Data Glacier

Week 5: Cloud & API deployment

Name : Amira Asta

Batch Code : LISUM01

Submission Date : July 12th, 2021

Submitted To : Data Glacier

Table of contents

Introduction

In this document, we are going to deploy the web application of a Machine Learning model on Cloud using Heroku. As a demonstration, our Machine Learning model will help us classify the variety of flowers based on the length and width of sepals and petals. We will build two simple HTML web pages to accept the measurements as input and classify the variety based on the classification model.

Dataset

When building the Machine Learning model, we will make use of the **IRIS** dataset. This Dataset contains **four features**, length and width of sepals and petals of 50 samples of **three species** of Iris:

- Iris setosa,
- Iris virginica and
- Iris versicolor.

Four features were measured from each sample. They are:

- Sepal Length
- Sepal Width
- Petal Length
- Petal Width.

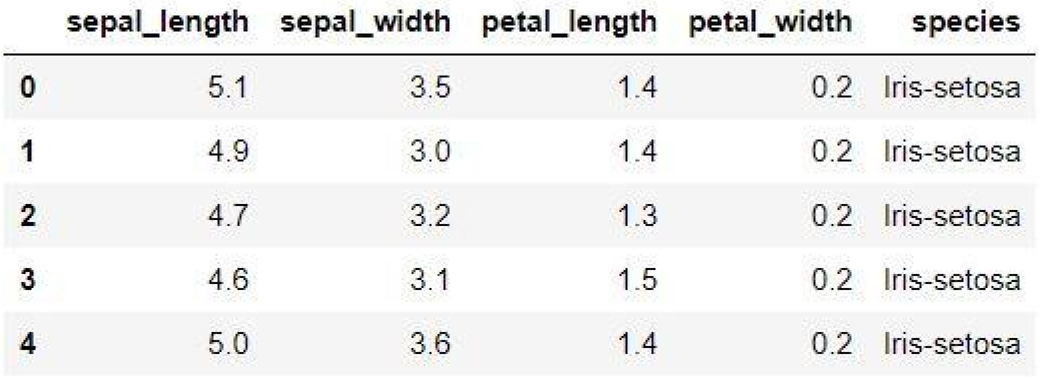
All these four parameters are measured in Centimeters. Based on the combination of these four features, the species among three can be predicted.

Machine Learning Model

Having chosen the dataset, it is time to build our classification model. First, we import the necessary Python libraries to work with for building our model. Here, we use:

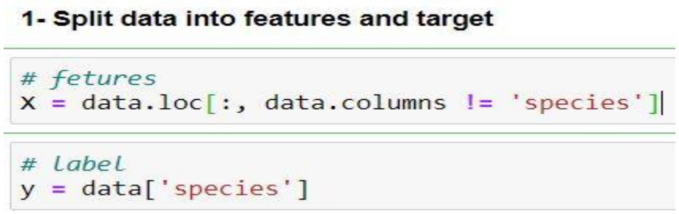
- Pandas
- Numpy
- Sklearn/Sci-kit learn

Next, we read the CSV file of our dataset `IRIS.csv` . As we can see from the following capture, the target variable is in the column '**species**':



	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In order to implement our classification model, we need to separate the independent values (features) from the dependent values (target).

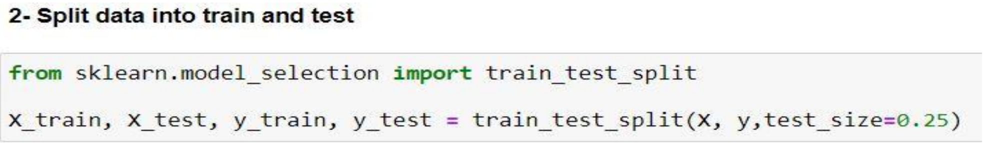


1- Split data into features and target

```
# fetures
X = data.loc[:, data.columns != 'species']
```

```
# label
y = data['species']
```

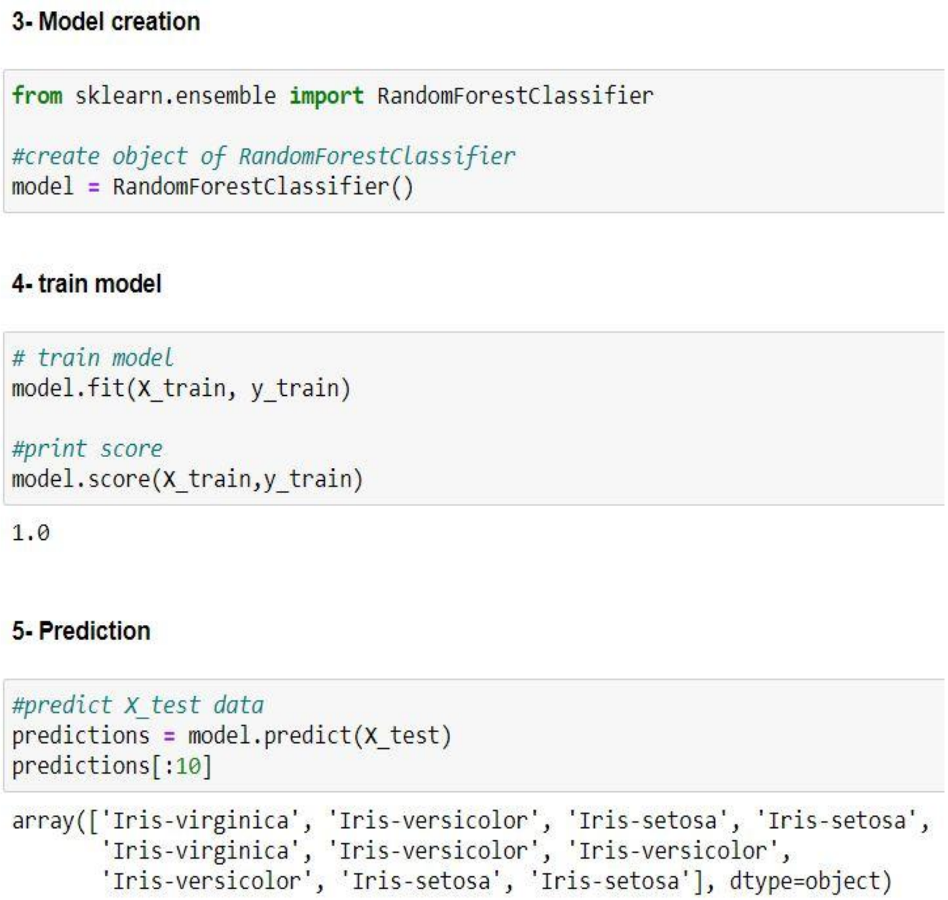
then, we split the data into train and test to train our model:



