WEEK 5: Cloud and API Deployment

Name: Sainad Reddy Naini
Batch Code: LISUM35
Submission Date: 04/08/2024
Submitted to: Data Glacier

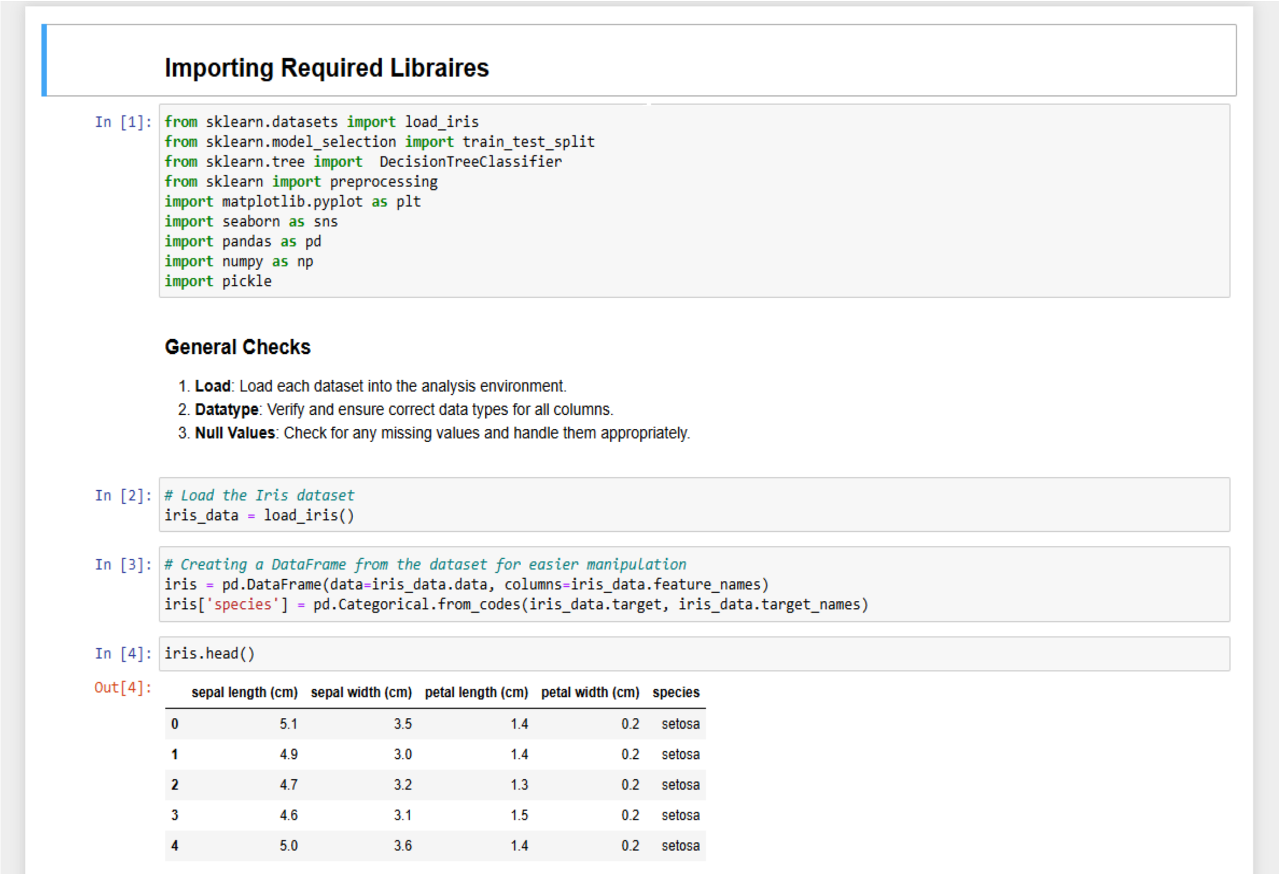
1: INTRODUCTION

In this project, we will show how to train a machine learning model using the Iris dataset, save the trained model, and then turn it into a web service with Flask. We will also explain each step of the process. The goal is to provide a clear, practical example of how to develop a machine learning model and make it available for use in a real-world application.

2: DATASET DESCRIPTION

In this part, we will explore the Iris dataset and look for relationships between its different features. The Iris dataset is popular in the machine learning world. It has 150 samples of iris flowers, each with four features: sepal length, sepal width, petal length, and petal width. The samples are divided into three types of iris flowers: Iris-setosa, Iris-versicolor, and Iris-virginica, with 50 samples of each type.

Code:



Importing Required Libraires

In [1]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import  DecisionTreeClassifier
from sklearn import preprocessing
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import pickle
```

General Checks

1. **Load**: Load each dataset into the analysis environment.
2. **Datatype**: Verify and ensure correct data types for all columns.
3. **Null Values**: Check for any missing values and handle them appropriately.

In [2]:
```
# Load the Iris dataset
iris_data = load_iris()
```

In [3]:
```
# Creating a DataFrame from the dataset for easier manipulation
iris = pd.DataFrame(data=iris_data.data, columns=iris_data.feature_names)
iris['species'] = pd.Categorical.from_codes(iris_data.target, iris_data.target_names)
```

In [4]:
```
iris.head()
```

Out[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
In [6]: #Checking dattypes
        iris.dtypes
```

```
Out[6]: sepal length (cm)     float64
        sepal width (cm)      float64
        petal length (cm)     float64
        petal width (cm)      float64
        species              category
        dtype: object
```

```
In [7]: #Sum of Null_values
        iris.isnull().sum()
```

```
Out[7]: sepal length (cm)    0
        sepal width (cm)     0
        petal length (cm)    0
        petal width (cm)     0
        species              0
        dtype: int64
```

```
In [8]: iris.info()
```

```
        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 150 entries, 0 to 149
        Data columns (total 5 columns):
         #   Column             Non-Null Count  Dtype
        ---  ------             --------------  -----
         0   sepal length (cm)  150 non-null    float64
         1   sepal width (cm)   150 non-null    float64
         2   petal length (cm)  150 non-null    float64
         3   petal width (cm)   150 non-null    float64
         4   species            150 non-null    category
        dtypes: category(1), float64(4)
        memory usage: 5.1 KB
```

```
In [9]: #count of each species
        iris.species.value_counts()
```

```
Out[9]: species
        setosa        50
        versicolor    50
        virginica     50
        Name: count, dtype: int64
```

```
In [10]: iris['species'].unique()
```

```
Out[10]: ['setosa', 'versicolor', 'virginica']
         Categories (3, object): ['setosa', 'versicolor', 'virginica']
```

Checking for Relationships (Linear or Non-Linear)

- From the correlation plot, we can observe strong linear relationships between petal length, petal width, and sepal length. However, the relationships involving sepal width are weaker. Therefore, we can conclude that the dataset exhibits strong linear relationships among certain features, while some features have weaker or less clear linear associations.

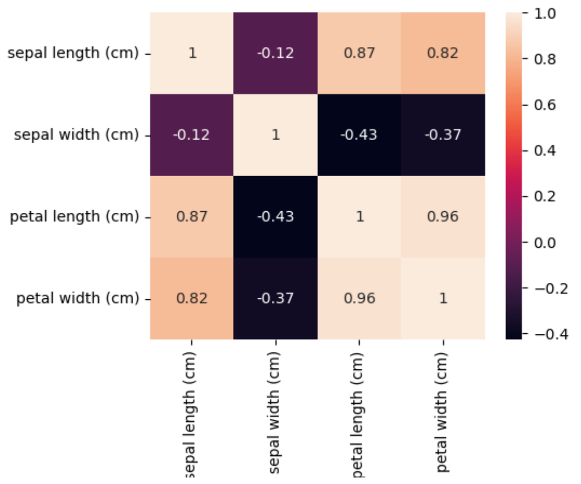
```
In [11]: iris.drop(['species'],axis=1).corr()
```

Out[11]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
sepal length (cm)	1.000000	-0.117570	0.871754	0.817941
sepal width (cm)	-0.117570	1.000000	-0.428440	-0.366126
petal length (cm)	0.871754	-0.428440	1.000000	0.962865
petal width (cm)	0.817941	-0.366126	0.962865	1.000000

```
In [12]: fig,ax=plt.subplots(figsize=(5,4))
         sns.heatmap(iris.drop(['species'],axis=1).corr(),annot=True,ax=ax)
```

