Cloud and API Deployment (Week 5)

Name: Sanjukta Choudhury

Batch Code: LISUM12

Submitted To: Data Glacier

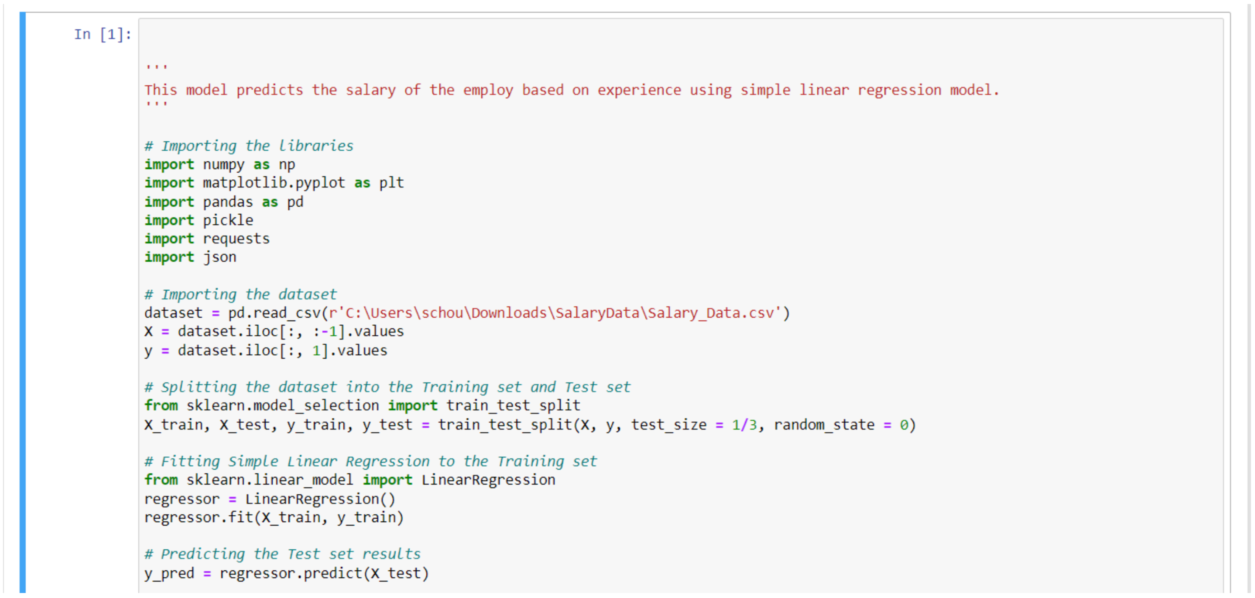
Submitted Date: 09/03/2022

Deploying a Machine Learning Model using Flask

1. Model.py

In this file we will develop our ML model and train it. We will predict the salary of an employee based on his/her experience in the field.

Importing the libraries that we are going to use to develop our model. *numpy* and *pandas* to manipulate the matrices and data respectively, *sklearn.model_selection* for splitting data into train and test set and *sklearn.linear_model* to train our model using *LinearRegression*

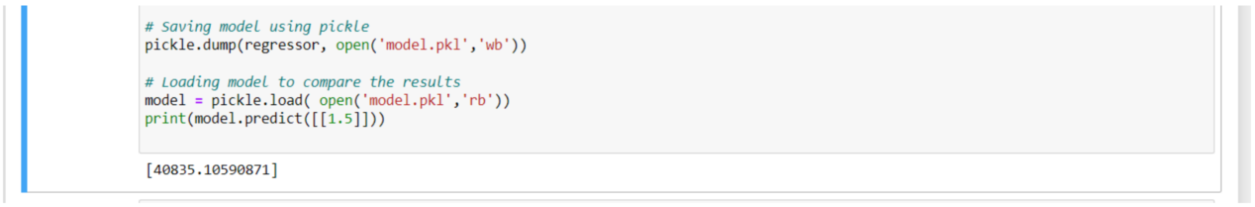


```
In [1]:
'''
This model predicts the salary of the employ based on experience using simple linear regression model.
'''

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pickle
import requests
import json

# Importing the dataset
dataset = pd.read_csv(r'C:\Users\schou\Downloads\SalaryData\Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)

# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predicting the Test set results
y_pred = regressor.predict(X_test)
```

We will save our trained model to the disk using the *pickle* library. *Pickle* is used to serializing and de-serializing a Python object structure. In which python object is converted into the byte stream. *dump()* method dumps the object into the file specified in the arguments.

