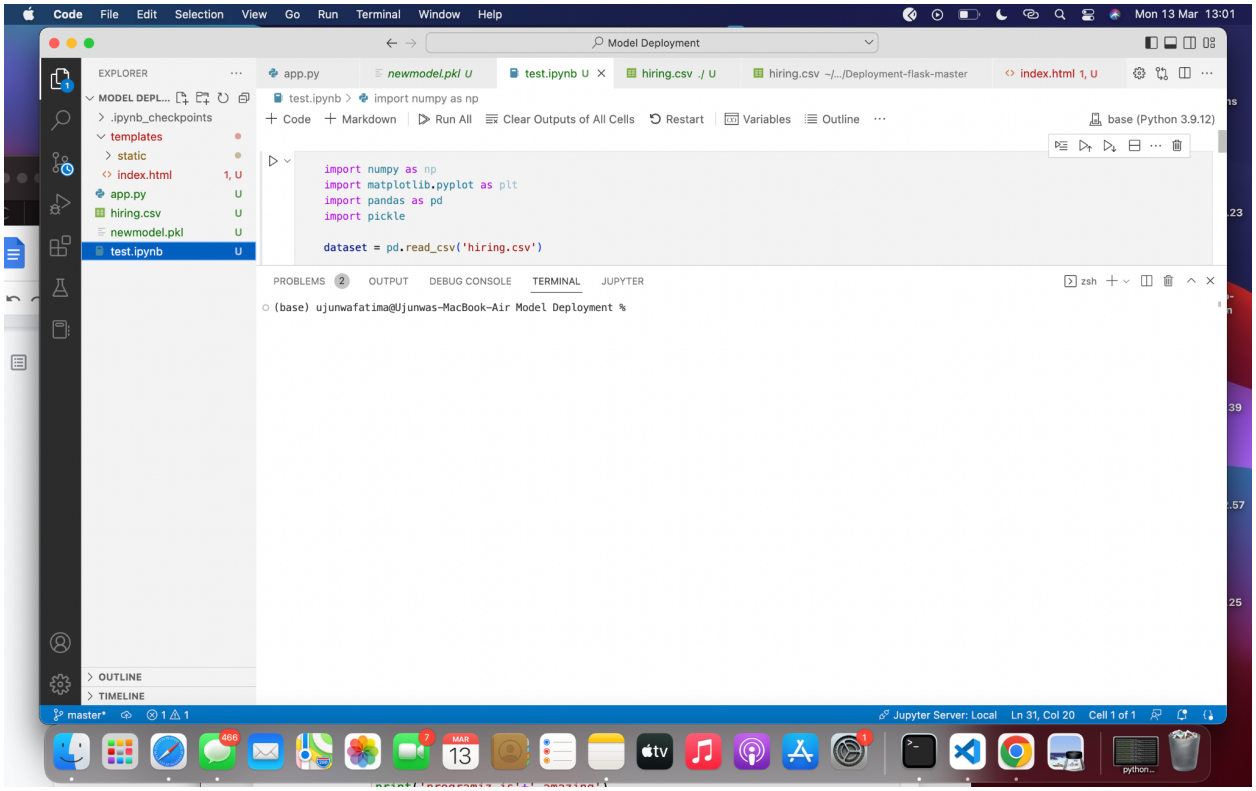
	Experience	test_score	interview_score	salary
1	NAN	8	9	50000
2	NAN	8	6	45000
3	five	6	7	60000
4	two	10	10	65000
5	seven	9	6	70000
6	three	7	10	62000