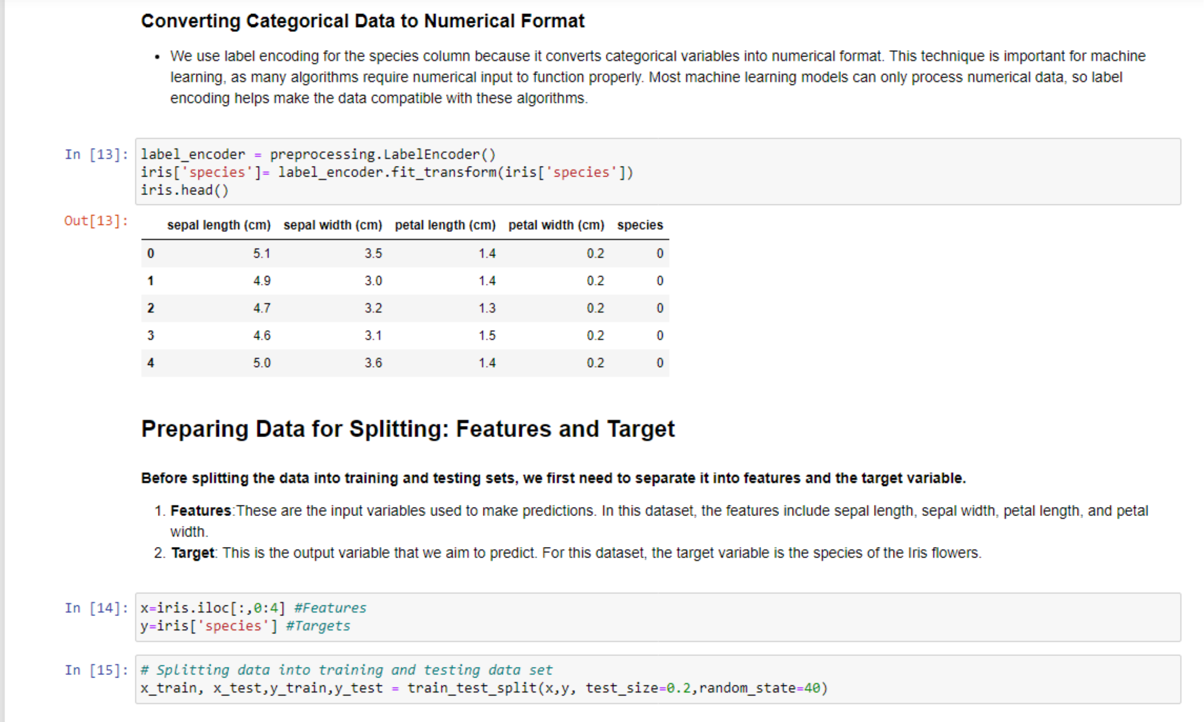
```
Out[12]: <Axes: >
```



3: MODEL TRAINING AND SAVING

In this step, we will split the data into training and testing sets, train a Decision Tree model on the training data, and save the trained model using Pickle.

Code:



Converting Categorical Data to Numerical Format

- We use label encoding for the species column because it converts categorical variables into numerical format. This technique is important for machine learning, as many algorithms require numerical input to function properly. Most machine learning models can only process numerical data, so label encoding helps make the data compatible with these algorithms.

In [13]:

```
label_encoder = preprocessing.LabelEncoder()
iris['species']= label_encoder.fit_transform(iris['species'])
iris.head()
```

Out[13]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Preparing Data for Splitting: Features and Target

Before splitting the data into training and testing sets, we first need to separate it into features and the target variable.

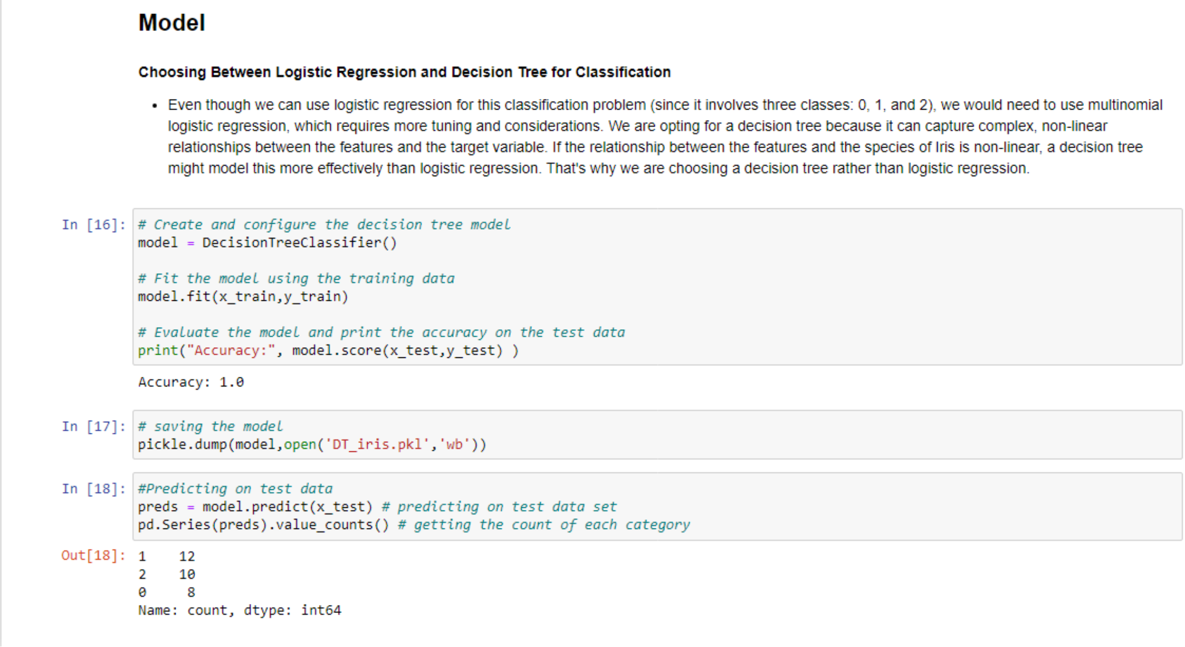
1. **Features**:These are the input variables used to make predictions. In this dataset, the features include sepal length, sepal width, petal length, and petal width.
2. **Target**: This is the output variable that we aim to predict. For this dataset, the target variable is the species of the Iris flowers.

In [14]:

```
x=iris.iloc[:,0:4] #Features
y=iris['species'] #Targets
```

In [15]:

```
# Splitting data into training and testing data set
x_train, x_test,y_train,y_test = train_test_split(x,y, test_size=0.2,random_state=40)
```



Model

Choosing Between Logistic Regression and Decision Tree for Classification

- Even though we can use logistic regression for this classification problem (since it involves three classes: 0, 1, and 2), we would need to use multinomial logistic regression, which requires more tuning and considerations. We are opting for a decision tree because it can capture complex, non-linear relationships between the features and the target variable. If the relationship between the features and the species of Iris is non-linear, a decision tree might model this more effectively than logistic regression. That's why we are choosing a decision tree rather than logistic regression.

In [16]:

```
# Create and configure the decision tree model
model = DecisionTreeClassifier()

# Fit the model using the training data
model.fit(x_train,y_train)

# Evaluate the model and print the accuracy on the test data
print("Accuracy:", model.score(x_test,y_test) )
```

```
Accuracy: 1.0
```

In [17]:

```
# saving the model
pickle.dump(model,open('DT_iris.pkl','wb'))
```

In [18]:

```
#Predicting on test data
preds = model.predict(x_test) # predicting on test data set
pd.Series(preds).value_counts() # getting the count of each category
```

Out[18]:

```
1    12
2    10
0     8
Name: count, dtype: int64
```

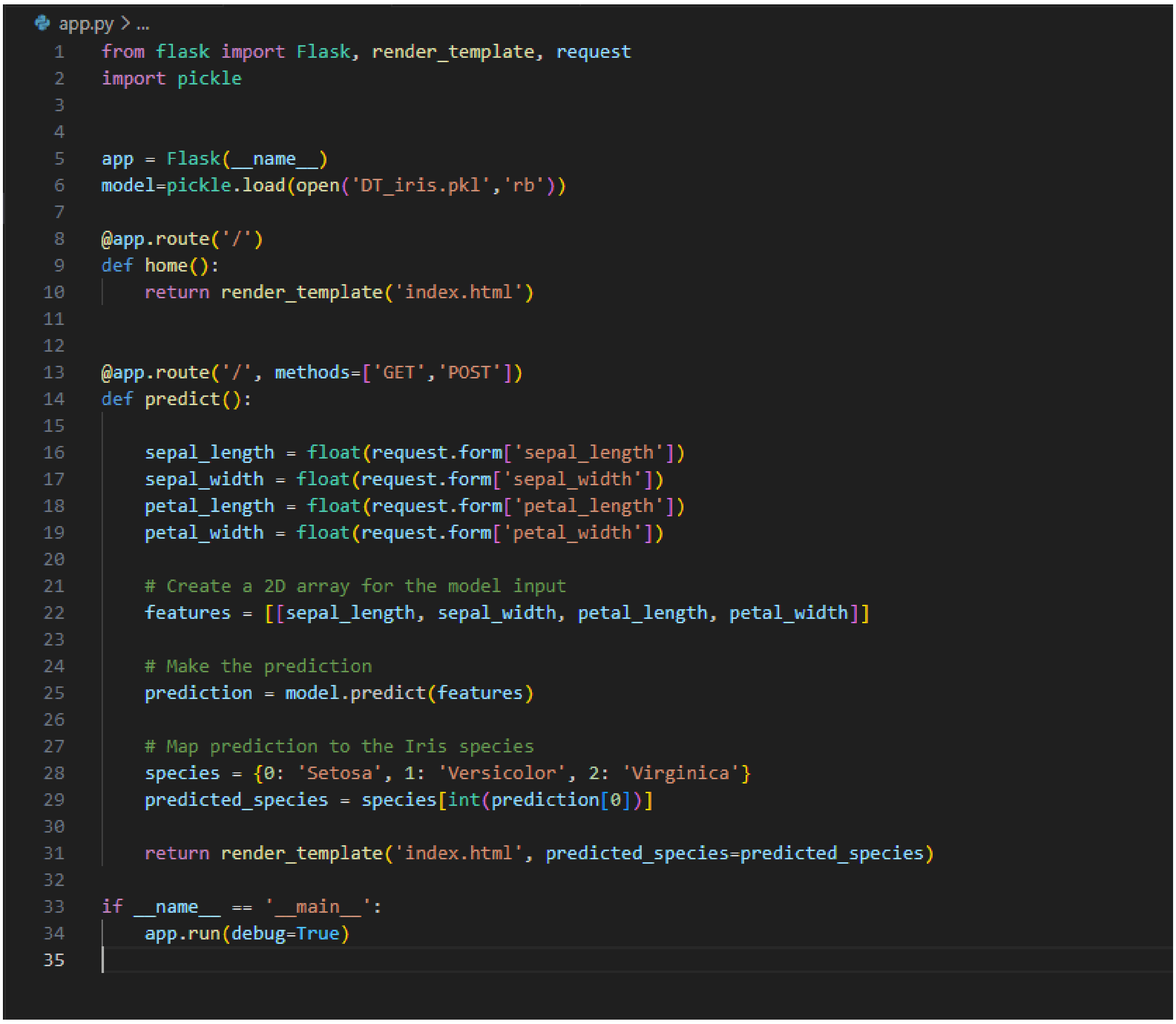
The model was trained on 80% of the dataset and tested on the remaining 20%.

