

**Data Science Internship**

**Deployment on Flask**

**Name: Taimoor Razi**

**Batch Code: LISUM11: 30**

**Submitted to: Data Glacier**

**Submission Date: 24<sup>th</sup> July, 2022**

# Table of Contents

Step 1: Generating Dataset.....	3
Step 2: Training & Saving Model.....	4
Step 3: Create web app using flask (integrating HTML) .....	5
Step 4: Create Procfile and requirement.txt.....	7
Step 5: Upload files on Github repository .....	8
Step 6: Creating and Connecting Heroku app to Github repository.....	9
Step 7: Deploying app on Heroku .....	10

## Step 1: Generating Dataset

For this assignment, I created my own simple dataset as an excel file with four columns and eight entries. The target variable is the people salaries while the independent variables/training features are “experience”, “test\_score” and “interview\_score” of an individual. The dataset is shown in the following figure.

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

*Figure 1: Dataframe*

## Step 2: Training & Saving Model

A simple python file containing the model is made as model.py.

Firstly, some feature engineering is done. The missing values in the experience column are assumed to be for fresh graduate with zero experience and hence are replaced with integer 0. The missing value for the test\_score column is replaced by the average of the column feature. Finally, the categorical values in the experience column are replaced with integer values by creating a function to handle it. Since the dataset is very small, the whole data is trained on a linear regression model. Finally the model is saved as a pickle file. The figure below shows all the steps.

```
model.py > ...
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import pandas as pd
4  import pickle
5
6  df = pd.read_excel("hiring_data.xlsx")
7
8  df["experience"].fillna(0,inplace=True)
9
10 df["test_score"].fillna(df["test_score"].mean(), inplace=True)
11
12 X= df.iloc[:, :3]
13 y=df.iloc[:, -1]
14
15 #Converting words to integer values
16 def convert_to_int(word):
17     word_dict = {"one":1, "two":2, "three":3, "four":4, "five":5, "six":6, "seven":7,
18                 "eight":8, "nine":9, "ten":10, "eleven":11, "twelve":12, "zero":0, 0:0}
19     return word_dict[word]
20
21 X["experience"] = X["experience"].apply(lambda x : convert_to_int(x))
22
23 # splitting training and testing set
24 # Since we have a very small dataset, we will train our model with all available data.
25
26 from sklearn.linear_model import LinearRegression
27 regressor = LinearRegression()
28
29 # fitting model with training data
30 regressor.fit(X,y)
31
32 # saving the model to disk
33 pickle.dump(regressor, open("model.pkl", "wb"))
34
35 # loading the model to compare the results
36 model = pickle.load(open("model.pkl", "rb"))
37 print(model.predict([[2,9,6]]))
```

Figure 2: Code for model.py

### Step 3: Create web app using flask (integrating HTML)

A new python file called “app.py” is created.

After importing flask, we create our web app by initializing a flask object which act as a WSGI application.

Then a decorator is created (app.route) with the url “/” as the argument. A home function is created inside the decorator which returns html.index file as render\_template. This html.index file is created to give our web app some text and design on home display.

Then another decorator (app.route()) is created with url as “/predict” and methods=[“POST”] request as the second argument . This means that after we input the score the page will be redirected to this route. A predict function is defined inside app.route(). And this make prediction of salary for the input features. The result is rendered on HTML GUI as the function returns the html.index file as render template.

The code written and the app display after predictions is shown in the following figures.

```
app.py > ...
1  import numpy as np
2  from flask import Flask, request, jsonify, render_template
3  import pickle
4
5  app = Flask(__name__)
6  model = pickle.load(open("model.pkl", "rb"))
7
8
9  @app.route("/")
10 def home():
11     return render_template("index.html")
12
13
14 @app.route('/predict', methods=["POST"])
15 def predict():
16     """
17     For rendering results on HTML GUI
18     """
19     int_features = [int(x) for x in request.form.values()]
20     final_features = [np.array(int_features)]
21     prediction = model.predict(final_features)
22
23     output = round(prediction[0], 2)
24
25     return render_template("index.html", prediction_text="Employee Salary should be $ {}".format(output))
26
27
28 if __name__ == "__main__":
29     app.run(debug=True)
30
```