7	**ten**	**NAN**	**7**	**72000**
8	**eleven**	**7**	**8**	**80000**

Table 2.1: Dataset Information

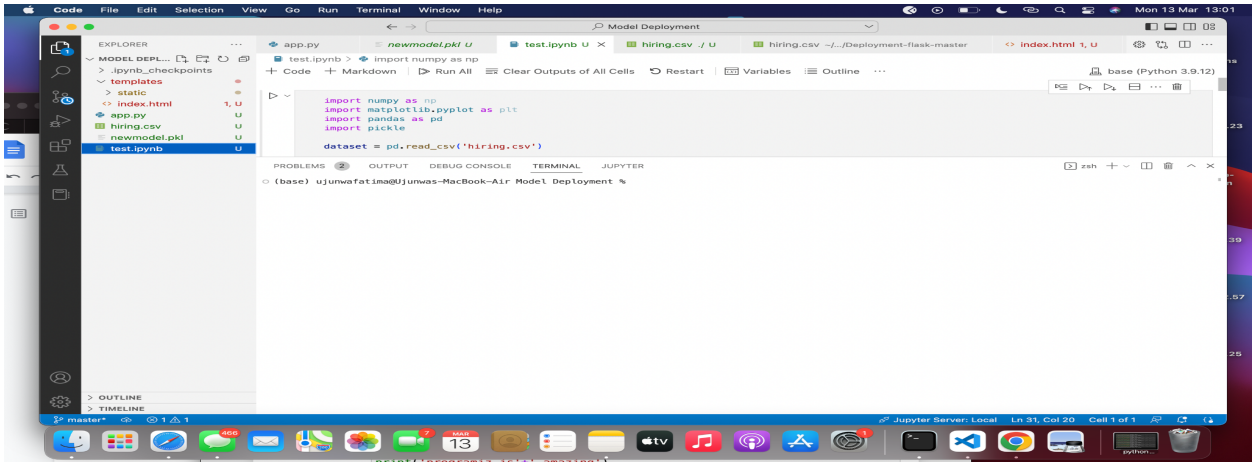
IMPORTING LIBRARIES

Here we imported the necessary libraries to aid in our model building as shown in the figure below.



LOADING THE DATA

The dataset is in comma separated value format (CSV) so we used the pandas library to read and convert it into a dataframe.



DATA PREPROCESSING

As we saw in table 2.1, we had a lot of missing values as well as all data points written in words for the *experience* feature. We would be required to convert the data points to int data type and also fill in the missing data to proceed.

```
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]
```