4: FLASK WEB APPLICATION FOR MODEL DEPLOYMENT

To make the trained model available as a web service, we create a Flask web application. This app loads the saved model and provides an API endpoint for making predictions. We use Visual Studio to develop and run the Flask app, as it includes a built-in live server that allows us to host the web service without needing additional tools.

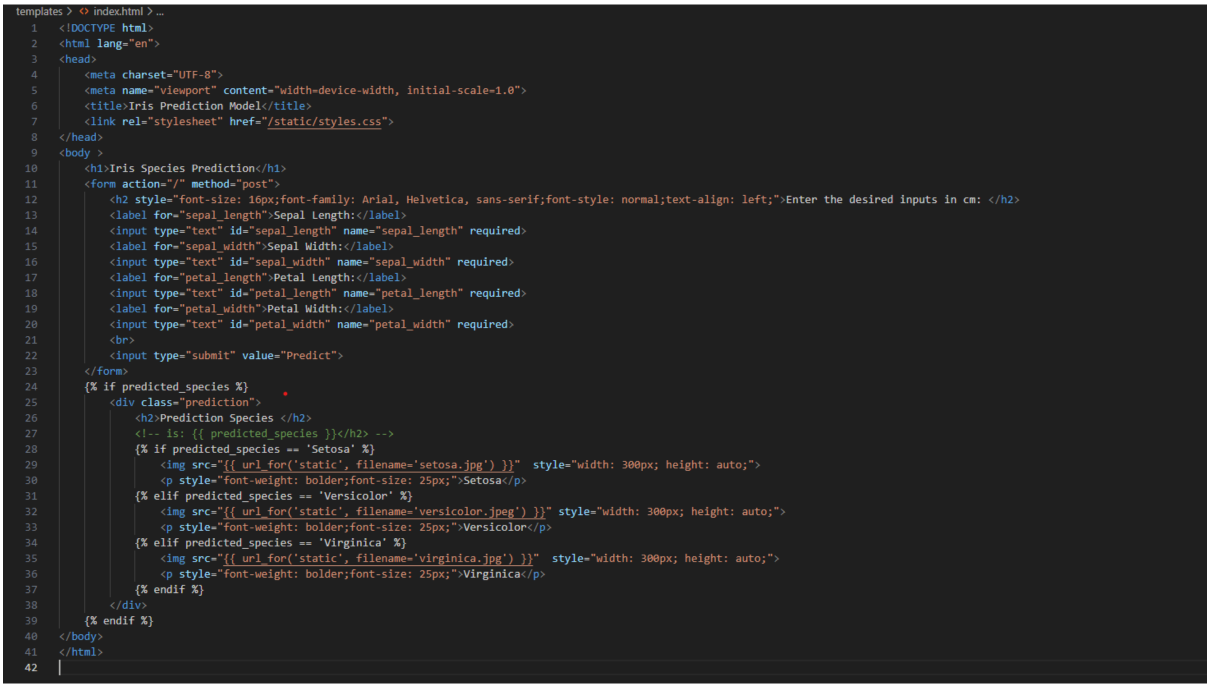
Code:

Here is the main Flask app code where everything happens when you run it. This app uses HTML and CSS files, allowing you to enter inputs and see the outputs.



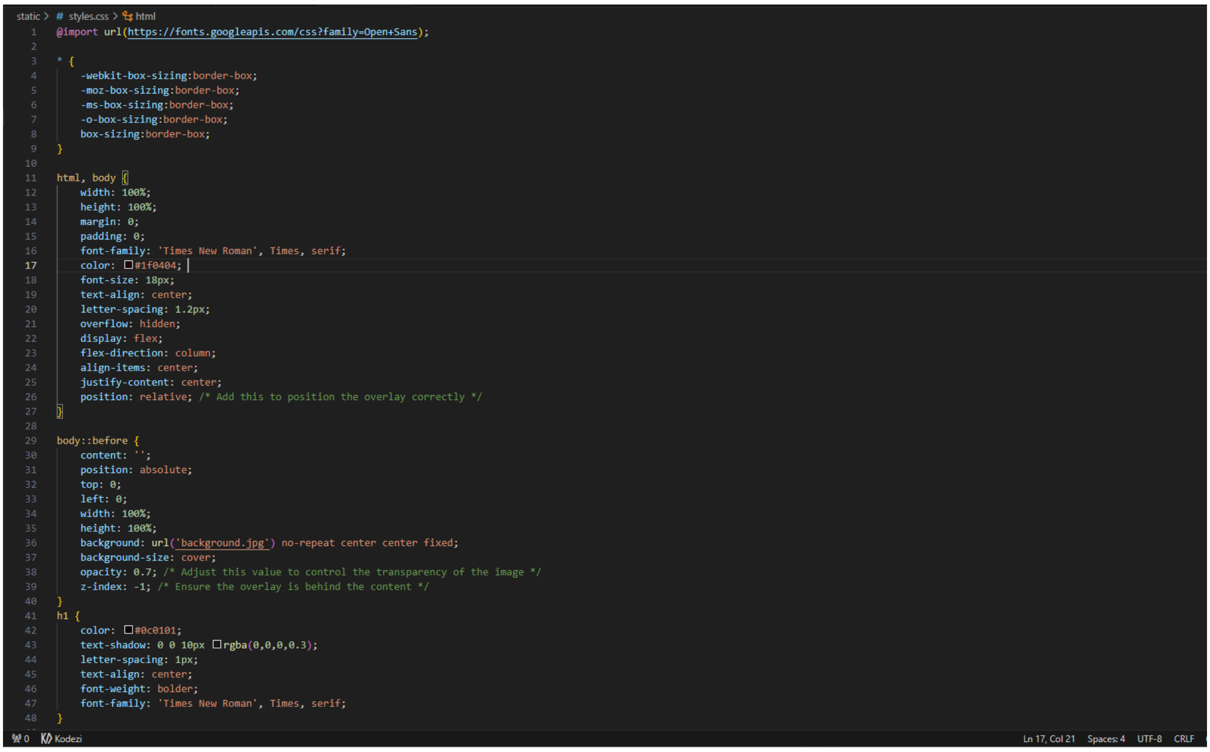
```python
from flask import Flask, render_template, request
import pickle


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

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


@app.route('/', methods=['GET','POST'])
def predict():

    sepal_length = float(request.form['sepal_length'])
    sepal_width = float(request.form['sepal_width'])
    petal_length = float(request.form['petal_length'])
    petal_width = float(request.form['petal_width'])

    # Create a 2D array for the model input
    features = [[sepal_length, sepal_width, petal_length, petal_width]]

    # Make the prediction
    prediction = model.predict(features)

    # Map prediction to the Iris species
    species = {0: 'Setosa', 1: 'Versicolor', 2: 'Virginica'}
    predicted_species = species[int(prediction[0])]

    return render_template('index.html', predicted_species=predicted_species)

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