2- Split data into train and test

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.25)
```

Next, we initialize the `RandomForestClassifier()` model by calling and creating a python object and assigning it to a variable called `model`. Finally, we fit the features with the target values. This can be done by making use of the fit() function. The following capture shows how we use model to make prediction:



3- Model creation

```
from sklearn.ensemble import RandomForestClassifier

#create object of RandomForestClassifier
model = RandomForestClassifier()
```

4- train model

```
# train model
model.fit(X_train, y_train)

#print score
model.score(X_train,y_train)
```

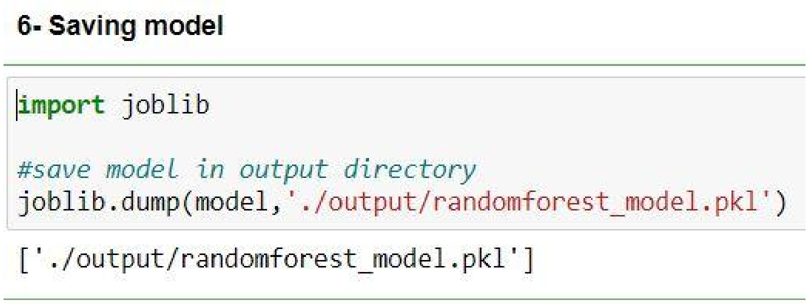
```
1.0
```

5- Prediction

```
#predict X_test data
predictions = model.predict(X_test)
predictions[:10]
```

```
array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-setosa'], dtype=object)
```