Figure 3: Code for app.py

```

templates > index.html > ...
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <meta charset="UTF-8">
6      <title>ML API</title>
7      <link href="https://fonts.googleapis.com/css?family=Pacifico" rel="stylesheet" type="text/css">
8      <link href="https://fonts.googleapis.com/css?family=Arimo" rel="stylesheet" type="text/css">
9      <link href="https://fonts.googleapis.com/css?family=Hind:300" rel="stylesheet" type="text/css">
10     <link href="https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300" rel="stylesheet" type="text/css">
11
12 </head>
13
14 <body>
15     <div class="login">
16         <h1>Predict Salary Analysis</h1>
17
18         <form action="{{ url_for('predict')}}" method="post">
19             <input type="text" name="experience" placeholder="Experience" required="required" />
20             <input type="text" name="test_score" placeholder="Test Score" required="required" />
21             <input type="text" name="interview_score" placeholder="Interview Score" required="required" />
22
23             <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
24         </form>
25
26         <br>
27         <br>
28         {{ prediction_text }}
29
30     </div>
31 </body>
32 </html>

```

Figure 4: Code for index.html

# Predict Salary Analysis

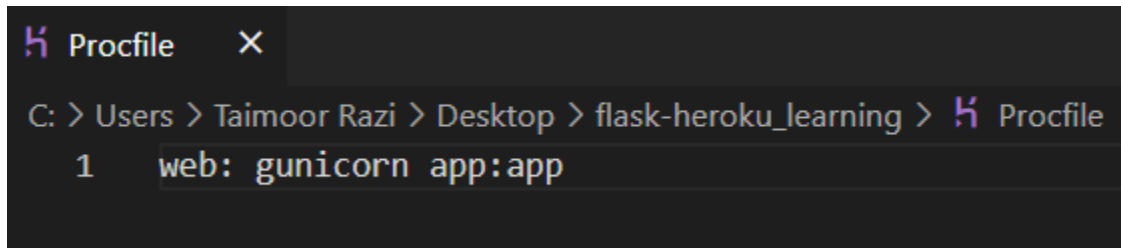
Experience	Test Score	Interview Score	Predict
------------	------------	-----------------	---------

Employee Salary should be \$ 294948.84

Figure 5: Web App Display

## Step 4: Create Procfile and requirement.txt

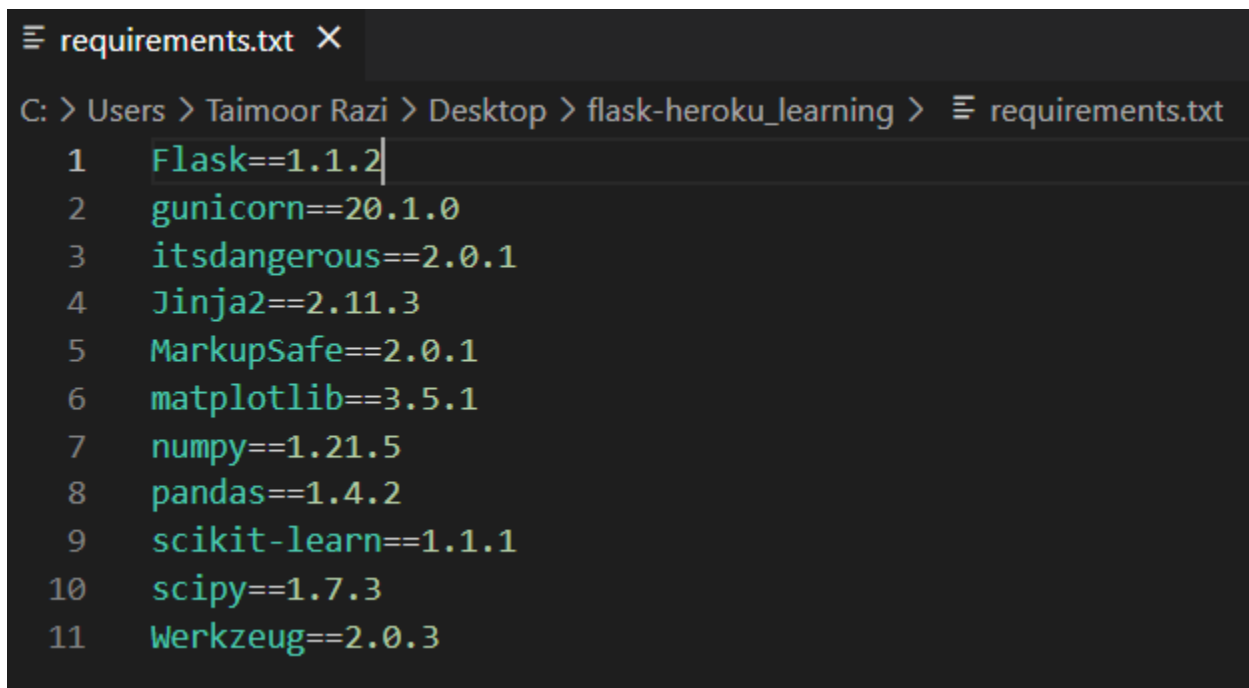
Make a Procfile specifying our Webserver as gunicorn and our app as “app.py” and “app” again as instance of our Flask object. This is shown in the following figure.