```

Html code: for front end

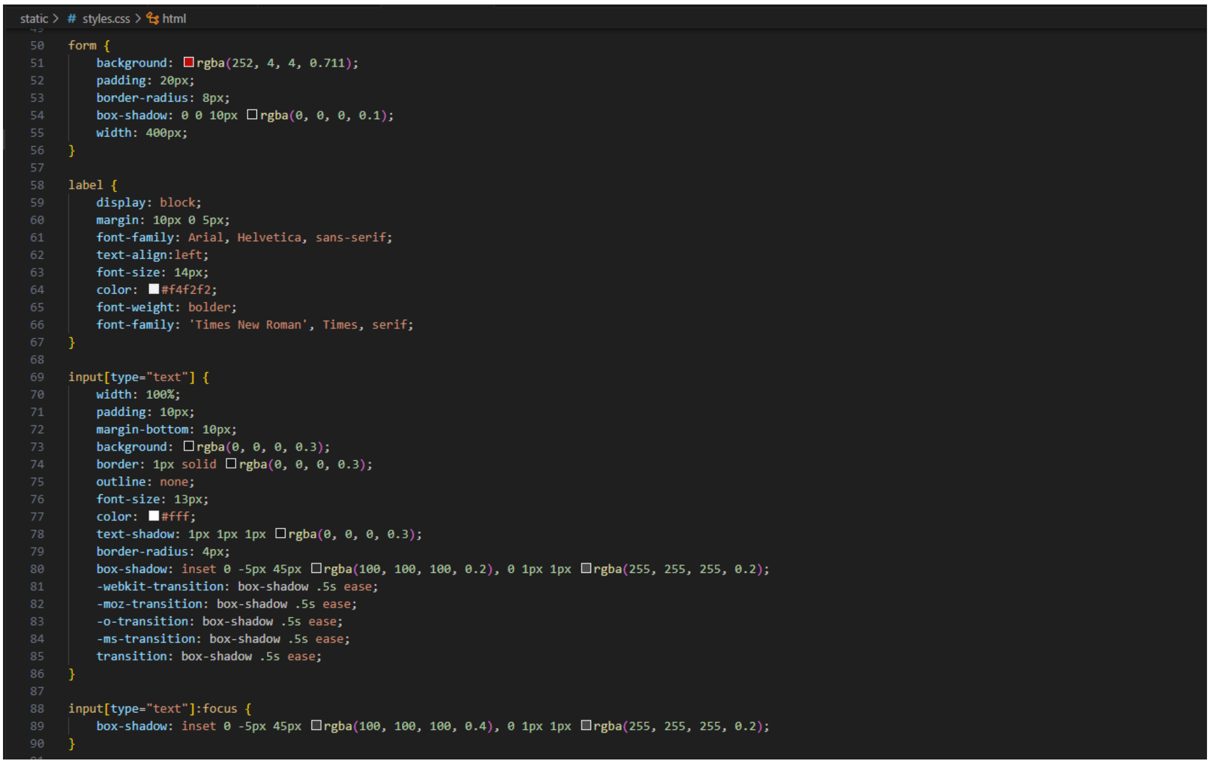


```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Iris Prediction Model</title>
    <link rel="stylesheet" href="/static/styles.css">
</head>
<body >
    <h1>Iris Species Prediction</h1>
    <form action="/" method="post">
        <h2 style="font-size: 16px;font-family: Arial, Helvetica, sans-serif;font-style: normal;text-align: left;">Enter the desired inputs in cm: </h2>
        <label for="sepal_length">Sepal Length:</label>
        <input type="text" id="sepal_length" name="sepal_length" required>
        <label for="sepal_width">Sepal Width:</label>
        <input type="text" id="sepal_width" name="sepal_width" required>
        <label for="petal_length">Petal Length:</label>
        <input type="text" id="petal_length" name="petal_length" required>
        <label for="petal_width">Petal Width:</label>
        <input type="text" id="petal_width" name="petal_width" required>
        <br>
        <input type="submit" value="Predict">
    </form>
    {% if predicted_species %}
        <div class="prediction">
            <h2>Prediction Species </h2>
            <!-- is: {{ predicted_species }}</h2> -->
            {% if predicted_species == 'Setosa' %}
                <img src="{{ url_for('static', filename='setosa.jpg') }}"  style="width: 300px; height: auto;">
                <p style="font-weight: bolder;font-size: 25px;">Setosa</p>
            {% elif predicted_species == 'Versicolor' %}
                <img src="{{ url_for('static', filename='versicolor.jpeg') }}" style="width: 300px; height: auto;">
                <p style="font-weight: bolder;font-size: 25px;">Versicolor</p>
            {% elif predicted_species == 'Virginica' %}
                <img src="{{ url_for('static', filename='virginica.jpg') }}"  style="width: 300px; height: auto;">
                <p style="font-weight: bolder;font-size: 25px;">Virginica</p>
            {% endif %}
        </div>
    {% endif %}
</body>
</html>
```

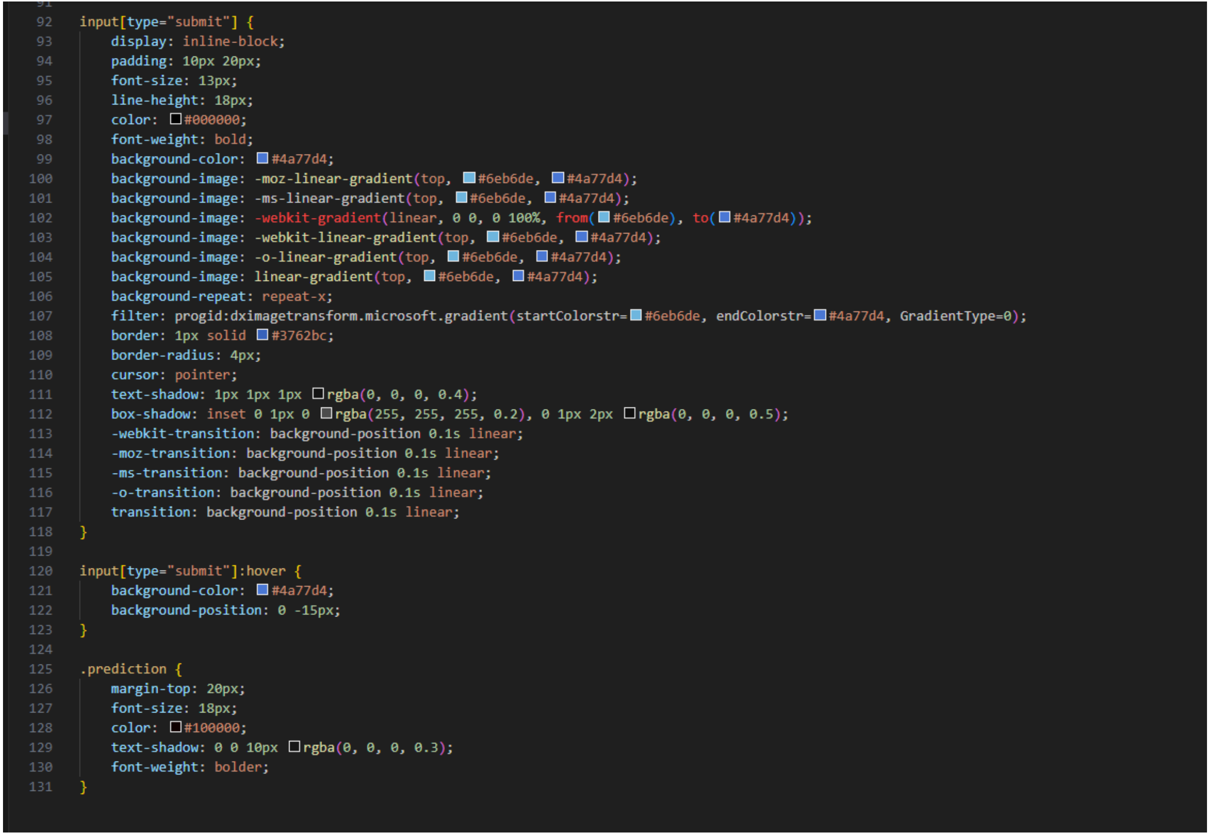
CSS code: we use this for styling the html page for front end



```css
@import url(https://fonts.googleapis.com/css?family=Open+Sans);

* {
    -webkit-box-sizing:border-box;
    -moz-box-sizing:border-box;
    -ms-box-sizing:border-box;
    -o-box-sizing:border-box;
    box-sizing:border-box;
}

html, body {
    width: 100%;
    height: 100%;
    margin: 0;
    padding: 0;
    font-family: 'Times New Roman', Times, serif;
    color: #1f0404;
    font-size: 18px;
    text-align: center;
    letter-spacing: 1.2px;
    overflow: hidden;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    position: relative; /* Add this to position the overlay correctly */
}

body::before {
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: url('background.jpg') no-repeat center center fixed;
    background-size: cover;
    opacity: 0.7; /* Adjust this value to control the transparency of the image */
    z-index: -1; /* Ensure the overlay is behind the content */
}
h1 {
    color: #0c0101;
    text-shadow: 0 0 10px rgba(0,0,0,0.3);
    letter-spacing: 1px;
    text-align: center;
    font-weight: bolder;
    font-family: 'Times New Roman', Times, serif;
}
```