Now, our random forest model classifies the species based on the above-pre-processed input. The last thing we need to add is to save the model before using it in the deployment process. To do so, we are using the joblib model to serialize python objects. `joblib.dump()` will allow us to save the object on disk.



6- Saving model

```
import joblib

#save model in output directory
joblib.dump(model,'./output/randomforest_model.pkl')
```

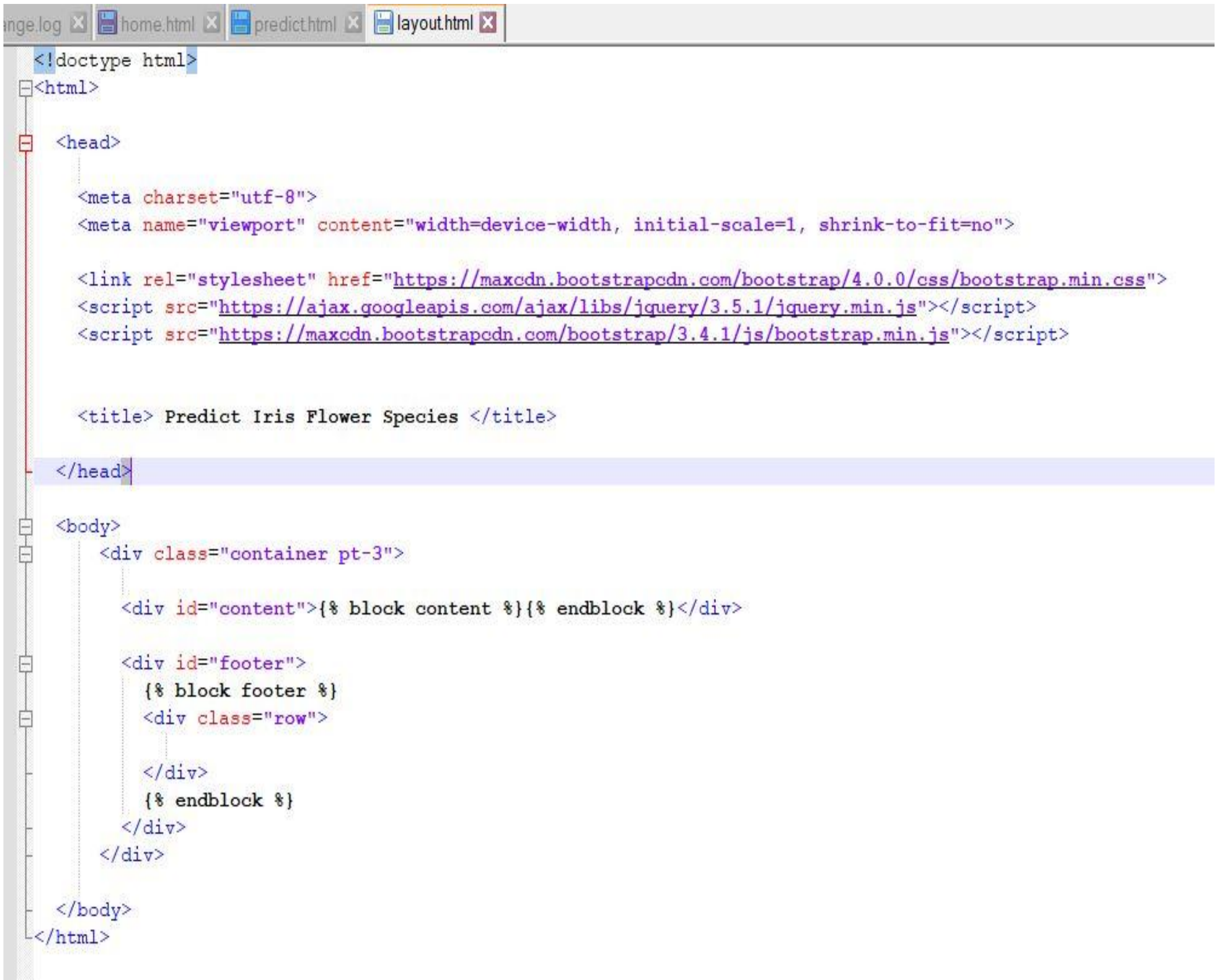
```
['./output/randomforest_model.pkl']
```

Deploy Model With Flask Web Framework

Having built our Machine Learning model, now let's build a simple form using HTML to accept the inputs from the user. We start by creating a Flask application with an `app.py` file. We create an instance of Flask, load the saved model and pass input data to model and predict.

We will use templates to render HTML which will display in the browser. The views are called by the `render_template()` function. The template files will be stored in the templates directory inside the flask package.

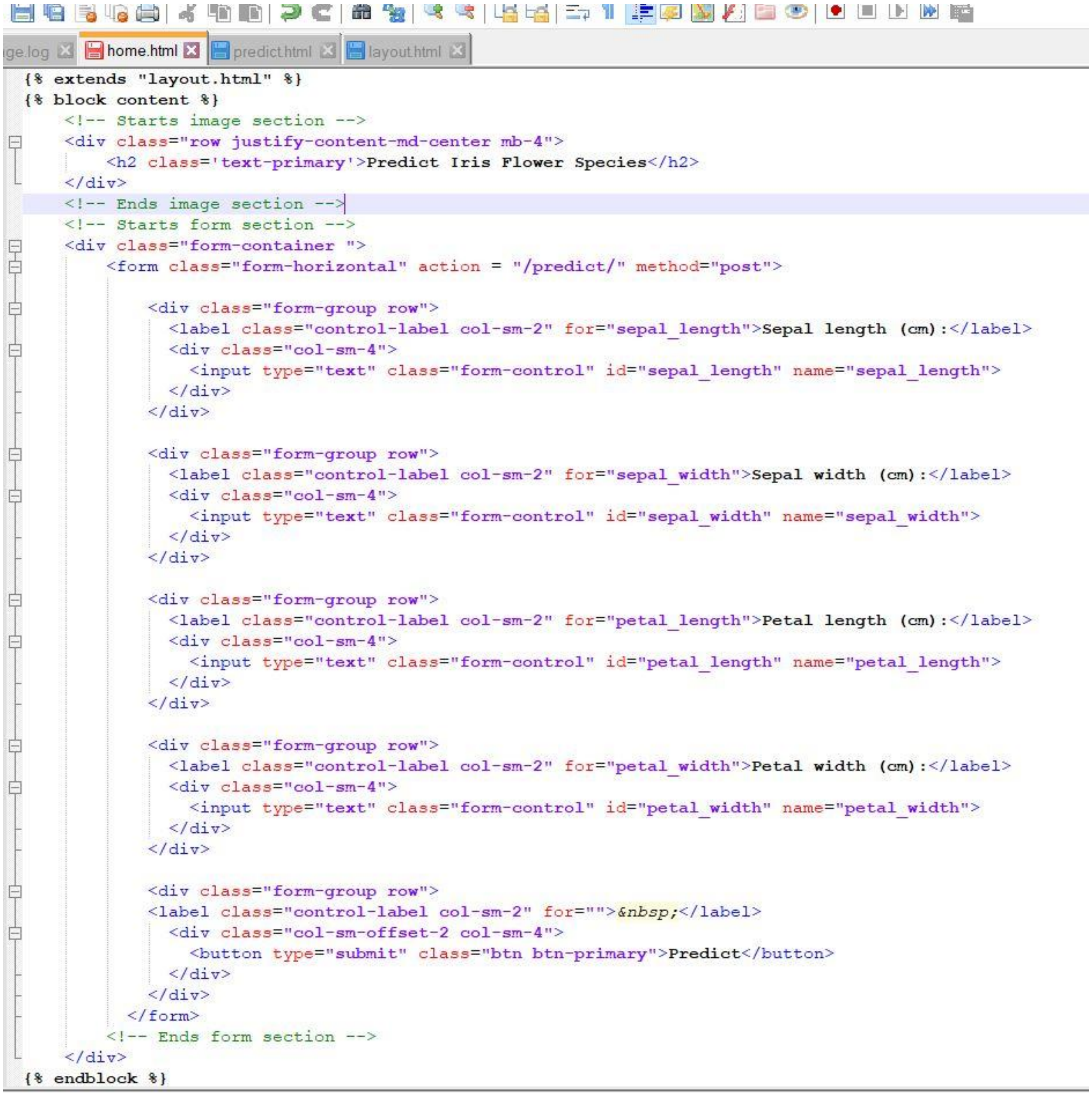
- ➔ **Create the base layout** : Each page in the application will have the same basic layout around a different body. `layout.html`