A screenshot of a code editor window titled 'Procfile'. The editor shows a single line of code: 'web: gunicorn app:app'. The file path in the top bar is 'C: > Users > Taimoor Razi > Desktop > flask-heroku\_learning > Procfile'.

```
Procfile
C: > Users > Taimoor Razi > Desktop > flask-heroku_learning > Procfile
1  web: gunicorn app:app
```

Figure 6: Procfile

Also, we will make a requirement.txt containing dependencies and their version used in our environment. This can be done manually or generated automatically. Heroku will read this to understand which packages and versions are we using. This is shown in the following figure.

A screenshot of a code editor window titled 'requirements.txt'. The editor shows a list of package versions, each on a new line, numbered 1 through 11. The file path in the top bar is 'C: > Users > Taimoor Razi > Desktop > flask-heroku\_learning > requirements.txt'.

```
requirements.txt
C: > Users > Taimoor Razi > Desktop > flask-heroku_learning > requirements.txt
1  Flask==1.1.2
2  gunicorn==20.1.0
3  itsdangerous==2.0.1
4  Jinja2==2.11.3
5  MarkupSafe==2.0.1
6  matplotlib==3.5.1
7  numpy==1.21.5
8  pandas==1.4.2
9  scikit-learn==1.1.1
10 scipy==1.7.3
11 Werkzeug==2.0.3
```

Figure 7: requirements.txt

## Step 5: Upload files on Github repository

Next, we upload all the files necessary (model.pkl, app.py, Procfile, html.index, requirements.txt) in a new Github repository. The result of this step is shown in the figure below.

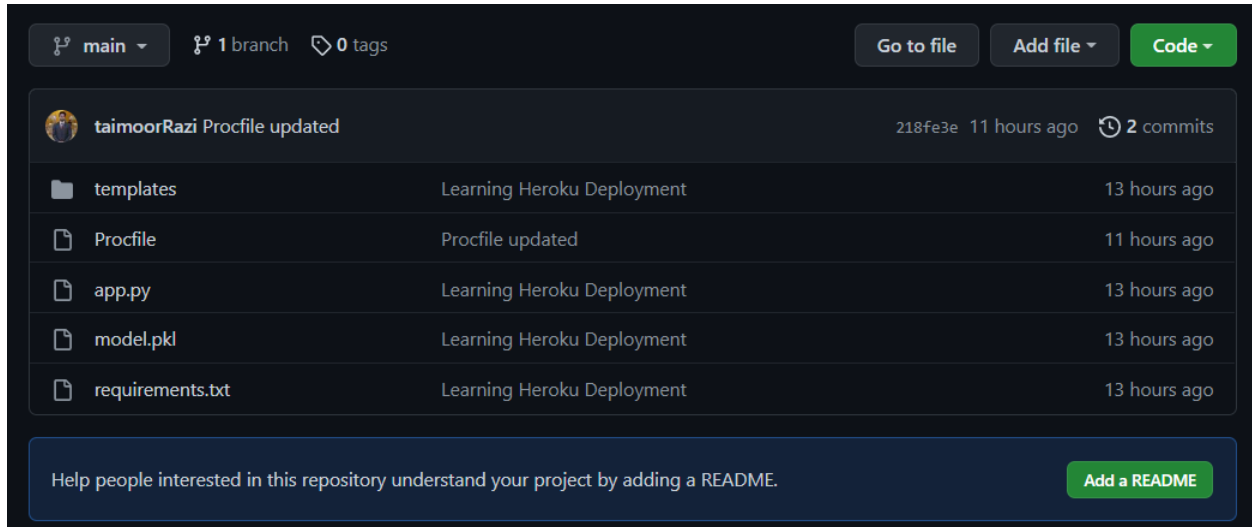


Figure 8: Files uploaded to Github repository



## Step 6: Creating and Connecting Heroku app to Github repository

Then we create an account on Heroku platform and create an app with name “salarypredictapi1”. And connect the deployment of app to our github repository.