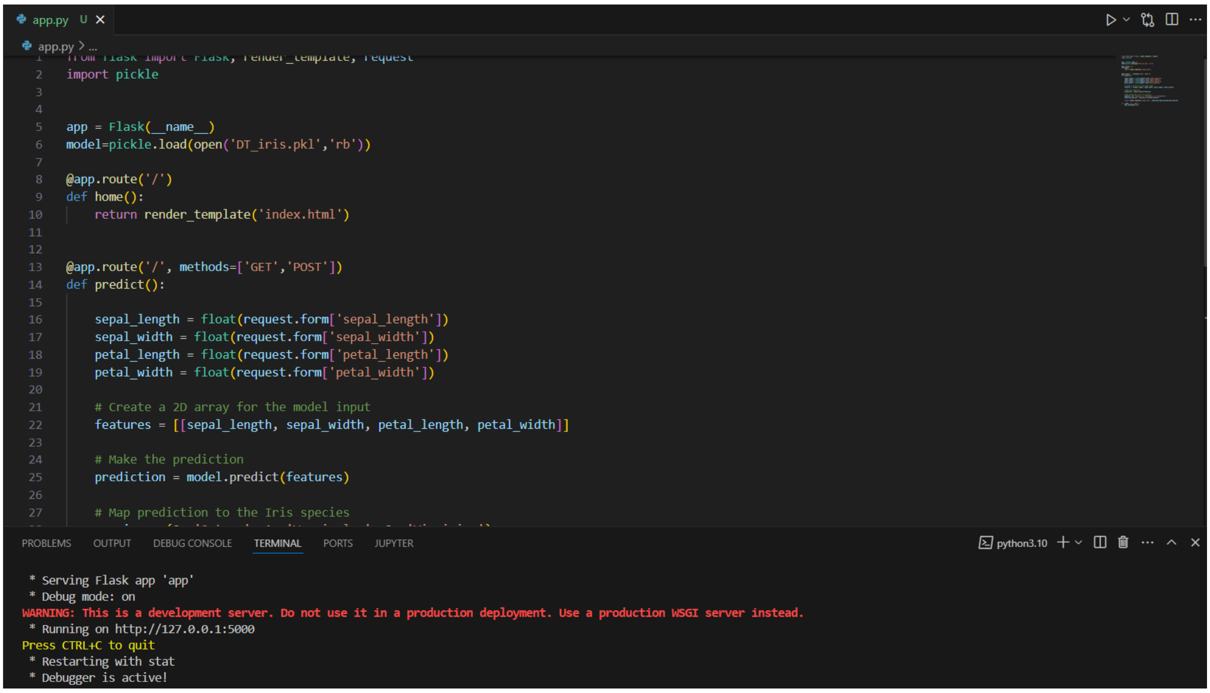
```css
form {
    background: rgba(252, 4, 4, 0.711);
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    width: 400px;
}

label {
    display: block;
    margin: 10px 0 5px;
    font-family: Arial, Helvetica, sans-serif;
    text-align:left;
    font-size: 14px;
    color: #f4f2f2;
    font-weight: bolder;
    font-family: 'Times New Roman', Times, serif;
}

input[type="text"] {
    width: 100%;
    padding: 10px;
    margin-bottom: 10px;
    background: rgba(0, 0, 0, 0.3);
    border: 1px solid rgba(0, 0, 0, 0.3);
    outline: none;
    font-size: 13px;
    color: #fff;
    text-shadow: 1px 1px 1px rgba(0, 0, 0, 0.3);
    border-radius: 4px;
    box-shadow: inset 0 -5px 45px rgba(100, 100, 100, 0.2), 0 1px 1px rgba(255, 255, 255, 0.2);
    -webkit-transition: box-shadow .5s ease;
    -moz-transition: box-shadow .5s ease;
    -o-transition: box-shadow .5s ease;
    -ms-transition: box-shadow .5s ease;
    transition: box-shadow .5s ease;
}

input[type="text"]:focus {
    box-shadow: inset 0 -5px 45px rgba(100, 100, 100, 0.4), 0 1px 1px rgba(255, 255, 255, 0.2);
}
```



```css
input[type="submit"] {
    display: inline-block;
    padding: 10px 20px;
    font-size: 13px;
    line-height: 18px;
    color: #000000;
    font-weight: bold;
    background-color: #4a77d4;
    background-image: -moz-linear-gradient(top, #6eb6de, #4a77d4);
    background-image: -ms-linear-gradient(top, #6eb6de, #4a77d4);
    background-image: -webkit-gradient(linear, 0 0, 0 100%, from(#6eb6de), to(#4a77d4));
    background-image: -webkit-linear-gradient(top, #6eb6de, #4a77d4);
    background-image: -o-linear-gradient(top, #6eb6de, #4a77d4);
    background-image: linear-gradient(top, #6eb6de, #4a77d4);
    background-repeat: repeat-x;
    filter: progid:dximagetransform.microsoft.gradient(startColorstr=#6eb6de, endColorstr=#4a77d4, GradientType=0);
    border: 1px solid #3762bc;
    border-radius: 4px;
    cursor: pointer;
    text-shadow: 1px 1px 1px rgba(0, 0, 0, 0.4);
    box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.2), 0 1px 2px rgba(0, 0, 0, 0.5);
    -webkit-transition: background-position 0.1s linear;
    -moz-transition: background-position 0.1s linear;
    -ms-transition: background-position 0.1s linear;
    -o-transition: background-position 0.1s linear;
    transition: background-position 0.1s linear;
}

input[type="submit"]:hover {
    background-color: #4a77d4;
    background-position: 0 -15px;
}

.prediction {
    margin-top: 20px;
    font-size: 18px;
    color: #100000;
    text-shadow: 0 0 10px rgba(0, 0, 0, 0.3);
    font-weight: bolder;
}
```

5: RUNNING THE APPLICATION (Local -in visual studio)

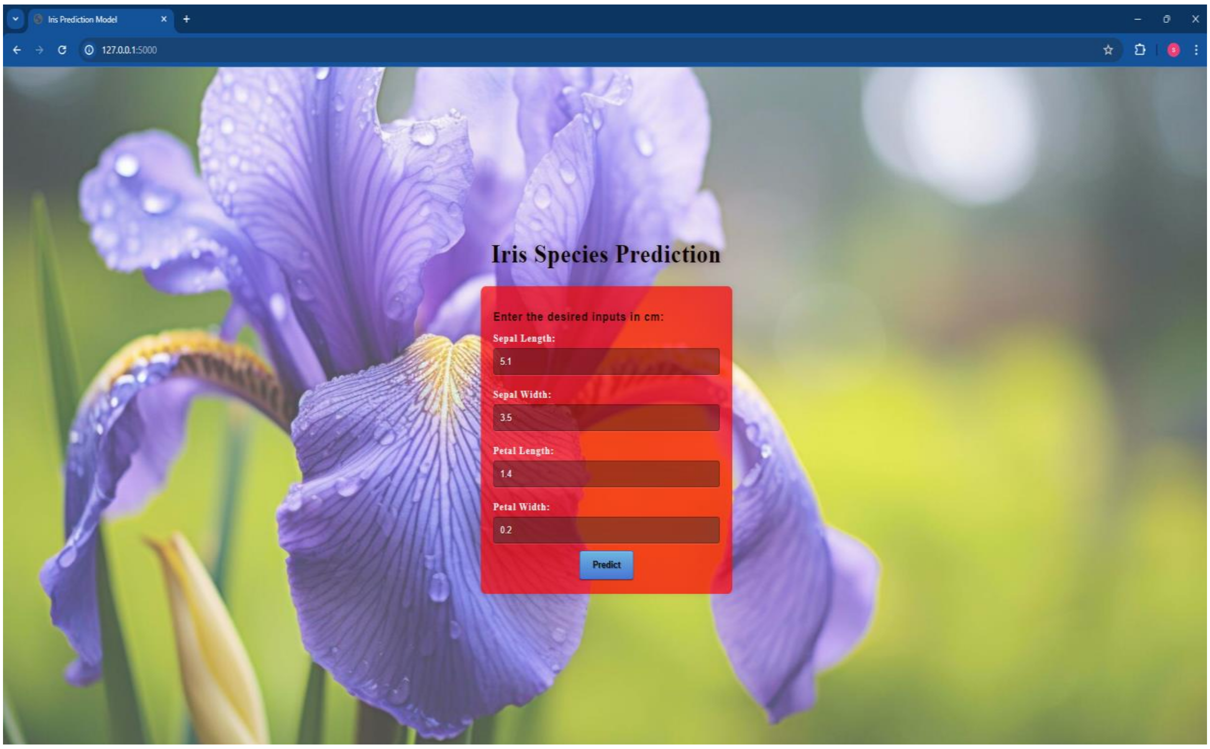
First, run the app.py file (the Flask app) using the terminal with the command python app.py. Then, paste the returned server address into your web browser, typically http://127.0.0.1:5000.



```python
from flask import Flask, render_template, request
import pickle


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

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


@app.route('/', methods=['GET','POST'])
def predict():

    sepal_length = float(request.form['sepal_length'])
    sepal_width = float(request.form['sepal_width'])
    petal_length = float(request.form['petal_length'])
    petal_width = float(request.form['petal_width'])

    # Create a 2D array for the model input
    features = [[sepal_length, sepal_width, petal_length, petal_width]]

    # Make the prediction
    prediction = model.predict(features)

    # Map prediction to the Iris species
```

```
 * 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://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
```

Once you paste the server address into your browser, enter the required inputs in centimetres on the web page



Iris Species Prediction

Enter the desired inputs in cm:

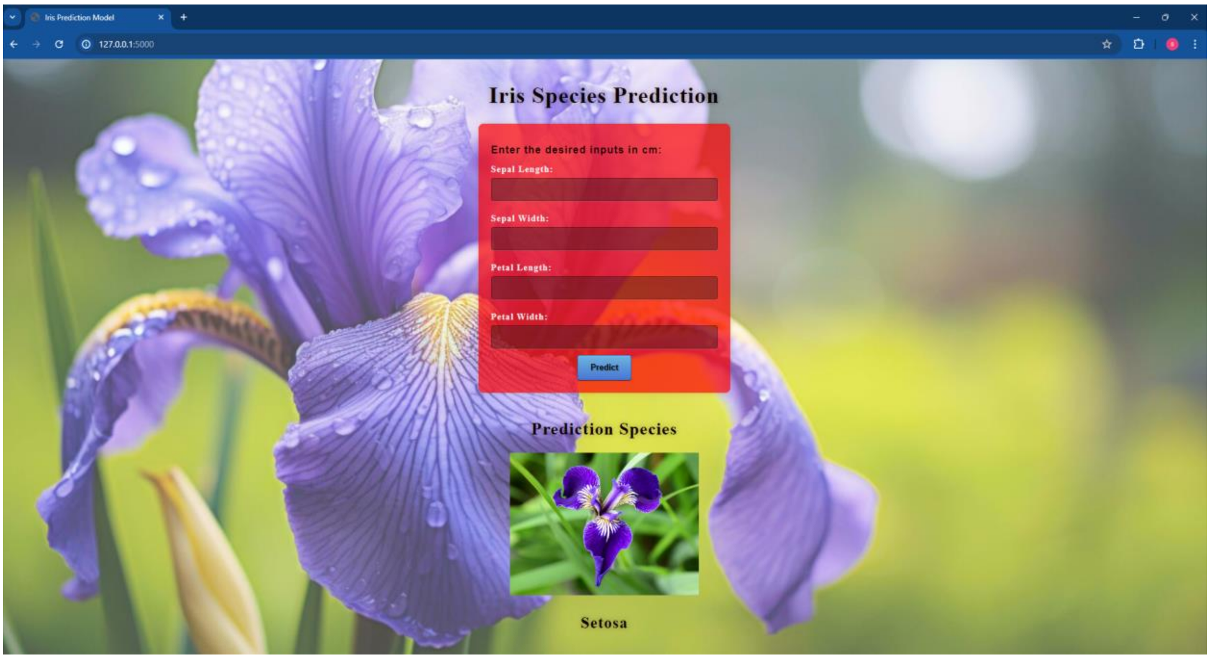
Sepal Length: 5.1

Sepal Width: 3.5

Petal Length: 1.4

Petal Width: 0.2

After entering the inputs, click the "Predict" button. This will provide you with a prediction of the iris species, along with an image of that species.