In our case, we want to save our model so that it can be used by the server. So we will save our object *regressor* to the file named *model.pkl.*

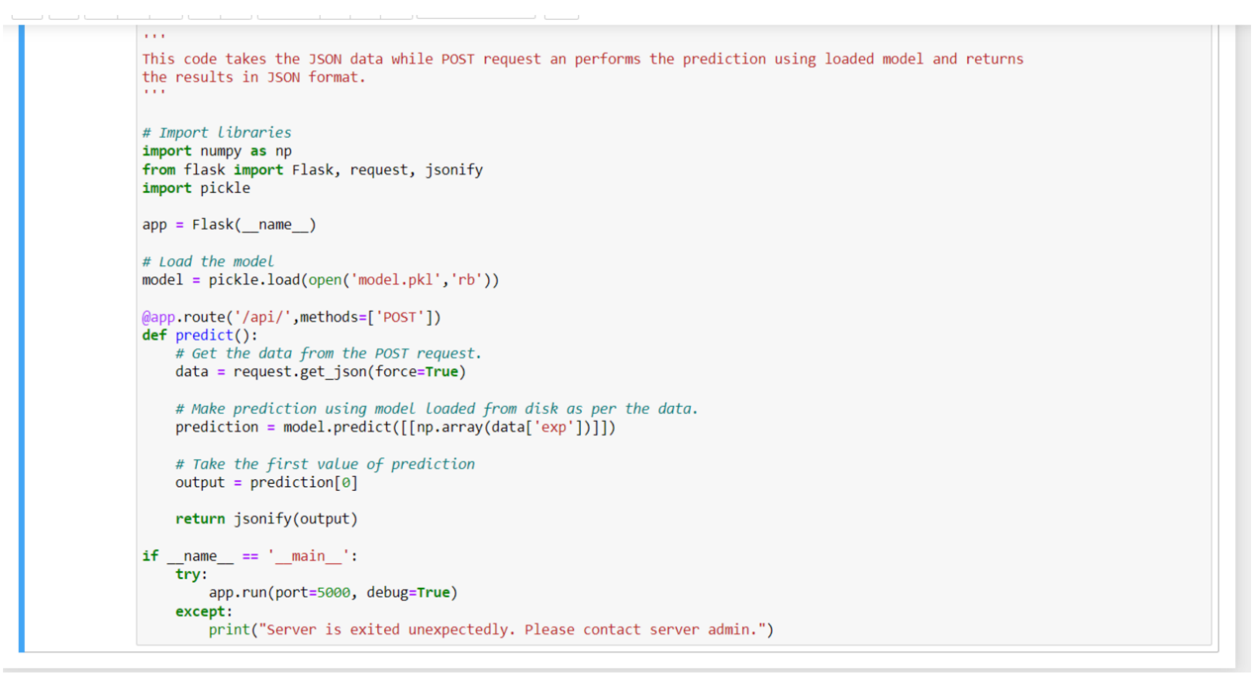
pickle.load() method loads the method and saves the deserialized bytes to *model.* Predictions can be done using *model.predict().*



```
# Saving model using pickle
pickle.dump(regressor, open('model.pkl','wb'))

# Loading model to compare the results
model = pickle.load( open('model.pkl','rb'))
print(model.predict([[1.5]]))

[40835.10590871]
```

2. Server.py

Here we have imported *numpy* to create the array of requested data, *pickle* to load our trained model to predict. We have created the instance of the *Flask()* and loaded the model into the *model.* We have bounded */api* with the method *predict().* In which predict method gets the data from the json passed by the requestor. *model.predict()* method takes input from the json and converts it into 2D *numpy array* the results are stored into the variable named *output* and we return this variable after converting it into the json object using flasks *jsonify()* method. Finally, we will run our server by following code section. Here I have used port 5000 and have set *debug=True* since if we get any error we can debug it and solve it.

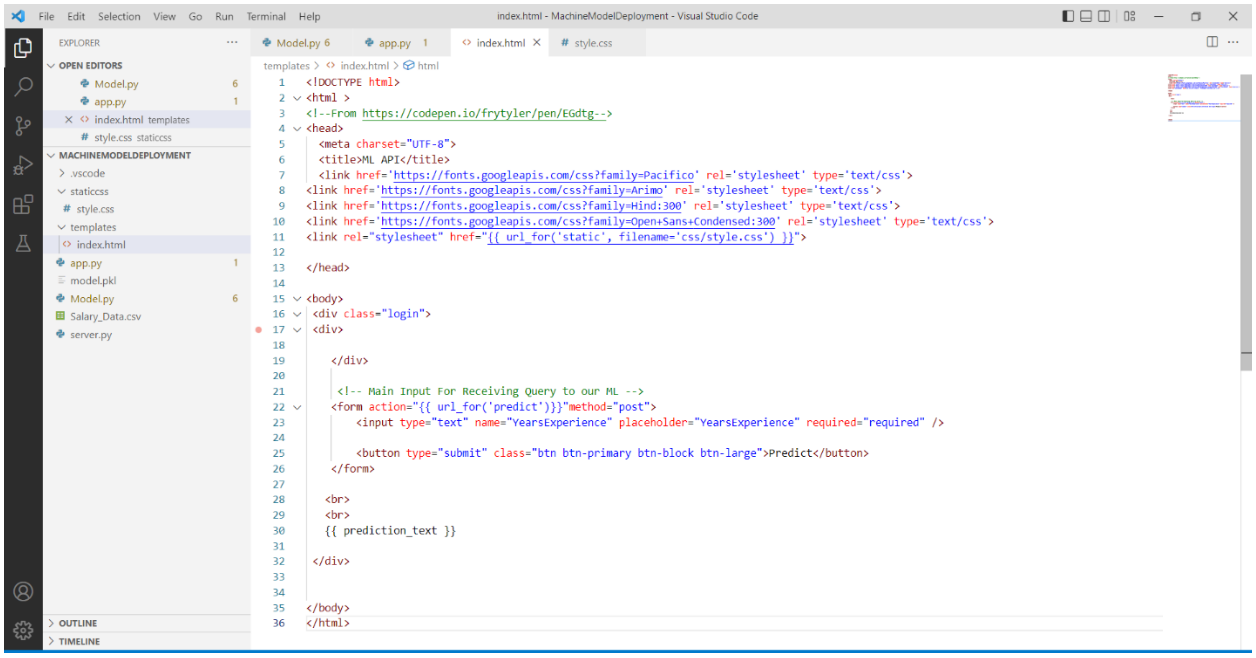


```
'''
This code takes the JSON data while POST request an performs the prediction using loaded model and returns
the results in JSON format.
'''

# Import libraries
import numpy as np
from flask import Flask, request, jsonify
import pickle

app = Flask(__name__)

# Load the model
model = pickle.load(open('model.pkl','rb'))

@app.route('/api/',methods=['POST'])
def predict():
    # Get the data from the POST request.
    data = request.get_json(force=True)

    # Make prediction using model loaded from disk as per the data.
    prediction = model.predict([[np.array(data['exp'])]])

    # Take the first value of prediction
    output = prediction[0]

    return jsonify(output)

if __name__ == '__main__':
    try:
        app.run(port=5000, debug=True)
    except:
        print("Server is exited unexpectedly. Please contact server admin.")
```

3. Turning Model into Web Application

We develop a web application that consist of a simple web page with a field that let us enter the years of experience. After submitting the value to the web application, it will render the salary.

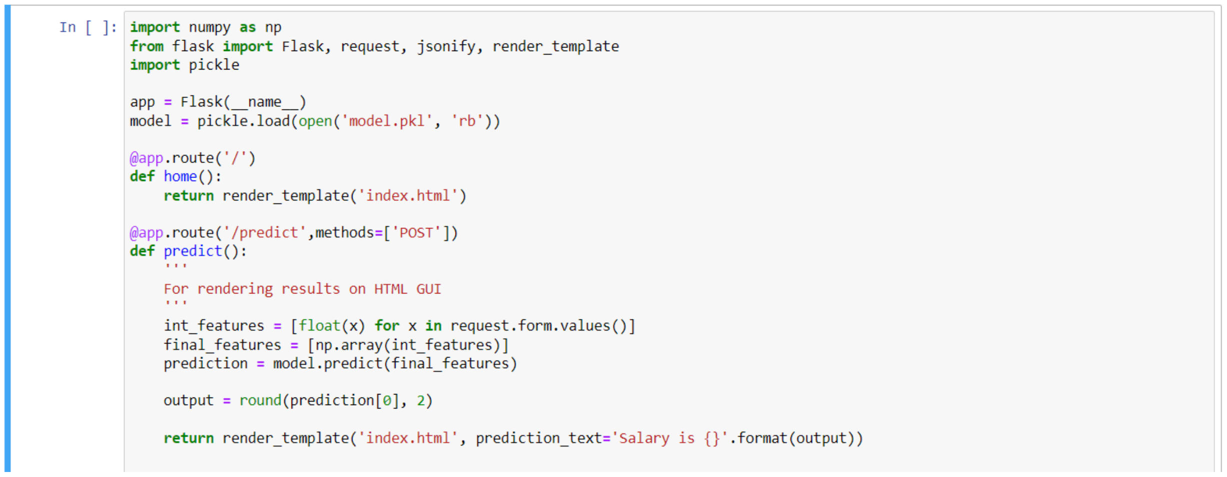
We have created a folder called templates in which flask will look for index.html file for rendering in the web browser.