```
<!doctype html>
<html>

  <head>

    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>


    <title> Predict Iris Flower Species </title>

  </head>

  <body>
      <div class="container pt-3">

        <div id="content">{% block content %}{% endblock %}</div>

        <div id="footer">
          {% block footer %}
          <div class="row">

          </div>
          {% endblock %}
        </div>
      </div>

  </body>
</html>
```

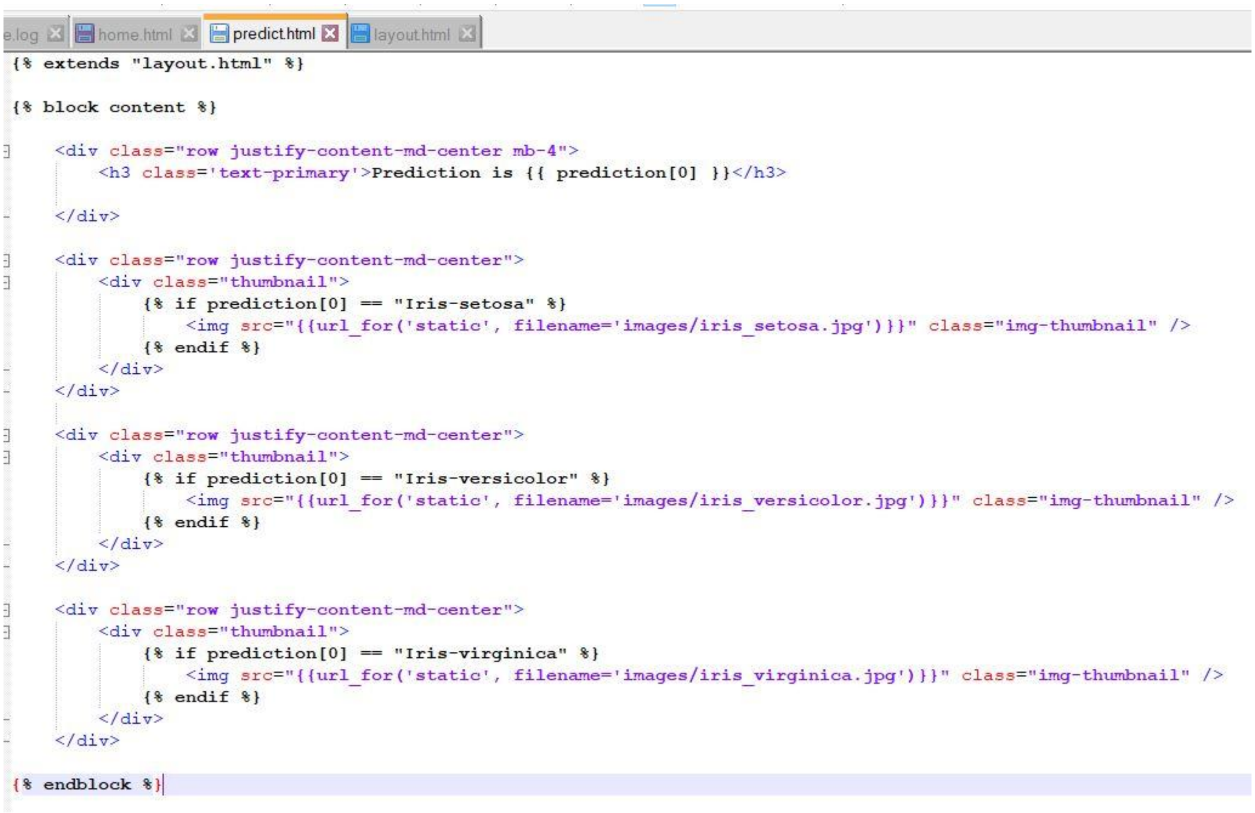
➔ **Add a static file for images** : Create a static folder and inside that images folder. After that keep images of iris flowers, which we are going to display on ‘predict’ page.

➔ **Create home page template** : `home.html`



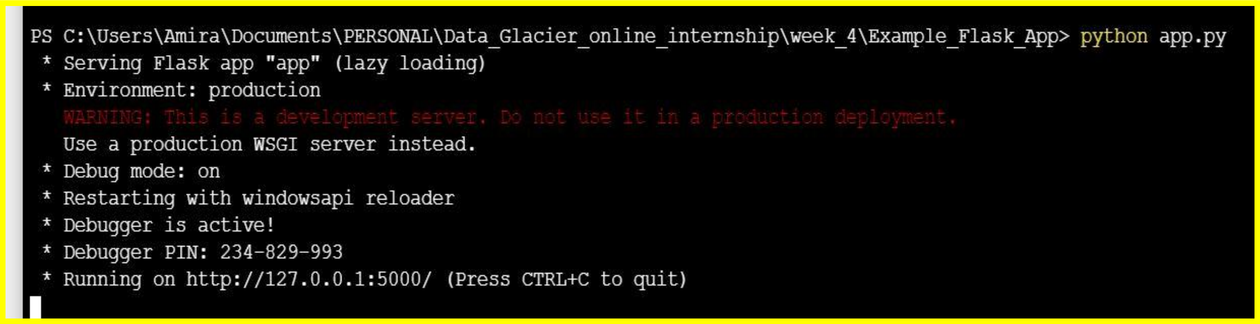
```
{% extends "layout.html" %}
{% block content %}
    <!-- Starts image section -->
    <div class="row justify-content-md-center mb-4">
        <h2 class='text-primary'>Predict Iris Flower Species</h2>
    </div>
    <!-- Ends image section -->
    <!-- Starts form section -->
    <div class="form-container ">
        <form class="form-horizontal" action = "/predict/" method="post">

            <div class="form-group row">
              <label class="control-label col-sm-2" for="sepal_length">Sepal length (cm):</label>
              <div class="col-sm-4">
                <input type="text" class="form-control" id="sepal_length" name="sepal_length">
              </div>
            </div>

            <div class="form-group row">
              <label class="control-label col-sm-2" for="sepal_width">Sepal width (cm):</label>
              <div class="col-sm-4">
                <input type="text" class="form-control" id="sepal_width" name="sepal_width">
              </div>
            </div>

            <div class="form-group row">
              <label class="control-label col-sm-2" for="petal_length">Petal length (cm):</label>
              <div class="col-sm-4">
                <input type="text" class="form-control" id="petal_length" name="petal_length">
              </div>
            </div>

            <div class="form-group row">
              <label class="control-label col-sm-2" for="petal_width">Petal width (cm):</label>
              <div class="col-sm-4">
                <input type="text" class="form-control" id="petal_width" name="petal_width">
              </div>
            </div>

            <div class="form-group row">
            <label class="control-label col-sm-2" for="">&nbsp;</label>
              <div class="col-sm-offset-2 col-sm-4">
                <button type="submit" class="btn btn-primary">Predict</button>
              </div>
            </div>
          </form>
        <!-- Ends form section -->
    </div>
{% endblock %}
```

➔ **Create a page for prediction** : When we submit a form from the home page, it will go to /predict/ url. `predict.html`



```
{% extends "layout.html" %}

{% block content %}

    <div class="row justify-content-md-center mb-4">
        <h3 class='text-primary'>Prediction is {{ prediction[0] }}</h3>

    </div>

    <div class="row justify-content-md-center">
        <div class="thumbnail">
            {% if prediction[0] == "Iris-setosa" %}
                <img src="{{url_for('static', filename='images/iris_setosa.jpg')}}" class="img-thumbnail" />
            {% endif %}
        </div>
    </div>

    <div class="row justify-content-md-center">
        <div class="thumbnail">
            {% if prediction[0] == "Iris-versicolor" %}
                <img src="{{url_for('static', filename='images/iris_versicolor.jpg')}}" class="img-thumbnail" />
            {% endif %}
        </div>
    </div>

    <div class="row justify-content-md-center">
        <div class="thumbnail">
            {% if prediction[0] == "Iris-virginica" %}
                <img src="{{url_for('static', filename='images/iris_virginica.jpg')}}" class="img-thumbnail" />
            {% endif %}
        </div>
    </div>

{% endblock %}
```

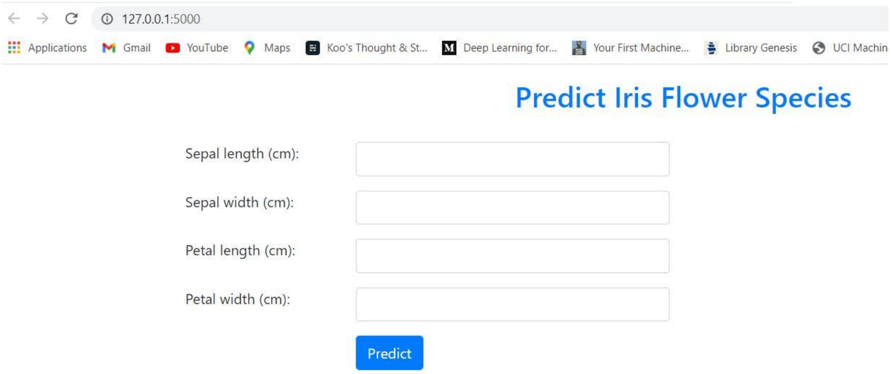
➔ **Run the server**: `python app.py`



```
PS C:\Users\Amira\Documents\PERSONAL\Data_Glacier_online_internship\week_4\Example_Flask_App> 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 windowsapi reloader
 * Debugger is active!
 * Debugger PIN: 234-829-993
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Run the Flask Application