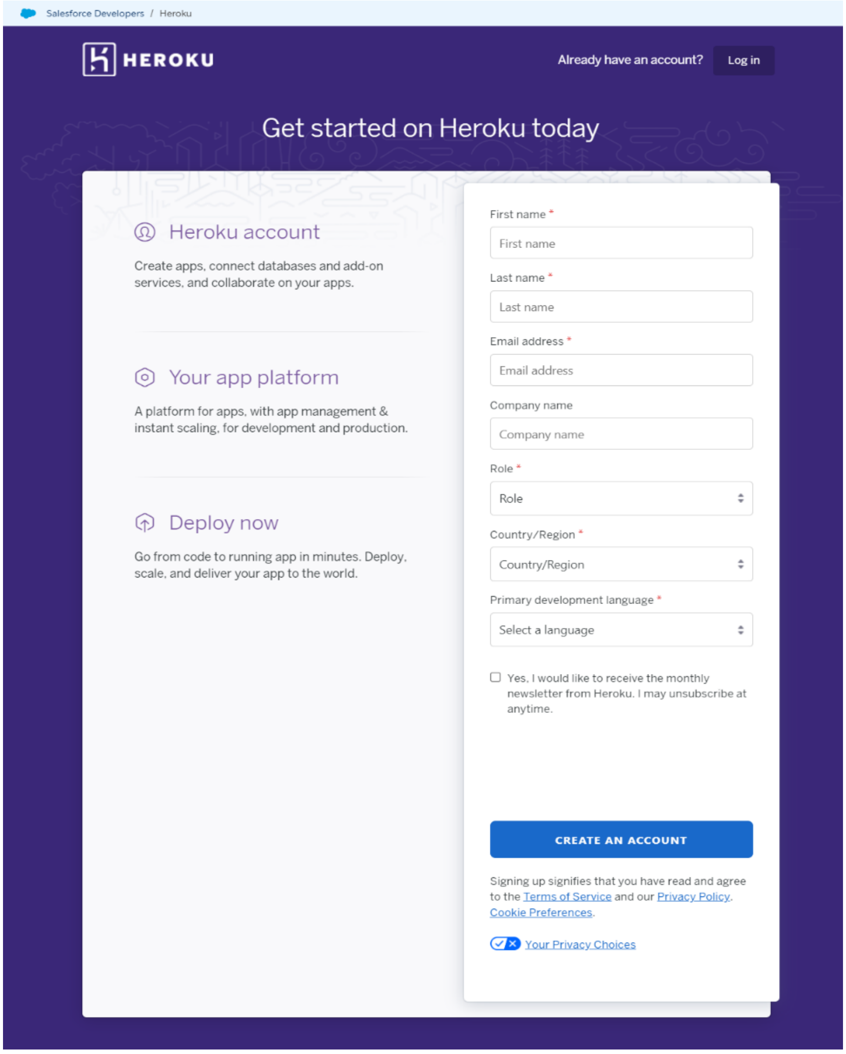


6.DEPLOYMENT ON CLOUD (HEROKU)

Before deploying to Heroku, there are two methods: using Heroku CLI (Heroku git) or directly connecting the GitHub repository on the Heroku platform. Here, we use the second method.

6.1: Creating the account

Create a Heroku account by filling in all the required details.



Salesforce Developers / Heroku

HEROKU

Already have an account? Log in

Get started on Heroku today

Heroku account

Create apps, connect databases and add-on services, and collaborate on your apps.

Your app platform

A platform for apps, with app management & instant scaling, for development and production.

Deploy now

Go from code to running app in minutes. Deploy, scale, and deliver your app to the world.

First name *

First name

Last name *

Last name

Email address *

Email address

Company name

Company name

Role *

Role

Country/Region *

Country/Region

Primary development language *

Select a language

□ Yes, I would like to receive the monthly newsletter from Heroku. I may unsubscribe at anytime.

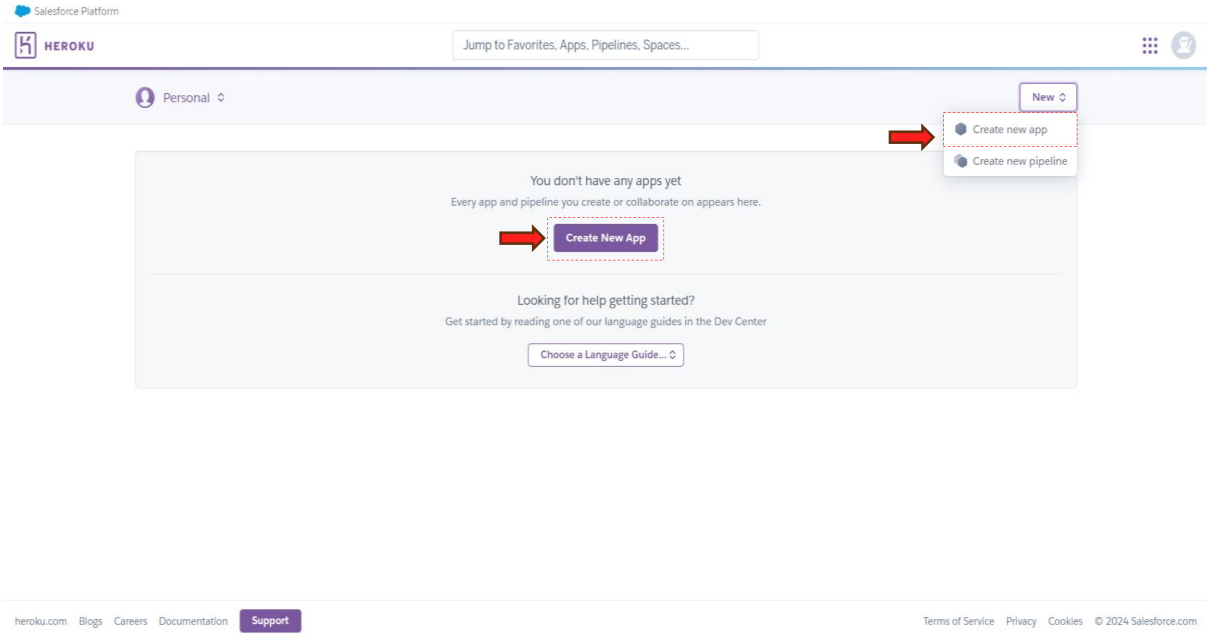
CREATE AN ACCOUNT

Signing up signifies that you have read and agree to the Terms of Service and our Privacy Policy. Cookie Preferences.

Your Privacy Choices

6.2 Creating an App

Once you log in, select "New" > "Create New App" or click "Create New App".



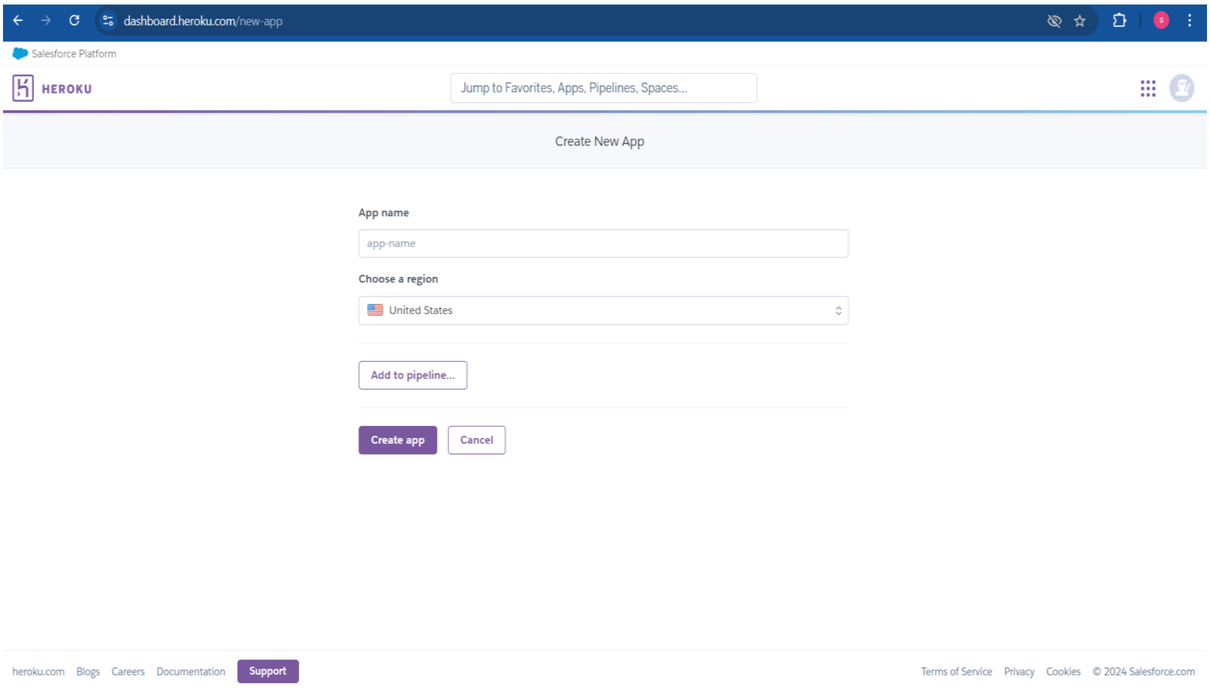
You don't have any apps yet

Every app and pipeline you create or collaborate on appears here.

Looking for help getting started?

Get started by reading one of our language guides in the Dev Center

Enter the app name, choose the region, and click "Create App".