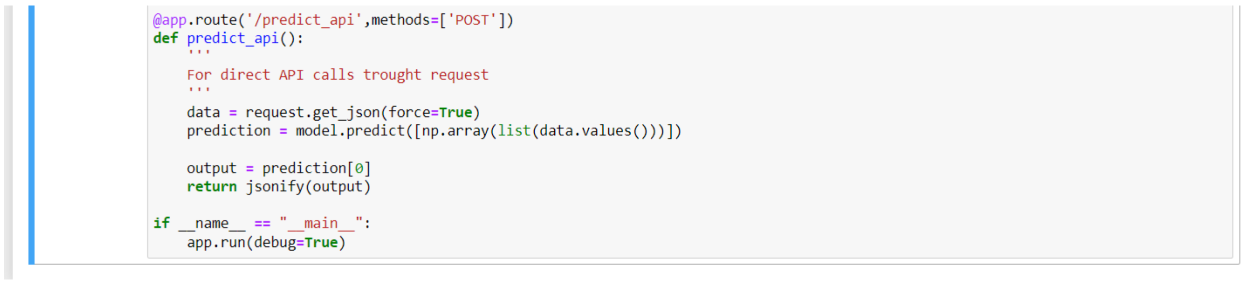
```html
<!DOCTYPE html>
<html >
<!--From https://codepen.io/frytyler/pen/EGdtg-->
<head>
  <meta charset="UTF-8">
  <title>ML API</title>
  <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">

</head>

<body>
 <div class="login">
 <div>

    </div>

     <!-- Main Input For Receiving Query to our ML -->
    <form action="{{ url_for('predict')}}"method="post">
        <input type="text" name="YearsExperience" placeholder="YearsExperience" required="required" />

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

   <br>
   <br>
   {{ prediction_text }}

 </div>


</body>
</html>
```

4. App.py

The app.py file contains the main code that will be executed by the python interpreter to run the flask web application. We used the route decorator to specify the URL that should trigger the execution of the home function.



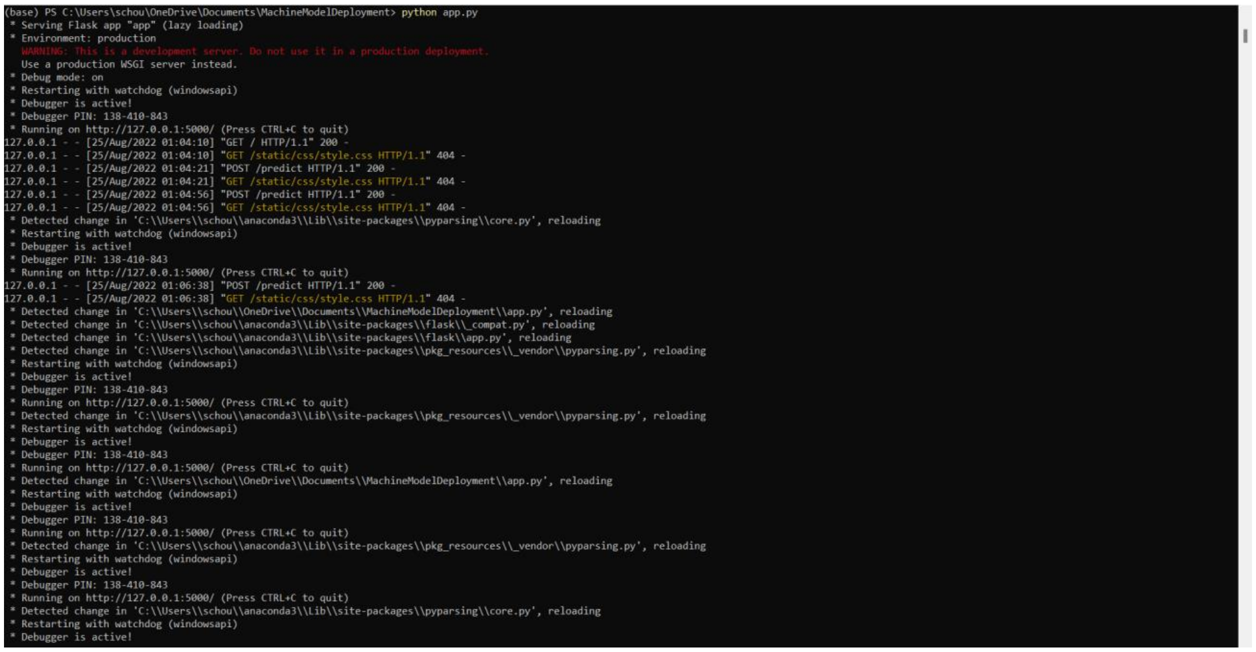
```
In [ ]: import numpy as np
        from flask import Flask, request, jsonify, render_template
        import pickle

        app = Flask(__name__)
        model = pickle.load(open('model.pkl', 'rb'))

        @app.route('/')
        def home():
            return render_template('index.html')

        @app.route('/predict',methods=['POST'])
        def predict():
            '''
            For rendering results on HTML GUI
            '''
            int_features = [float(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='Salary is {}'.format(output))
```



```
@app.route('/predict_api',methods=['POST'])
def predict_api():
    '''
    For direct API calls trought request
    '''
    data = request.get_json(force=True)
    prediction = model.predict([np.array(list(data.values()))])

    output = prediction[0]
    return jsonify(output)

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

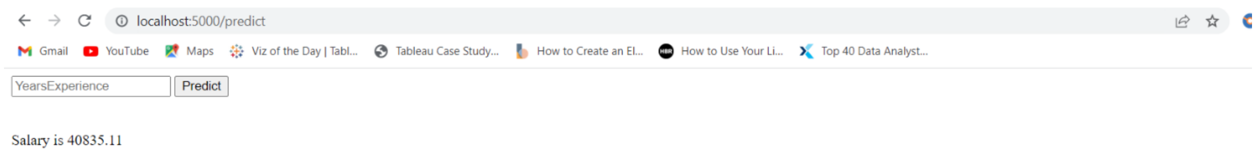
5. Running Procedures

Once we are done coding the file, we will run the file on the terminal. First, we will run model.py, then server.py and lastly app.py.



```
(base) PS C:\Users\schou\OneDrive\Documents\MachineModelDeployment> python app.py
 * 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: on
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 138-410-843
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [25/Aug/2022 01:04:10] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [25/Aug/2022 01:04:10] "GET /static/css/style.css HTTP/1.1" 404 -
127.0.0.1 - - [25/Aug/2022 01:04:21] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [25/Aug/2022 01:04:21] "GET /static/css/style.css HTTP/1.1" 404 -
127.0.0.1 - - [25/Aug/2022 01:04:56] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [25/Aug/2022 01:04:56] "GET /static/css/style.css HTTP/1.1" 404 -
 * Detected change in 'C:\\Users\\schou\\anaconda3\\Lib\\site-packages\\pyparsing\\core.py', reloading
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 138-410-843
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [25/Aug/2022 01:06:38] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [25/Aug/2022 01:06:38] "GET /static/css/style.css HTTP/1.1" 404 -
 * Detected change in 'C:\\Users\\schou\\OneDrive\\Documents\\MachineModelDeployment\\app.py', reloading
 * Detected change in 'C:\\Users\\schou\\anaconda3\\Lib\\site-packages\\flask\\_compat.py', reloading
 * Detected change in 'C:\\Users\\schou\\anaconda3\\Lib\\site-packages\\flask\\app.py', reloading
 * Detected change in 'C:\\Users\\schou\\anaconda3\\Lib\\site-packages\\pkg_resources\\_vendor\\pyparsing.py', reloading
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 138-410-843