→ **Check url:** http://127.0.0.1:5000/



Predict Iris Flower Species

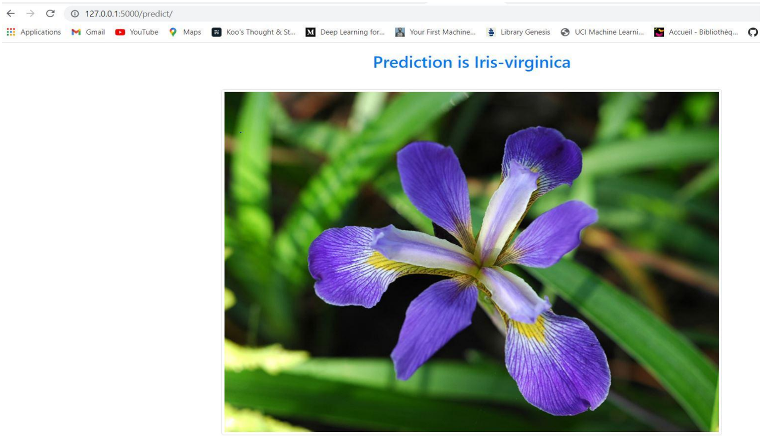
Sepal length (cm):

Sepal width (cm):

Petal length (cm):

Petal width (cm):

→ **Check url:** http://127.0.0.1:5000/predict/



Prediction is Iris-virginica

Deploy Machine Learning Model With Flask on Heroku

Create account in Heroku:

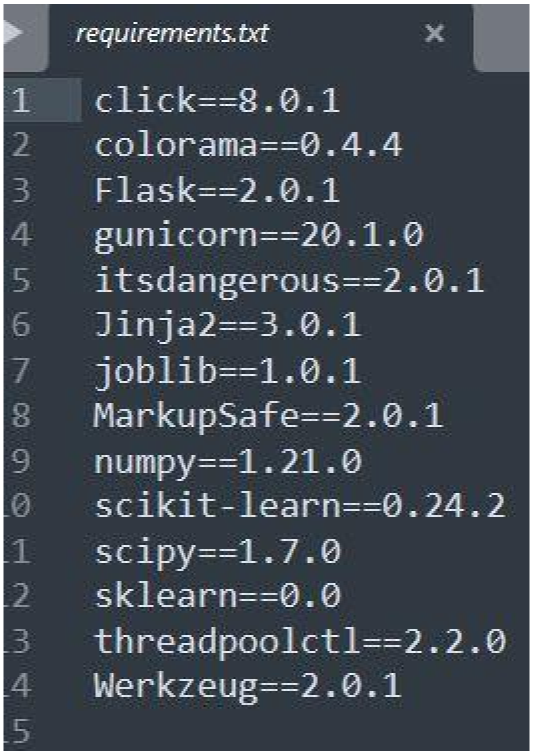
As the first step towards deployment, we need to create an account in an open source cloud platform. In our case we choose Heroku.

Install gunicorn:

Gunicorn is a Python WSGI HTTP Server for UNIX. It allows you to run any Python application concurrently by running multiple Python processes within a single dyno. The Gunicorn server is broadly compatible with various web frameworks, simply implemented, light on server resources, and fairly speedy.

Declare app dependencies:

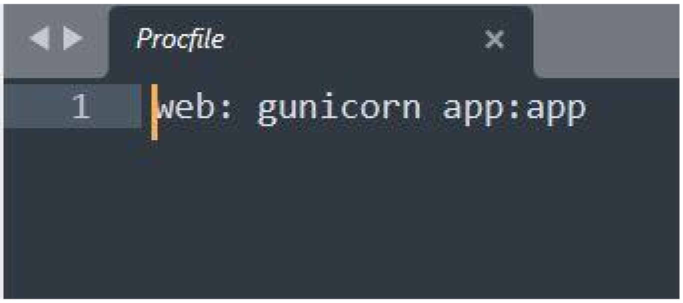
Create requirements.txt file in the root directory of the project by pip freeze command. The requirements.txt file lists all the app dependencies together. When an app is deployed, Heroku reads this file and installs the appropriate Python dependencies using the **pip install -r** command.



```
click==8.0.1
colorama==0.4.4
Flask==2.0.1
gunicorn==20.1.0
itsdangerous==2.0.1
Jinja2==3.0.1
joblib==1.0.1
MarkupSafe==2.0.1
numpy==1.21.0
scikit-learn==0.24.2
scipy==1.7.0
sklearn==0.0
threadpoolctl==2.2.0
Werkzeug==2.0.1
```

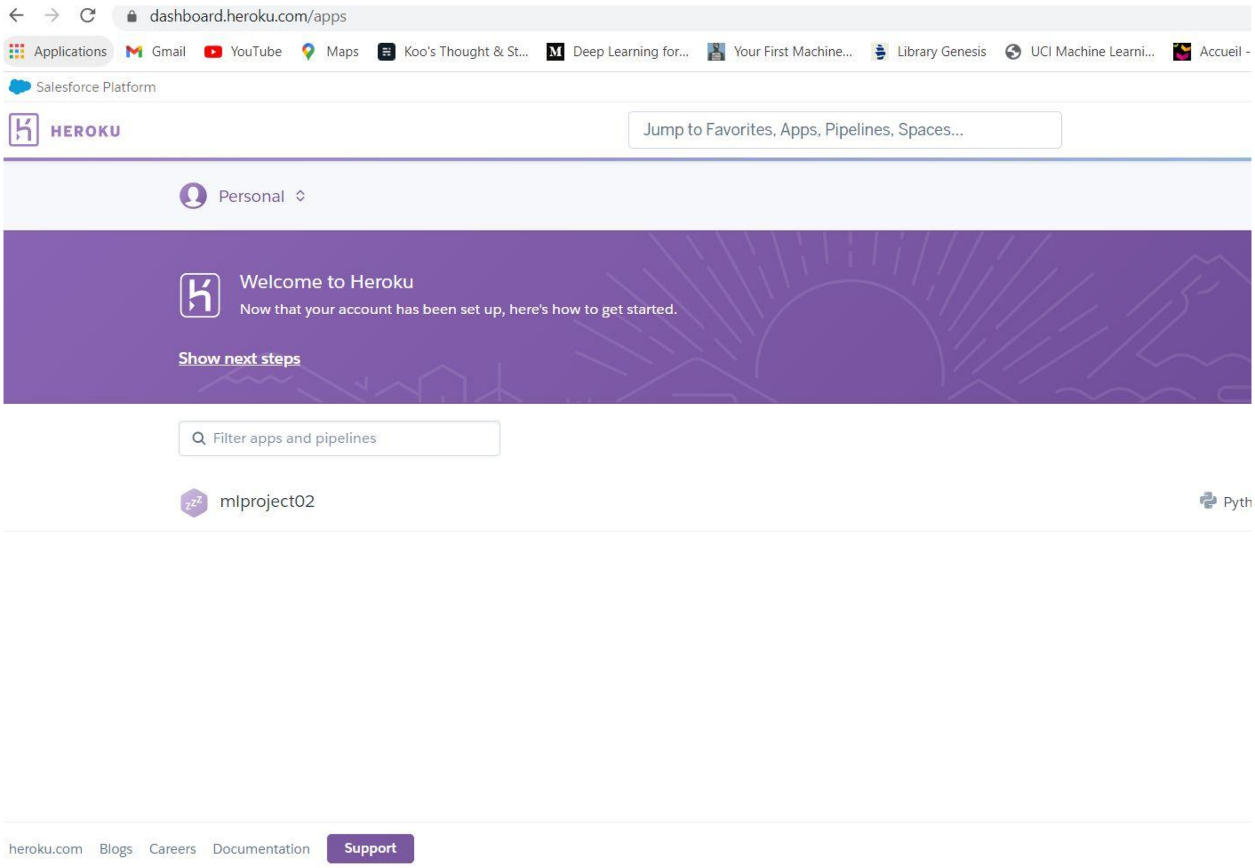
Create Procfile:

The Procfile is always a simple text file that is named Procfile in the root directory of the project, to explicitly declare what command should be executed to start our app.



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

Create Heroku App:



dashboard.heroku.com/apps

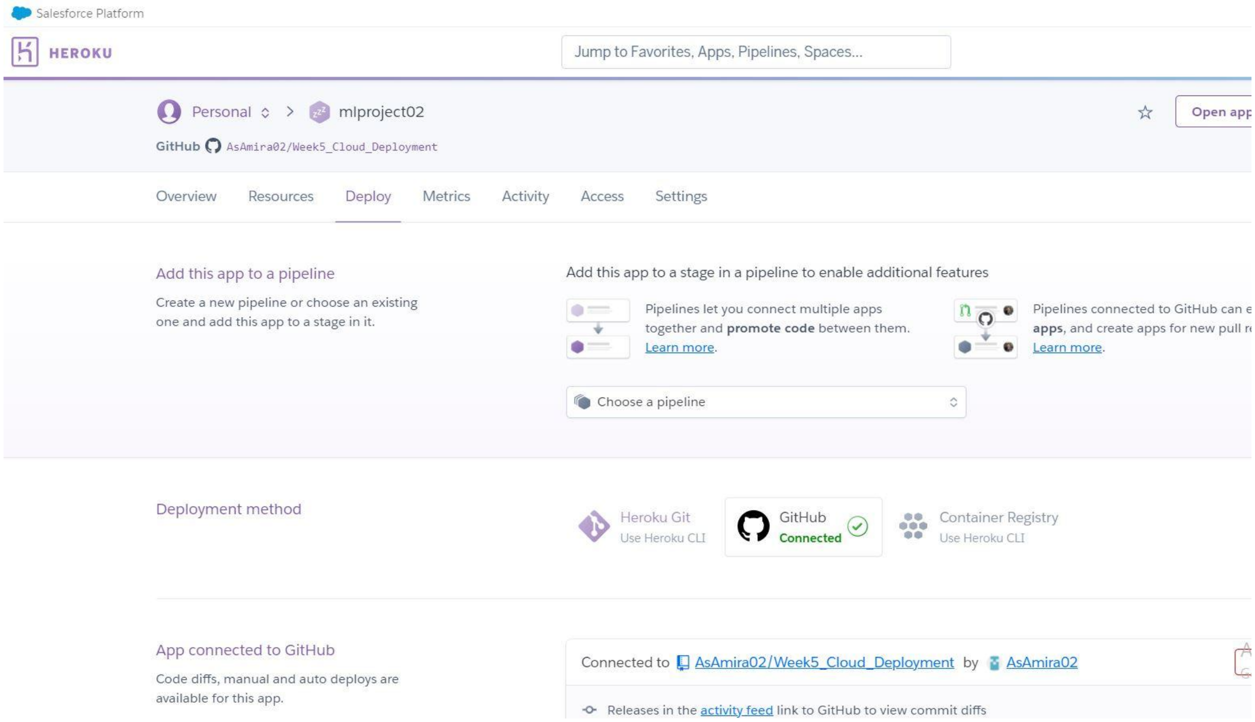
Welcome to Heroku

Now that your account has been set up, here's how to get started.

Show next steps

mlproject02

link Github account with Heroku app:



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 e **apps**, and create apps for new pull r Learn more.

Choose a pipeline

Deployment method

App connected to GitHub

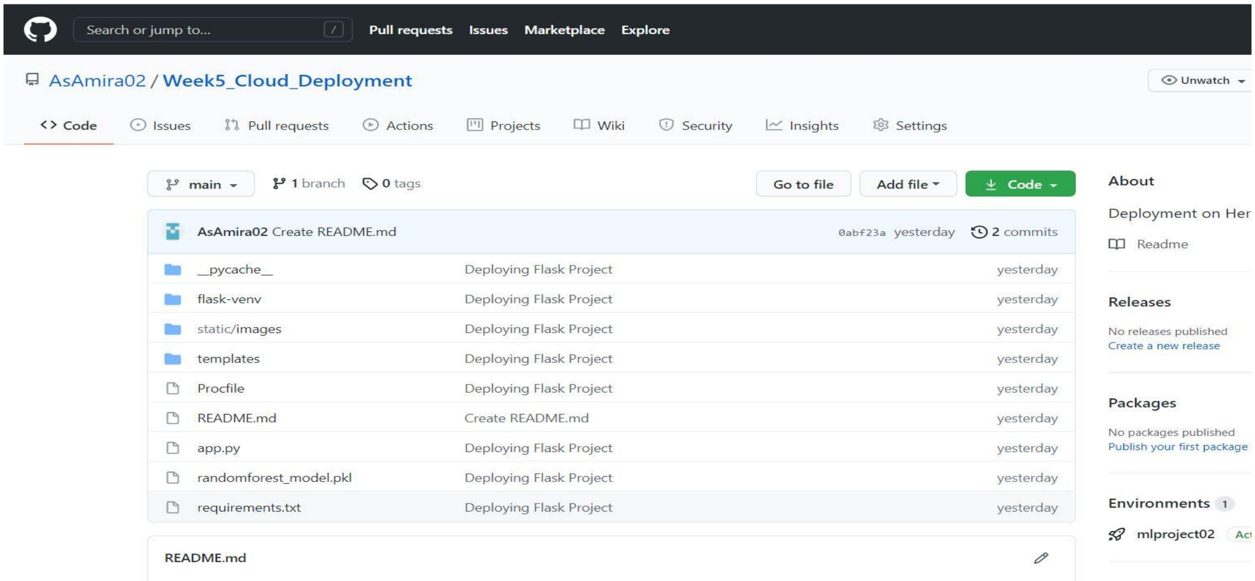
Code diffs, manual and auto deploys are available for this app.