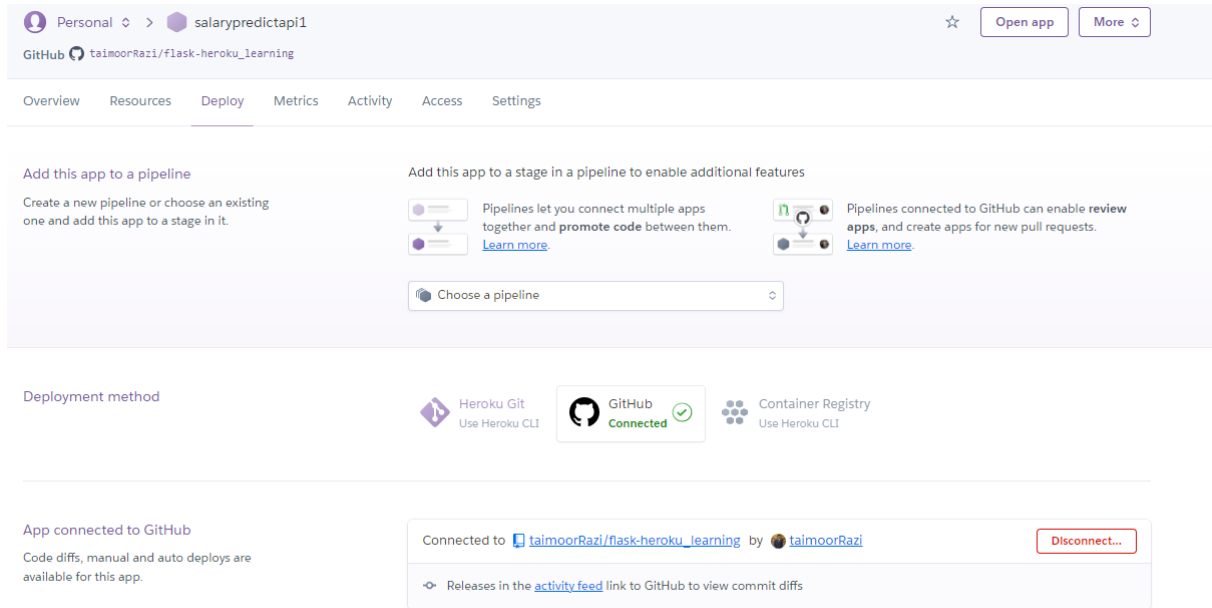


Figure 9: Heroku app connected to Github repository

## Step 7: Deploying app on Heroku

And then finally we deploy it and the result shown in the following figures.

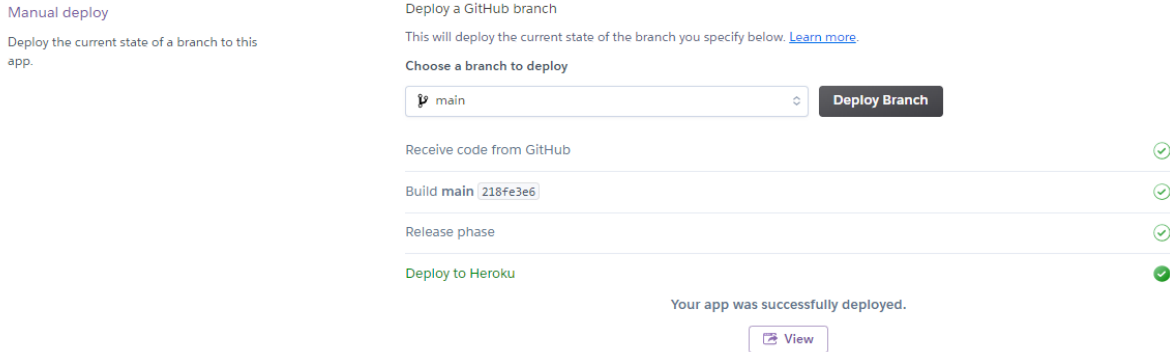


Figure 10: App deployed on Heroku

The app gets built on Heroku platform and the url of app where it is hosted is provided. The app looks exactly the same how it looked on our local system. It can be tested by putting some input features and see if it is working as intended.

## Predict Salary Analysis

Experience	Test Score	Interview Score	Predict
------------	------------	-----------------	---------

Employee Salary should be \$ 294948.84

Figure 11: App display