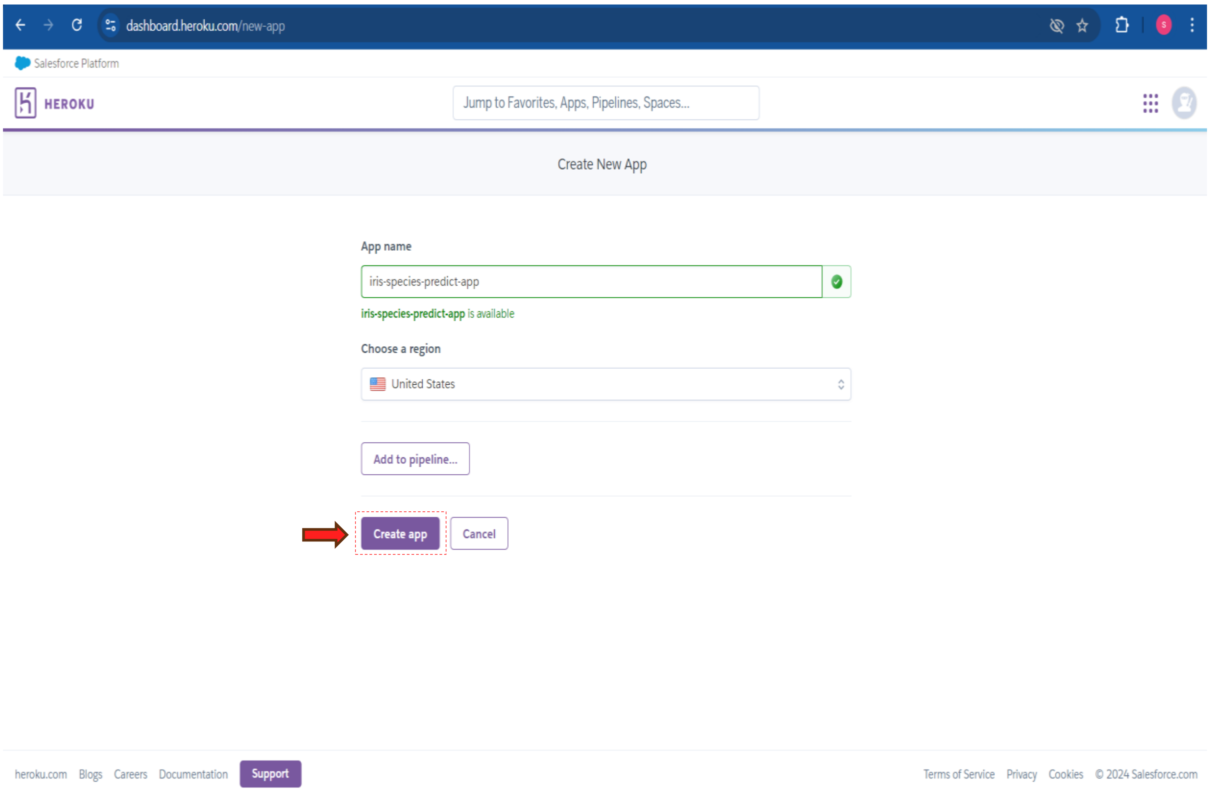


App name

app-name

Choose a region

United States



Create New App

App name

iris-species-predict-app

iris-species-predict-app is available

Choose a region

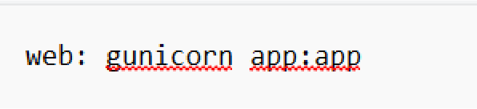
United States

6.3 Creating the Repository

Before connecting to GitHub, create a repository and upload the project files. Ensure you include `**Procfile**` ,`**requirements.txt**`, and `**Runtime.txt**`

Procfile:

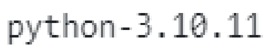
This is a simple text file without an extension that specifies the commands to be executed by the app on startup. Since we are using a web process type, the syntax is **`<process type>: <command>`**.



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

Runtime.txt:

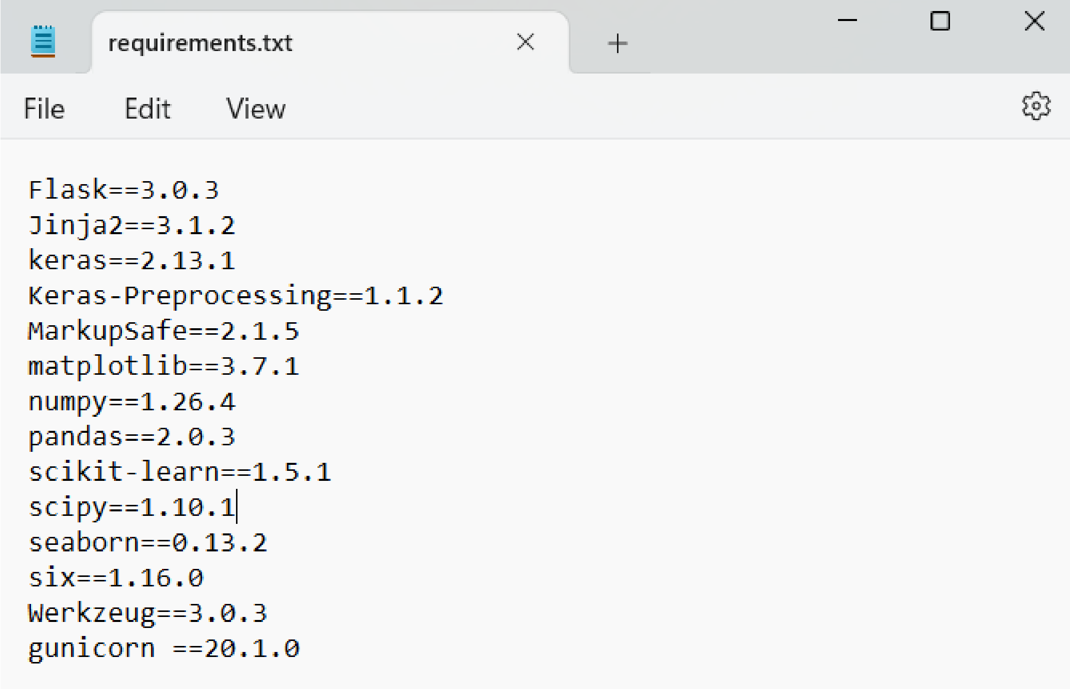
This file specifies the version of Python to be used. The packages listed in **requirements.txt** work best with a specific Python version, so we use **runtime.txt** to ensure compatibility.



```
python-3.10.11
```

Requirements.txt:

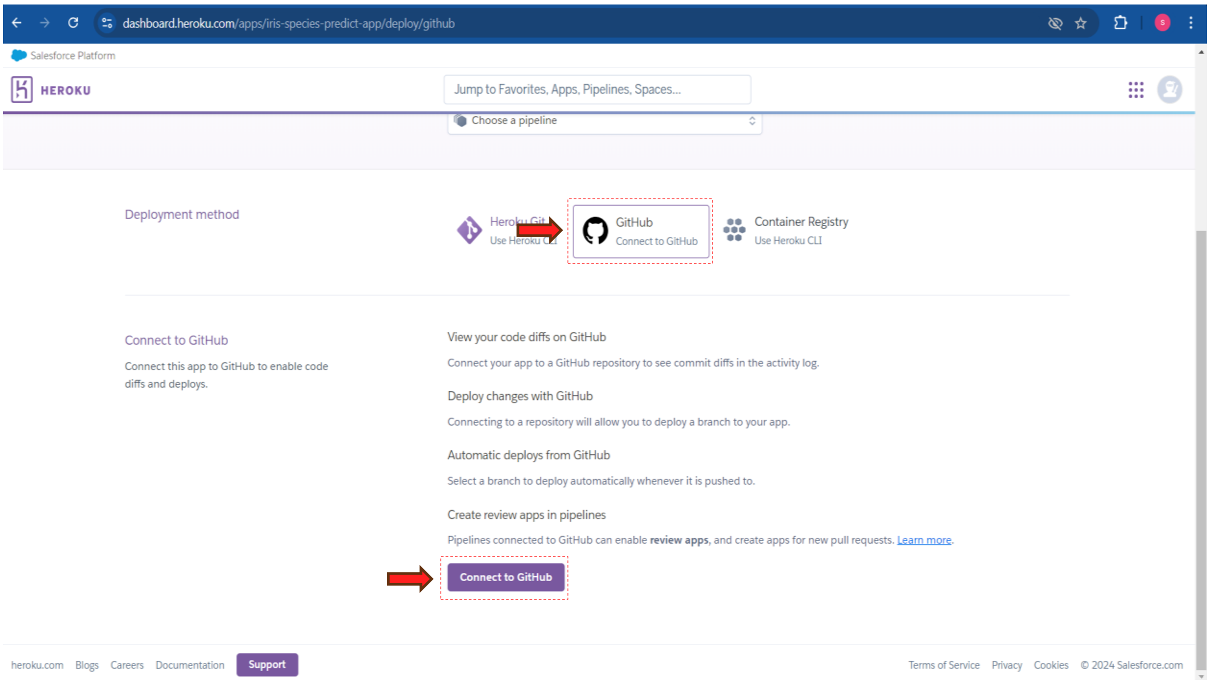
This file lists the Python packages required for the app. Use the `**pip freeze**` command to see the current versions of all the packages in visual studio, and include the necessary ones. Below are the versions I used:



```
Flask==3.0.3
Jinja2==3.1.2
keras==2.13.1
Keras-Preprocessing==1.1.2
MarkupSafe==2.1.5
matplotlib==3.7.1
numpy==1.26.4
pandas==2.0.3
scikit-learn==1.5.1
scipy==1.10.1
seaborn==0.13.2
six==1.16.0
Werkzeug==3.0.3
gunicorn ==20.1.0
```

6.4 connecting to GitHub

Select "GitHub" as the deployment method and click "Connect to GitHub".