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Detected change in 'C:\\Users\\schou\\anaconda3\\Lib\\site-packages\\pkg_resources\\_vendor\\pyparsing.py', reloading
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 138-410-843
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Detected change in 'C:\\Users\\schou\\OneDrive\\Documents\\MachineModelDeployment\\app.py', reloading
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 138-410-843
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Detected change in 'C:\\Users\\schou\\anaconda3\\Lib\\site-packages\\pkg_resources\\_vendor\\pyparsing.py', reloading
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 138-410-843
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Detected change in 'C:\\Users\\schou\\anaconda3\\Lib\\site-packages\\pyparsing\\core.py', reloading
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
```

Now we could open a web browser and navigate to localhost:5000/predict. The following screen will appear where we can enter the years of experience and click on predict which will give the respective salary.

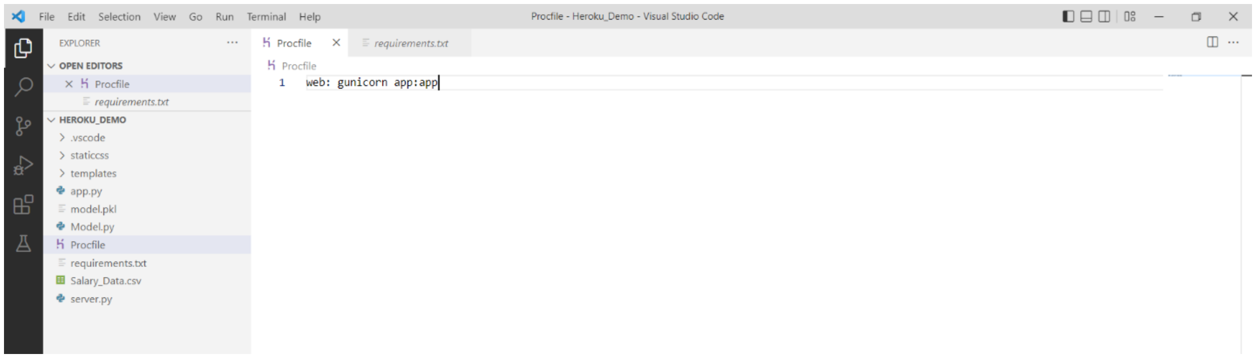


Salary is 40835.11

Deploying a Machine Learning Model using Heroku

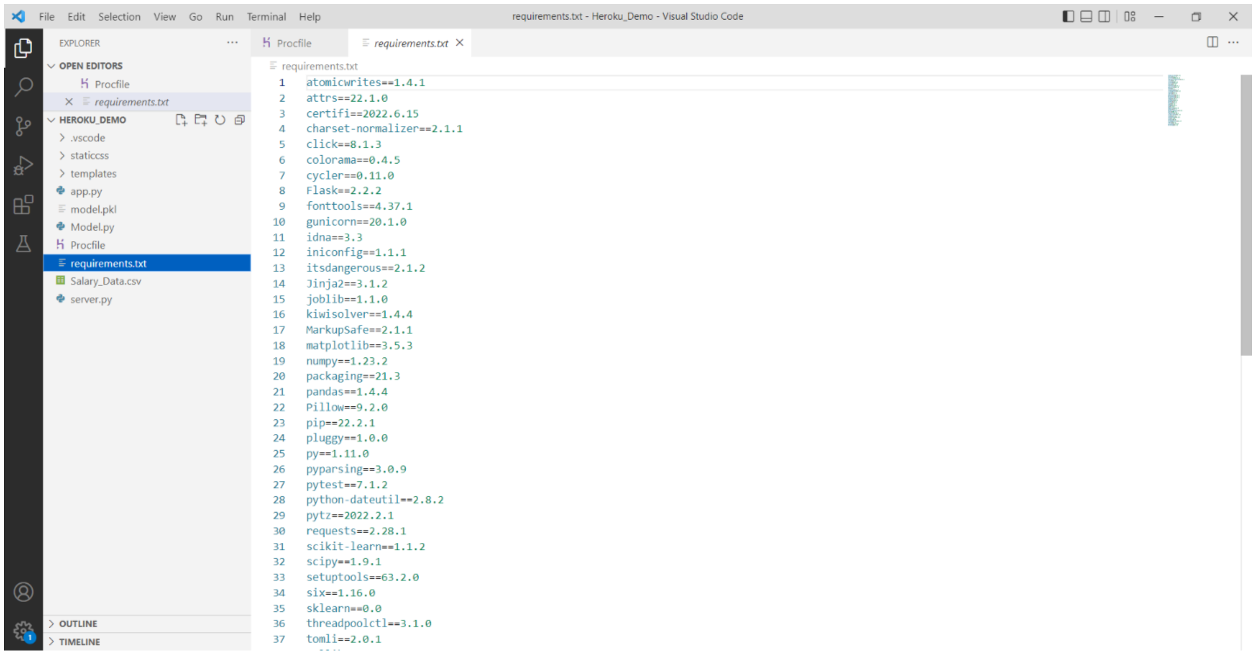
Heroku requires two more files for model deployment.

The first file is the **Procfile** that contains three pieces of information: application type, server information, and file name from where the application should start. In our model deployment folder we have to create a new file named Procfile and write the below code. Before writing the code we have to install gunicorn -> pip install gunicorn.