Connected to AsAmira02/Week5_Cloud_Deployment by AsAmira02

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

Add files to the GIT repository and deploy the app:

https://github.com/AsAmira02/Week5_Cloud_Deployment



AsAmira02 / Week5_Cloud_Deployment

1 branch 0 tags

AsAmira02 Create README.md 0abf23a yesterday 2 commits

__pycache__	Deploying Flask Project	yesterday
flask-venv	Deploying Flask Project	yesterday
static/images	Deploying Flask Project	yesterday
templates	Deploying Flask Project	yesterday
Procfile	Deploying Flask Project	yesterday
README.md	Create README.md	yesterday
app.py	Deploying Flask Project	yesterday
randomforest_model.pkl	Deploying Flask Project	yesterday
requirements.txt	Deploying Flask Project	yesterday

README.md

About

Deployment on Her

Readme

Releases

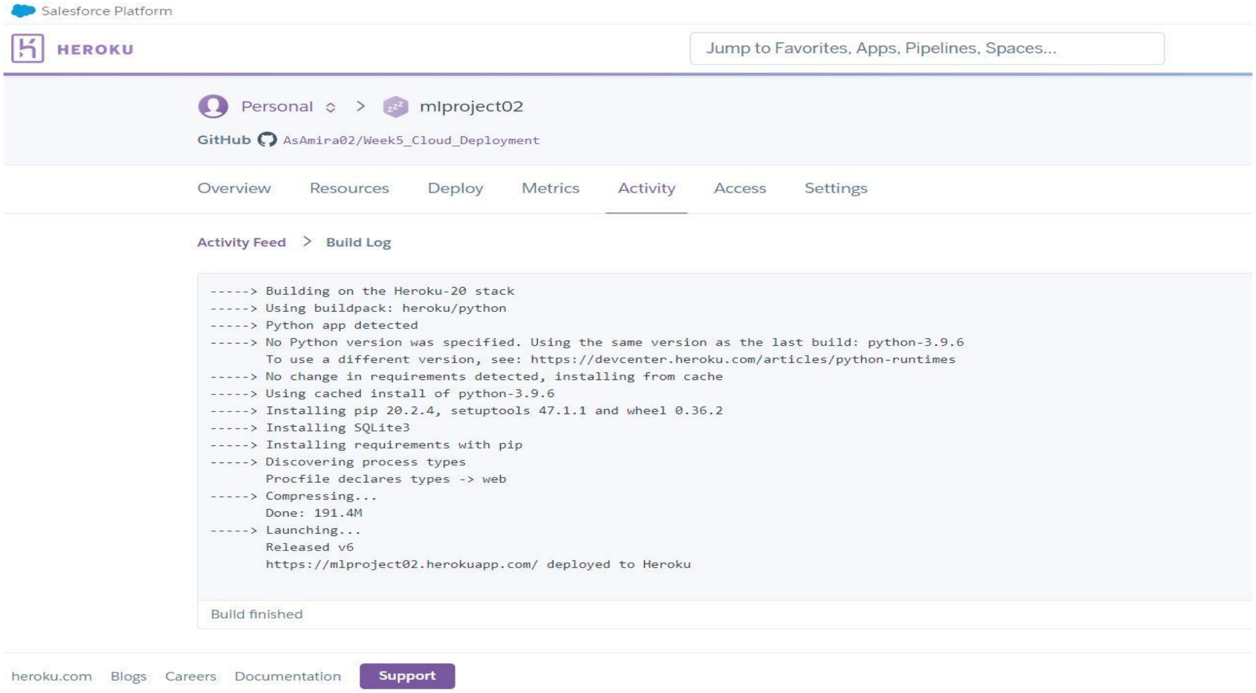
No releases published
Create a new release

Packages

No packages published
Publish your first package

Environments 1

mlproject02 Act



Personal > mlproject02

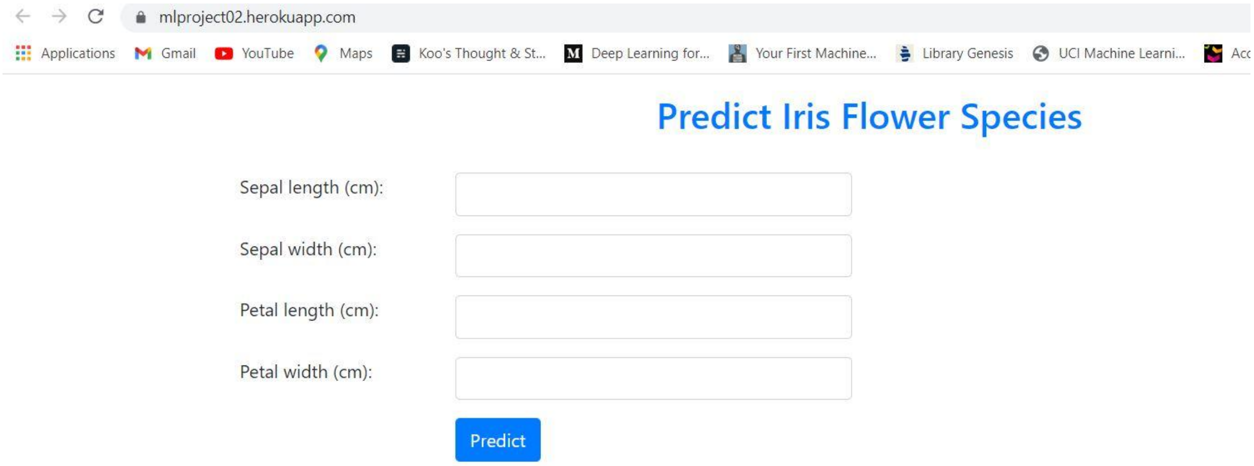
GitHub AsAmira02/Week5_Cloud_Deployment

Activity Feed > Build Log

```
-----> Building on the Heroku-20 stack
-----> Using buildpack: heroku/python
-----> Python app detected
-----> No Python version was specified. Using the same version as the last build: python-3.9.6
       To use a different version, see: https://devcenter.heroku.com/articles/python-runtimes
-----> No change in requirements detected, installing from cache
-----> Using cached install of python-3.9.6
-----> Installing pip 20.2.4, setuptools 47.1.1 and wheel 0.36.2
-----> Installing SQLite3
-----> Installing requirements with pip
-----> Discovering process types
       Procfile declares types -> web
-----> Compressing...
       Done: 191.4M
-----> Launching...
       Released v6
       https://mlproject02.herokuapp.com/ deployed to Heroku
```

Build finished

Browse deployed url:



Predict Iris Flower Species

Sepal length (cm):

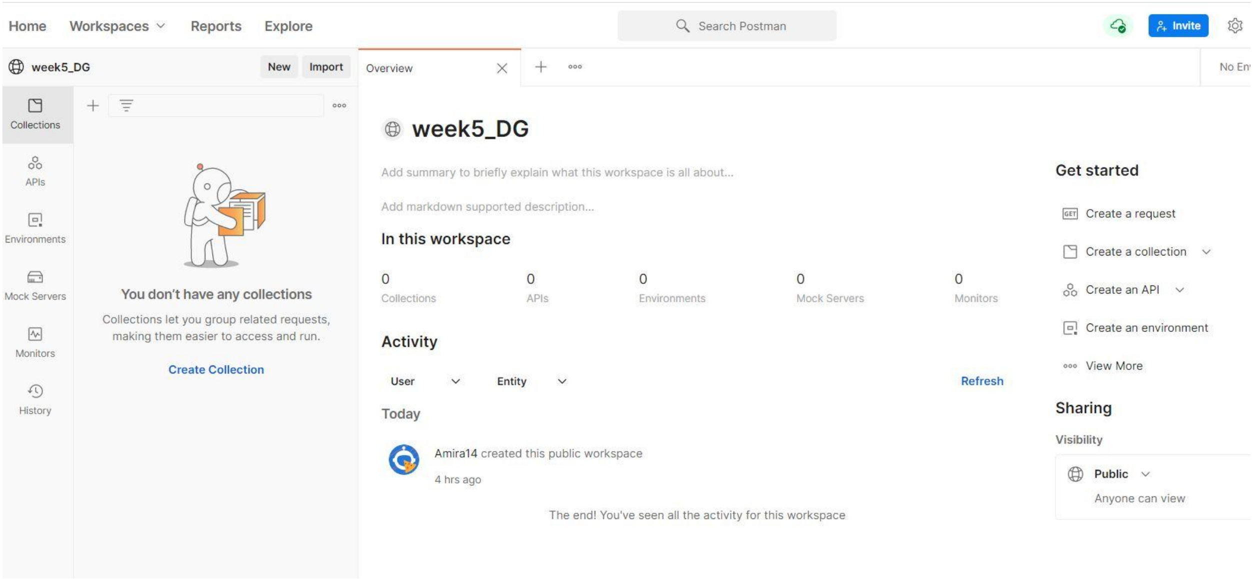
Sepal width (cm):

Petal length (cm):

Petal width (cm):

API based Deployment

For calling the application by API, I created a Postman account to make a GET request. We create a workspace named :



week5_DG

Add summary to briefly explain what this workspace is all about...

Add markdown supported description...

In this workspace

0	0	0	0	0
Collections	APIs	Environments	Mock Servers	Monitors

Activity

Today

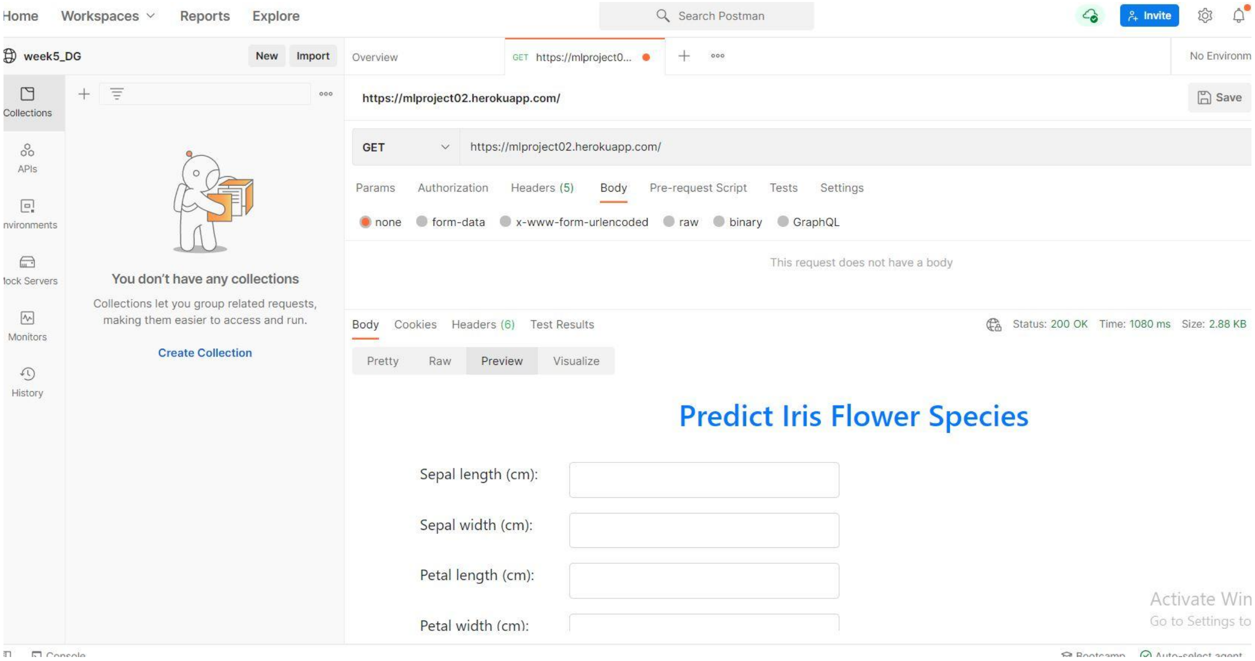
Amira14 created this public workspace
4 hrs ago

The end! You've seen all the activity for this workspace

You don't have any collections

Collections let you group related requests, making them easier to access and run.

As we can see, we get the same response:



GET https://mlproject0...

https://mlproject02.herokuapp.com/

GET https://mlproject02.herokuapp.com/

This request does not have a body

Status: 200 OK Time: 1080 ms Size: 2.88 KB

Predict Iris Flower Species

Sepal length (cm):

Sepal width (cm):

Petal length (cm):

Petal width (cm):

You don't have any collections

Collections let you group related requests, making them easier to access and run.