BUILDING THE MODEL

To build the model did not split the data into test and training dataset because of how small our dataset was. We employed the linear regression model to fit our data.

```
#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 import LinearRegression
regressor = LinearRegression()

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

SAVING THE MODEL TO DISK

We finally saved our model, as 'newmodel' using *Pickle*. *Pickle* helps us deploy the model to the Flask framework easily.

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

DEPLOYING MODEL TO FLASK

To achieve this we created a folder *Model Deployment* with the following file arrangement as shown below.

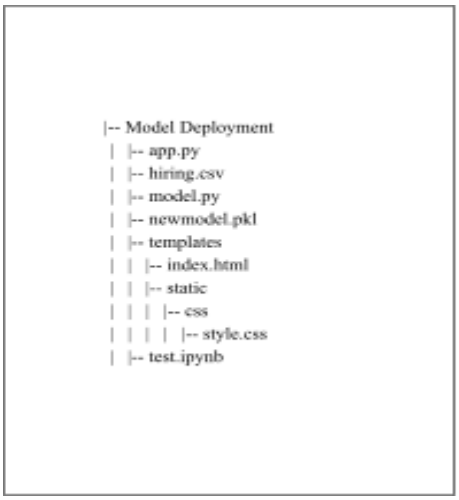
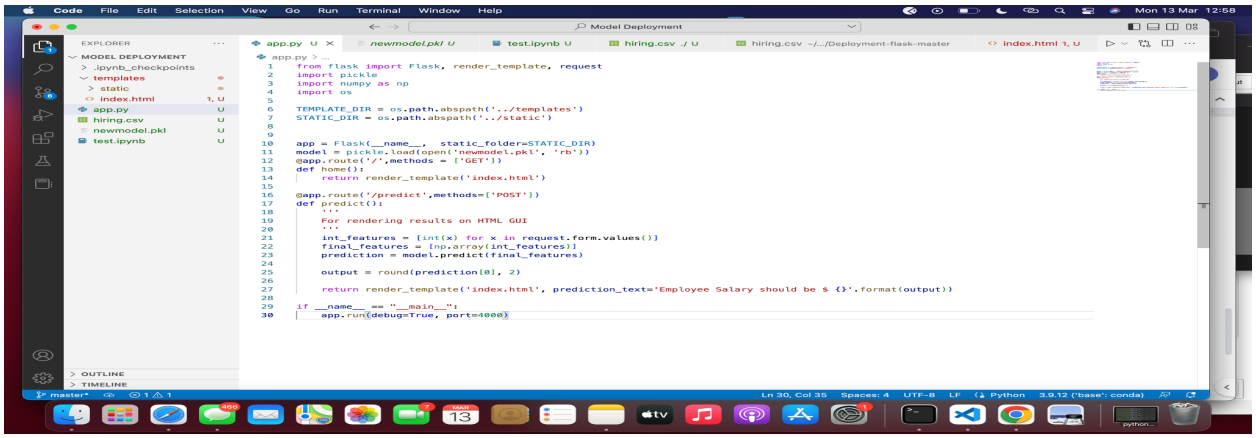


Figure 1.2: Folder Structure

FOLDER STRUCTURE

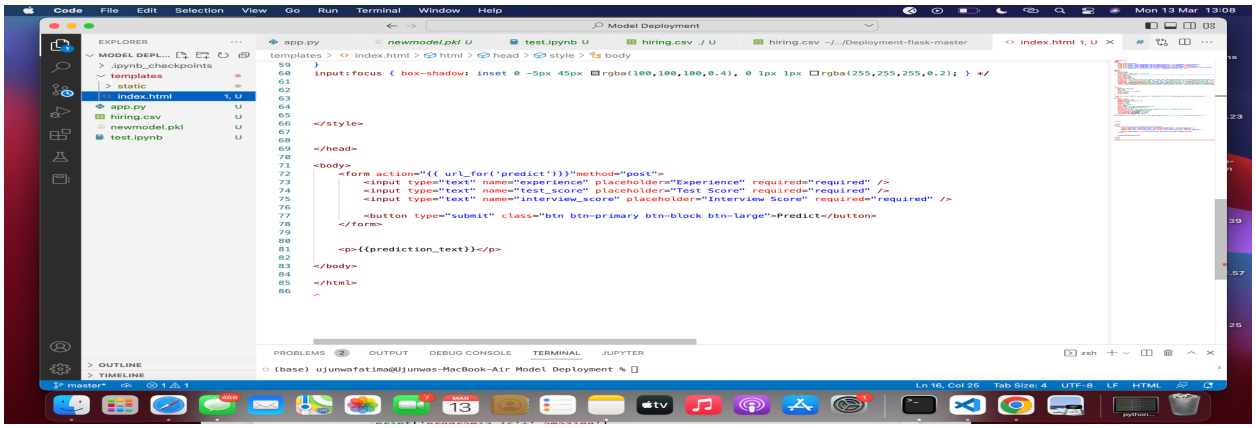
App.py: This file is to be usually executed by a Python interpreter. We load our save model *newmodel.pkl* and import Flask on this file.

- *@app.route* is used to indicate the endpoint and the method used to execute the function.
- *def predict* is written to save the inputs to variable *int_features,* transform the inputs to array and then finally use the inputs to predict the salary. The return value is then rendered on the html file, index.html



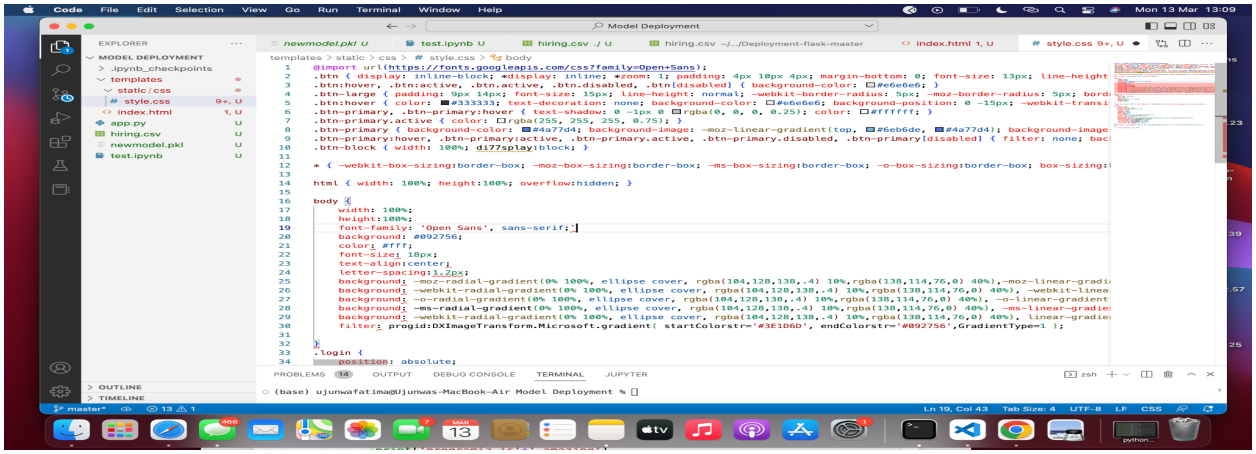
```
from flask import Flask, render_template, request
import pickle
import numpy as np
import os

TEMPLATE_DIR = os.path.abspath('../templates')
STATIC_DIR = os.path.abspath('../static')

app = Flask(__name__,  static_folder=STATIC_DIR)
model = pickle.load(open('newmodel.pkl', 'rb'))
@app.route('/',methods = ['GET'])
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))

if __name__ == "__main__":
    app.run(debug=True, port=4000)
```

Index.html: This file contains the html code that displays the structure of the input fields, buttons and text used on the website. The result of the prediction is also rendered on the web app through this file. We save this file in a folder *templates.*



```
input:focus { box-shadow: inset 0 -5px 45px rgba(100,100,100,0.4), 0 1px 1px rgba(255,255,255,0.2); } */



</style>

</head>

<body>
    <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>

    <p>{{prediction_text}}</p>
</body>

</html>
```

Style.css: This contains the css code applied to give the web application styling. This styling method employed here is the external styling which allows the styling to be done in a single file and linked to the html file. The *.css* file is usually saved in a folder which is in turn saved in a folder named *static.* Flask automatically sources out the styling by looking for this folder.



RESULTS

Once all the code is written as shown in the figures above, we run the app.py file and load the server. The resulting web page is as shown below: