

Week 4: Deployment on Flask

Name: Marija Babić

Batch code: LISUM14

Submission date: 2022-10-22

Submitted to: <https://github.com/marija0408/NLP-Data-Science-Internship/tree/main/Week%204>

1. Finding the dataset

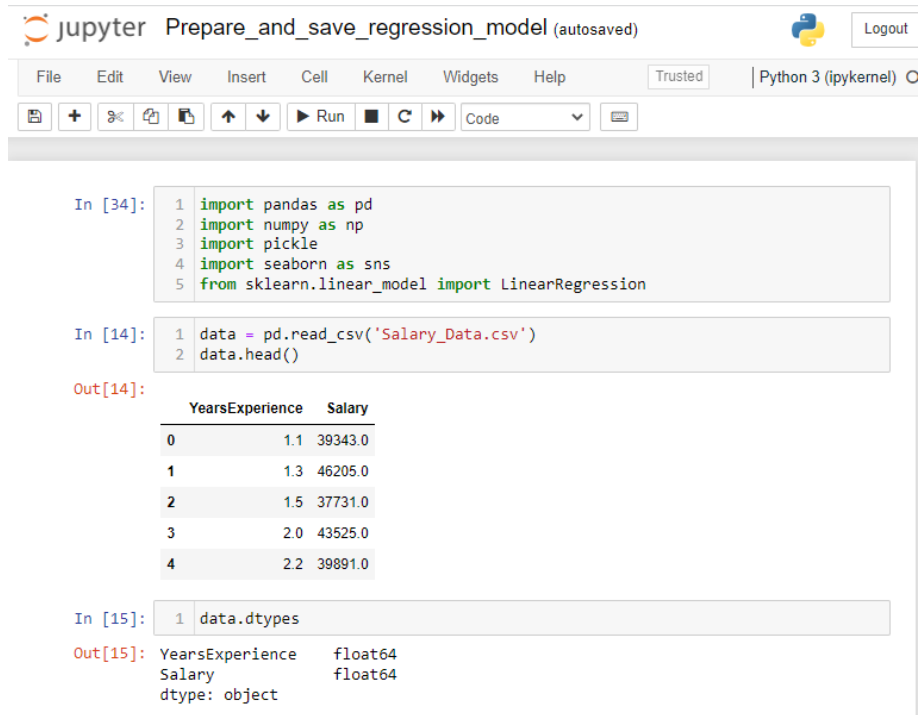
I have found a simple dataset for linear regression on Kaggle. The link is below:

<https://www.kaggle.com/datasets/karthickveerakumar/salary-data-simple-linear-regression>

It is a small dataset, 30 observations all together, but enough for me to perform a simple model. The dataset has two columns: YearsExperience and Salary. YearsExperience would be independent variable (or X in my code) and Salary would be dependent variable (y in my code).

2. Fitting the model and saving the model

The next step is fitting the model. I created ipynb notebook called *Prepare_and_save_regression_model.ipynb*. After reading the data from the csv file downloaded from Kaggle website, I checked a couple of things.



The image shows a Jupyter Notebook interface with the title "Prepare_and_save_regression_model (autosaved)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running code, and other functions. The notebook is running on a Python 3 (ipykernel) environment.

```
In [34]: 1 import pandas as pd
          2 import numpy as np
          3 import pickle
          4 import seaborn as sns
          5 from sklearn.linear_model import LinearRegression
```

```
In [14]: 1 data = pd.read_csv('Salary_Data.csv')
          2 data.head()
```

Out[14]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

```
In [15]: 1 data.dtypes
```

Out[15]: YearsExperience float64
Salary float64
dtype: object

Figure 1: Reading the data and checking column types

The first thing is checking column types (everything was good, both variables were in float format) and the second thing is describe function which gave me info about number of values, mean, std, min column values, max column values etc. Everything looked good, so the last thing I did to check the data quality was to plot the data to see how it looked. I used the same dataset for article writing so I have checked more things while preparing the article and knew that everything is good from the data side.

```
In [16]: 1 data.describe()
```

```
Out[16]:
```

	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

```
In [35]: 1 fig = sns.scatterplot(data = data ,x ='YearsExperience',y ='Salary')
2 fig.set_xlabel('Years of experience')
3 fig.set_ylabel('Salary')
```

```
Out[35]: Text(0, 0.5, 'Salary')
```

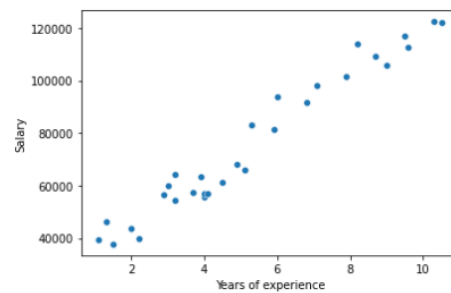


Figure 2: Checking descriptive statistics and plotting the scatterplot

After checking the data quality, it was time to fit the model and save it. I used *LinearRegression* library from *sklearn.linear_model* to fit the model and *pickle.dump* for saving the model. I also did a test where I checked the model coefficient and intercept to see if it produces the same value as the model.

```
In [44]: 1 X = data.iloc[:, :1].values
        2 y = data.iloc[:, -1]
        3
        4 regressor = LinearRegression()
        5 regressor.fit(X, y)
        6 pickle.dump(regressor, open('model.pkl', 'wb'))
```

```
In [45]: 1 regressor.coef_
```

```
Out[45]: array([9449.96232146])
```

```
In [46]: 1 regressor.intercept_
```

```
Out[46]: 25792.20019866871
```

```
In [47]: 1 regressor.predict([[3]])
```

```
Out[47]: array([54142.08716303])
```

```
In [48]: 1 #check result
        2 3*regressor.coef_[0] + 25792.20019866871
```

```
Out[48]: 54142.08716303393
```

Figure 3: Fitting the model, saving the model and checking the regressor coefficient and intercept

3. Deployment on flask

The third and the last step is deployment on Flask. I downloaded the Flask Code from LISUM14 Resources to use it as a guideline. I had to change app.py and index.html a bit so it would work on my model.

In app.py I had to change a couple of things:

- Cast input features to float because my model expects floats
- Add one variable, *input_feature*, which I used in the output sentence
- Change the output sentence
- Add some parameters to *app.run* because otherwise the code wouldn't run on my computer. I changed the host to localhost and port to 9874

```

app.py U X
app.py > predict
1 import numpy as np
2 from flask import Flask, request, render_template
3 import pickle
4
5 app = Flask(__name__)
6 model = pickle.load(open('model.pkl', 'rb'))
7
8 @app.route('/')
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
18     float_features = [float(x) for x in request.form.values()]
19     final_features = [np.array(float_features)]
20     prediction = model.predict(final_features)
21
22     input_feature = float_features[0]
23     output = round(prediction[0], 2)
24
25     return render_template('index.html', prediction_text=f'Salary for {input_feature} years of experience should be ${output}')
26
27 if __name__ == "__main__":
28     app.run(debug=True, host='localhost', port=9874)
29

```

Figure 4: App.py for my model

In index.html I changed the *input* type to be *YearsExperience*:

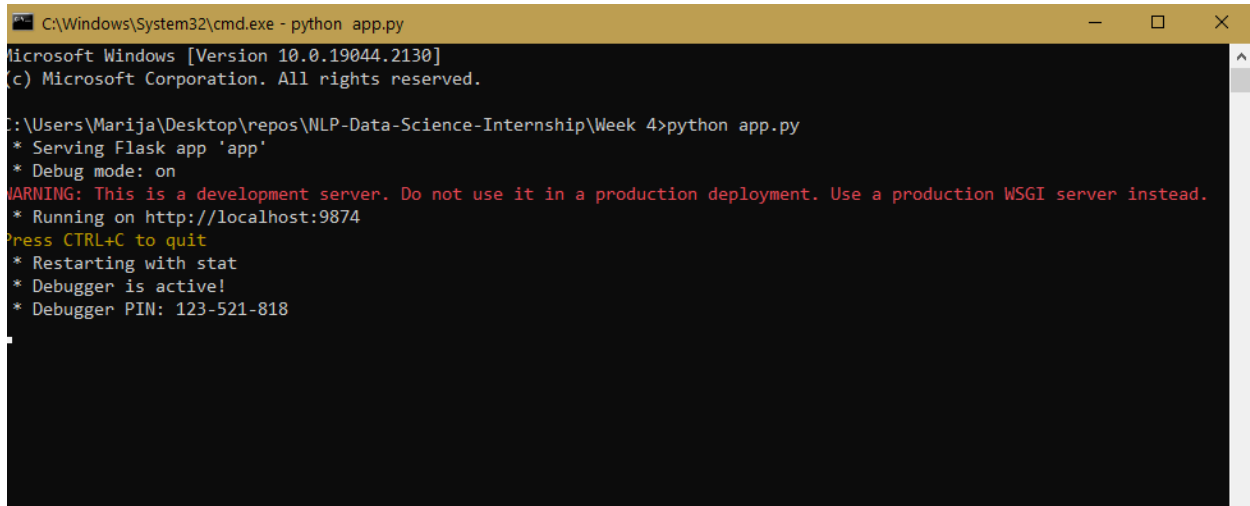
```

index.html U X app.py U
templates > index.html > ...
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="UTF-8">
5     <title>ML API</title>
6     <link href="https://fonts.googleapis.com/css?family=Pacifico" rel="stylesheet" type="text/css">
7     <link href="https://fonts.googleapis.com/css?family=Arimo" rel="stylesheet" type="text/css">
8     <link href="https://fonts.googleapis.com/css?family=Hind:300" rel="stylesheet" type="text/css">
9     <link href="https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300" rel="stylesheet" type="text/css">
10    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
11
12 </head>
13
14 <body>
15     <div class="login">
16         <h1>Predict House Price</h1>
17
18         <!-- Main Input For Receiving Query to our ML -->
19         <form action="{{ url_for('predict') }}" method="post">
20             <input type="text" name="YearsExperience" placeholder="Years of experience" required="required" />
21             <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
22         </form>
23
24         <br>
25         <br>
26         {{ prediction_text }}
27
28     </div>
29     
30
31 </body>
32 </html>
33

```

Figure 5: Changed the input type

After all of the changes written above, I was ready to start my app. I ran cmd from the Week 4 folder and ran the script:



```
C:\Windows\System32\cmd.exe - python app.py
Microsoft Windows [Version 10.0.19044.2130]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Marija\Desktop\repos\NLP-Data-Science-Internship\Week 4>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://localhost:9874
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 123-521-818
```

Figure 6: Running app.py from Week 4 folder

I copied the url and pasted it to the Chrome browser:

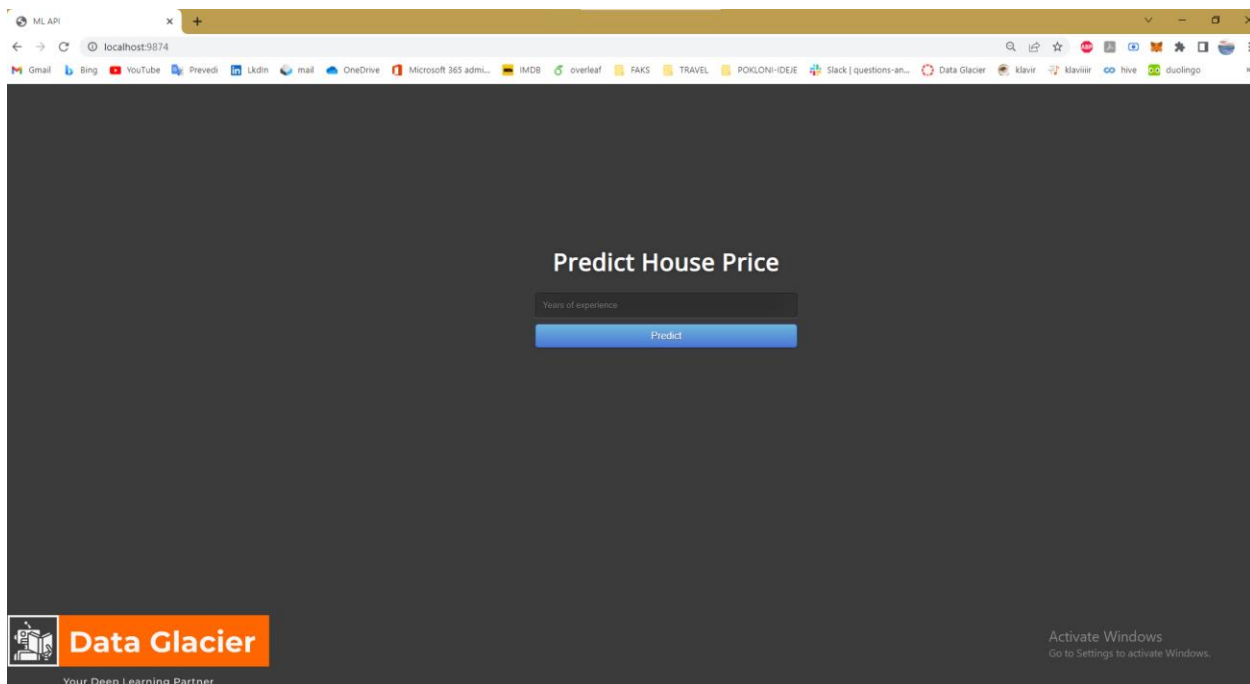


Figure 7: The app interface

I wanted to test the app. In my *Prepare_and_save_regression_model.ipynb* notebook there is a part of code where I test the prediction of one value of years of experience. The output was:

```
In [47]: 1 regressor.predict([[3]])  
Out[47]: array([54142.08716303])
```

Figure 8: Model prediction of Salary for 3 years of experience

I will insert the same value in the app to see if the results match. This is the screen before I clicked Predict button:

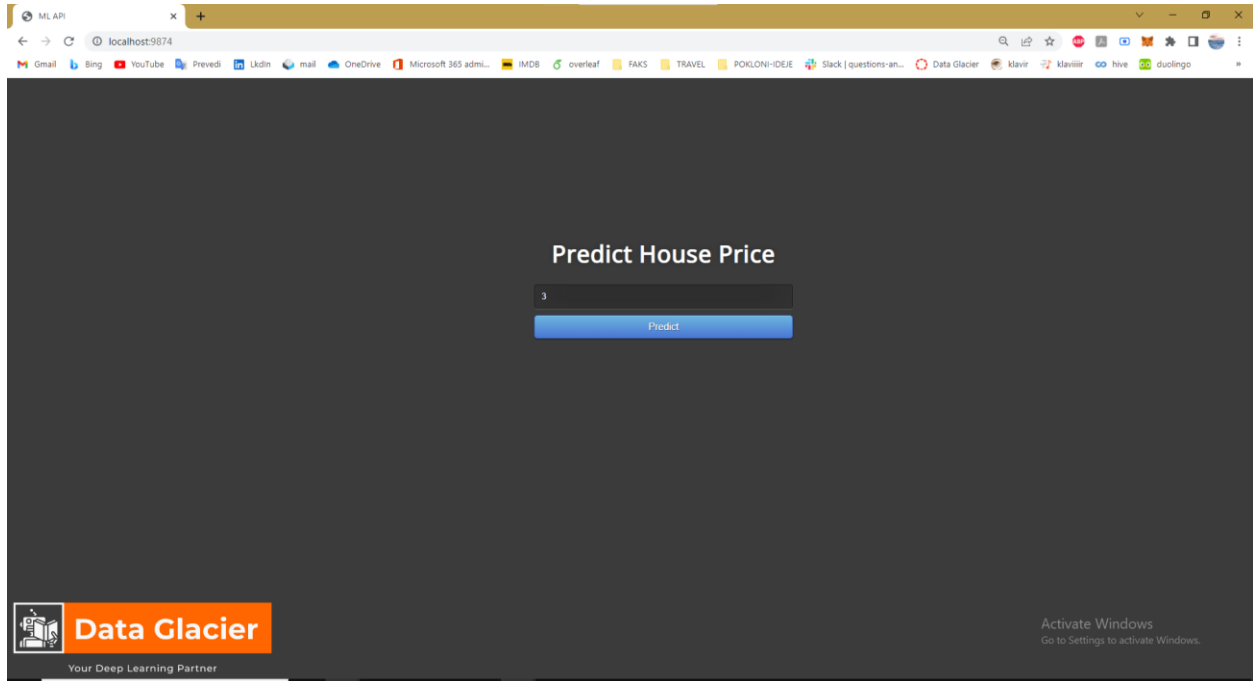


Figure 9: Checking the prediction

This is the screen after I clicked the Predict button:

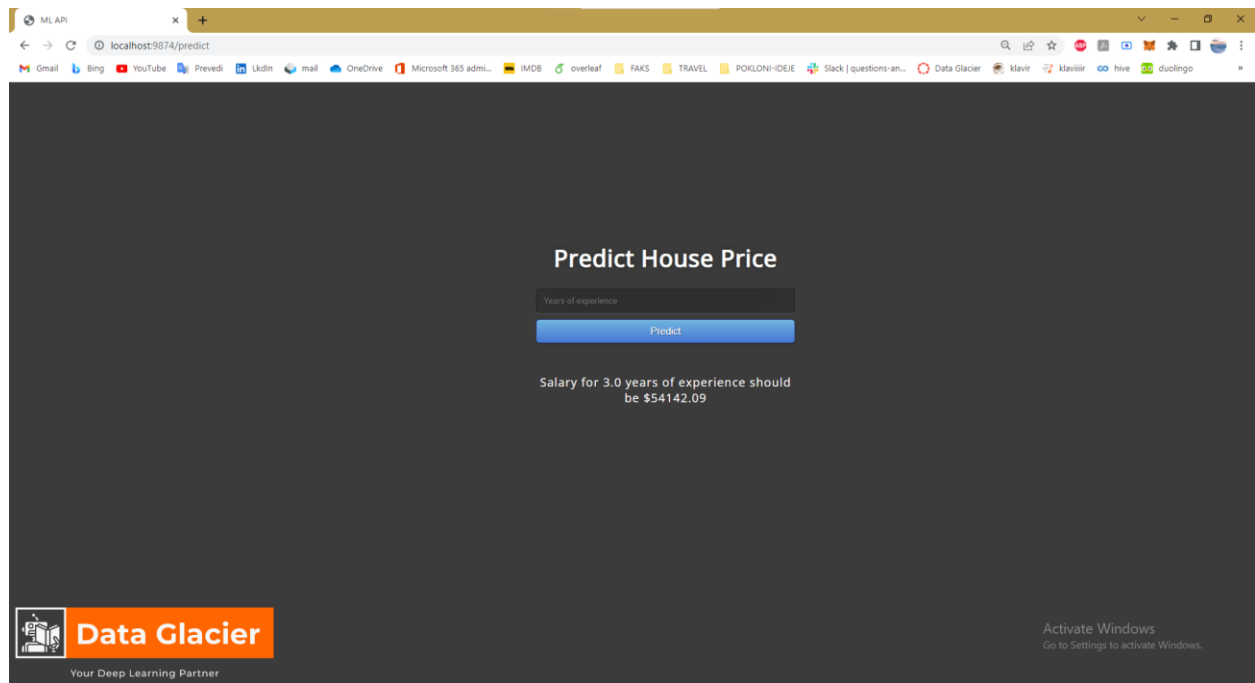


Figure 10: The prediction result

We can see that this result matches the one I got in the python notebook.

The app is running properly and gives the correct prediction value of the saved model.