```
web: gunicorn app:app
```

The second file is the **requirements.txt** file that contains all the libraries used during model building.
To get the requirements.txt file we have to write pip list --format=freeze > requirements.txt



```
atomicwrites==1.4.1
attrs==22.1.0
certifi==2022.6.15
charset-normalizer==2.1.1
click==8.1.3
colorama==0.4.5
cycler==0.11.0
Flask==2.2.2
fonttools==4.37.1
gunicorn==20.1.0
idna==3.3
iniconfig==1.1.1
itsdangerous==2.1.2
Jinja2==3.1.2
joblib==1.1.0
kiwisolver==1.4.4
MarkupSafe==2.1.1
matplotlib==3.5.3
numpy==1.23.2
packaging==21.3
pandas==1.4.4
Pillow==9.2.0
pip==22.2.1
pluggy==1.0.0
py==1.11.0
pyparsing==3.0.9
pytest==7.1.2
python-dateutil==2.8.2
pytz==2022.2.1
requests==2.28.1
scikit-learn==1.1.2
scipy==1.9.1
setuptools==63.2.0
six==1.16.0
sklearn==0.0
threadpoolctl==3.1.0
tomli==2.0.1
```

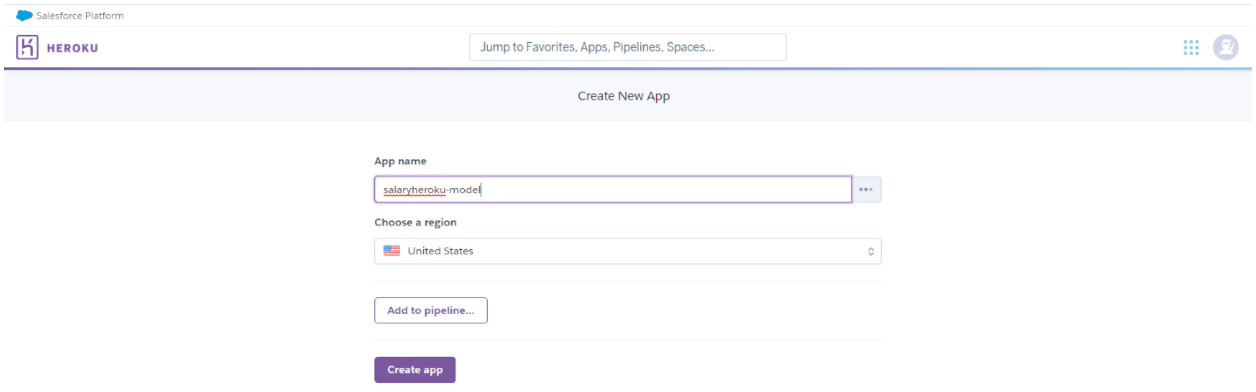
Once all required files are in the folder, the model is ready for deployment on Heroku.

Heroku offers three methods of model deployment: Heroku git, GitHub, and container registry.

I have used the GitHub method

Step 1: Upload the project folder to a git repository.

Step 2: Create an account on Heroku. Navigate to **New** and Click on **Create a new app** and give the name of the app. Please note that App name should be unique.

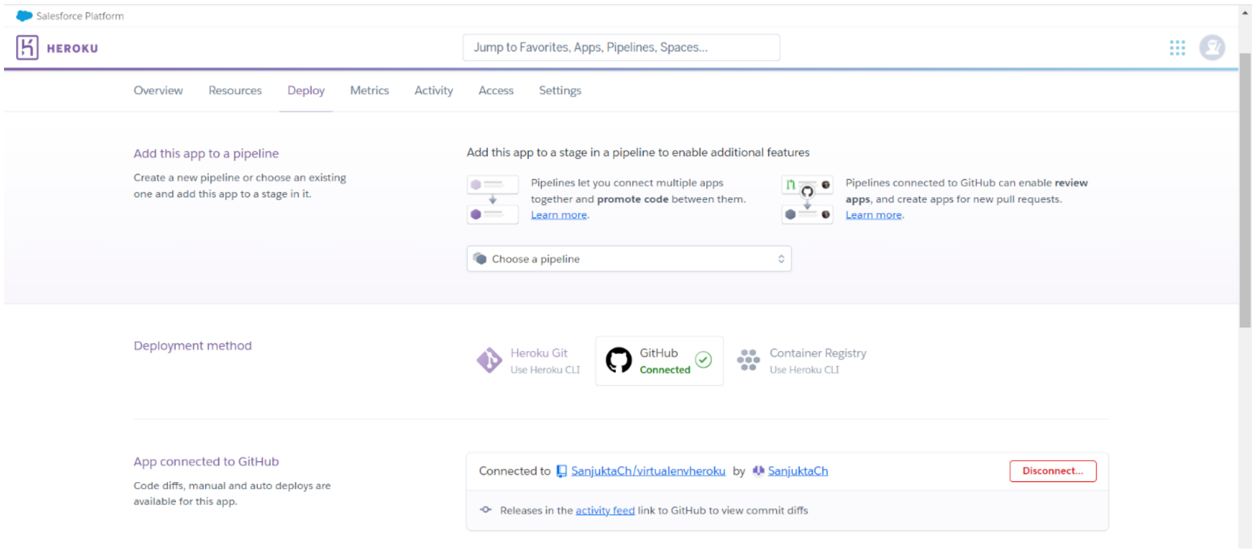


Create New App

App name: salaryheroku-model

Choose a region: United States

Step 3: Choose the deployment method, in this case, GitHub. Once Heroku is authenticated to GitHub, then choose the repository to connect.



Add this app to a pipeline

Create a new pipeline or choose an existing one and add this app to a stage in it.

Add this app to a stage in a pipeline to enable additional features

Pipelines let you connect multiple apps together and **promote code** between them. Learn more.

Pipelines connected to GitHub can enable **review apps**, and create apps for new pull requests. Learn more.

Deployment method

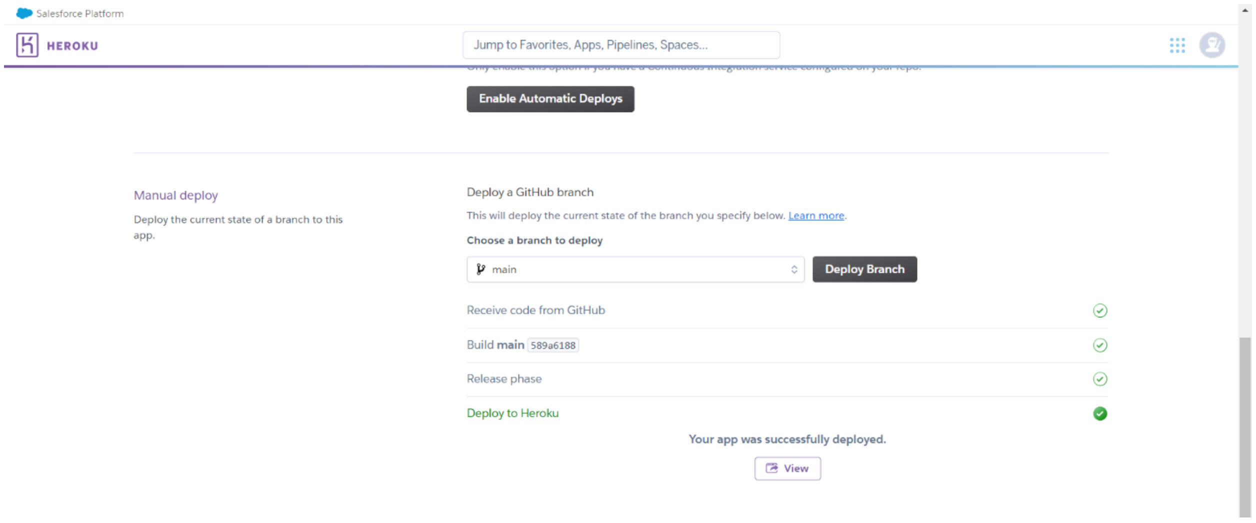
App connected to GitHub

Code diffs, manual and auto deploys are available for this app.

Connected to SanjuktaCh/virtualenvheroku by SanjuktaCh

Releases in the activity feed link to GitHub to view commit diffs

Step 4: After connecting the repository to Heroku, it is time to deploy a repository branch. Start the deployment by clicking on the Deploy branch button. Upon the successful completion of deployment of the app, click on the View button.



Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. Learn more.

Choose a branch to deploy

main

Deploy Branch

Receive code from GitHub ✓

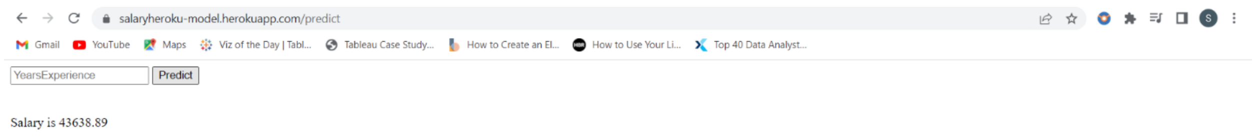
Build main 589a6188 ✓

Release phase ✓

Deploy to Heroku ✓

Your app was successfully deployed.

View


salaryheroku-model.herokuapp.com/predict

Salary is 43638.89