Deployment method

Connect to GitHub

Connect this app to GitHub to enable code diffs and deploys.

View your code diffs on GitHub

Connect your app to a GitHub repository to see commit diffs in the activity log.

Deploy changes with GitHub

Connecting to a repository will allow you to deploy a branch to your app.

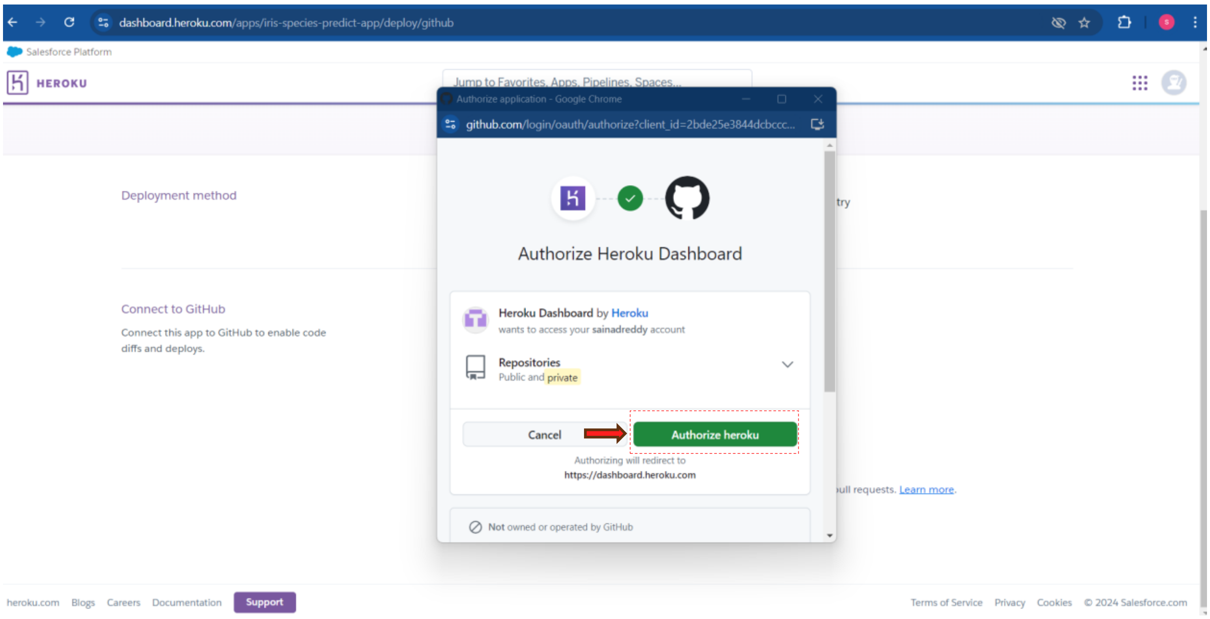
Automatic deploys from GitHub

Select a branch to deploy automatically whenever it is pushed to.

Create review apps in pipelines

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

Enter your GitHub credentials and click "Authorize Heroku".



dashboard.heroku.com/apps/iris-species-predict-app/deploy/github

Salesforce Platform

HEROKU

Jump to Favorites, Apps, Pipelines, Spaces...

Deployment method

Connect to GitHub

Connect this app to GitHub to enable code diffs and deploys.

Authorize application - Google Chrome

github.com/login/oauth/authorize?client_id=2bde25e3844dcbccc...

Authorize Heroku Dashboard

Heroku Dashboard by Heroku
wants to access your sainadreddy account

Repositories
Public and private

Cancel

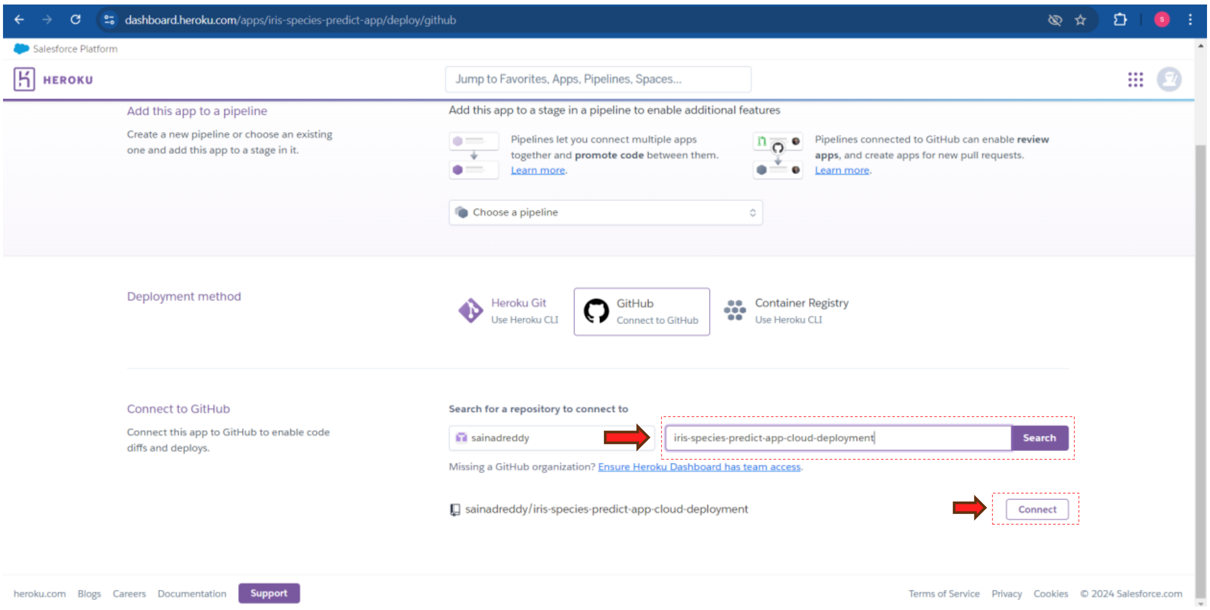
Authorize heroku

Authorizing will redirect to
https://dashboard.heroku.com

Not owned or operated by GitHub

6.5 After Linking GitHub

Search for the name of the repository you created and connect to it.



dashboard.heroku.com/apps/iris-species-predict-app/deploy/github

Salesforce Platform

HEROKU

Jump to Favorites, Apps, Pipelines, Spaces...

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.

Choose a pipeline

Deployment method

Heroku Git
Use Heroku CLI

GitHub
Connect to GitHub

Container Registry
Use Heroku CLI

Connect to GitHub

Connect this app to GitHub to enable code diffs and deploys.

Search for a repository to connect to

sainadreddy

iris-species-predict-app-cloud-deployment

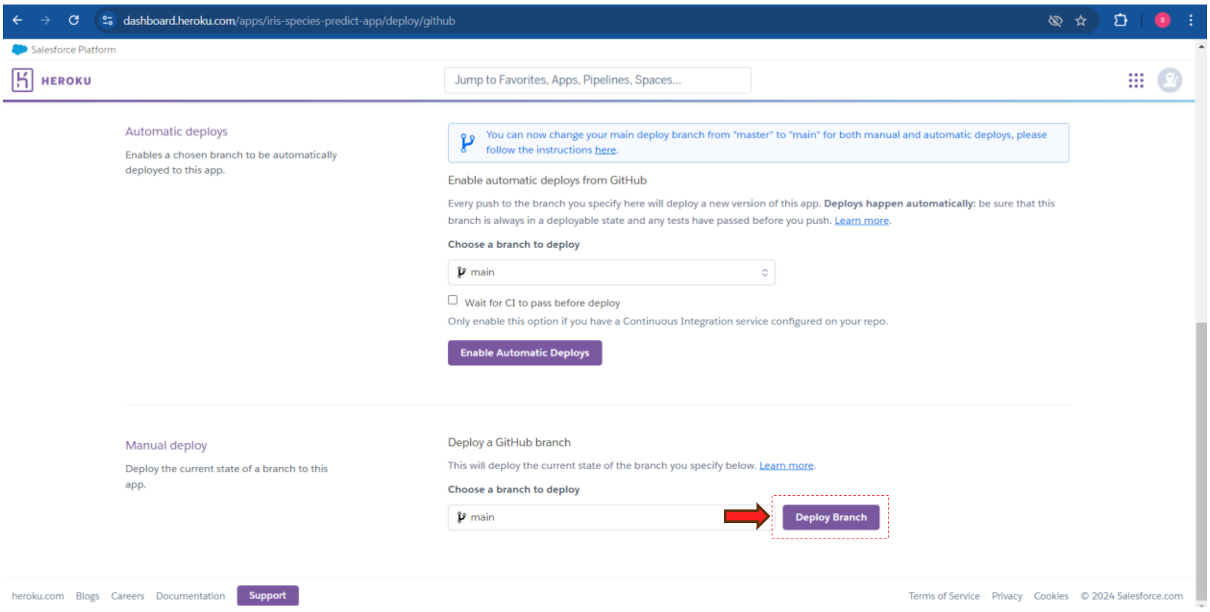
Search

Missing a GitHub organization? Ensure Heroku Dashboard has team access.

sainadreddy/iris-species-predict-app-cloud-deployment

Connect

Click "Deploy Branch".



dashboard.heroku.com/apps/iris-species-predict-app/deploy/github

Salesforce Platform

HEROKU

Jump to Favorites, Apps, Pipelines, Spaces...

Automatic deploys

Enables a chosen branch to be automatically deployed to this app.

You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the instructions here.

Enable automatic deploys from GitHub

Every push to the branch you specify here will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch is always in a deployable state and any tests have passed before you push. Learn more.

Choose a branch to deploy

main

□ Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

Enable Automatic Deploys

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

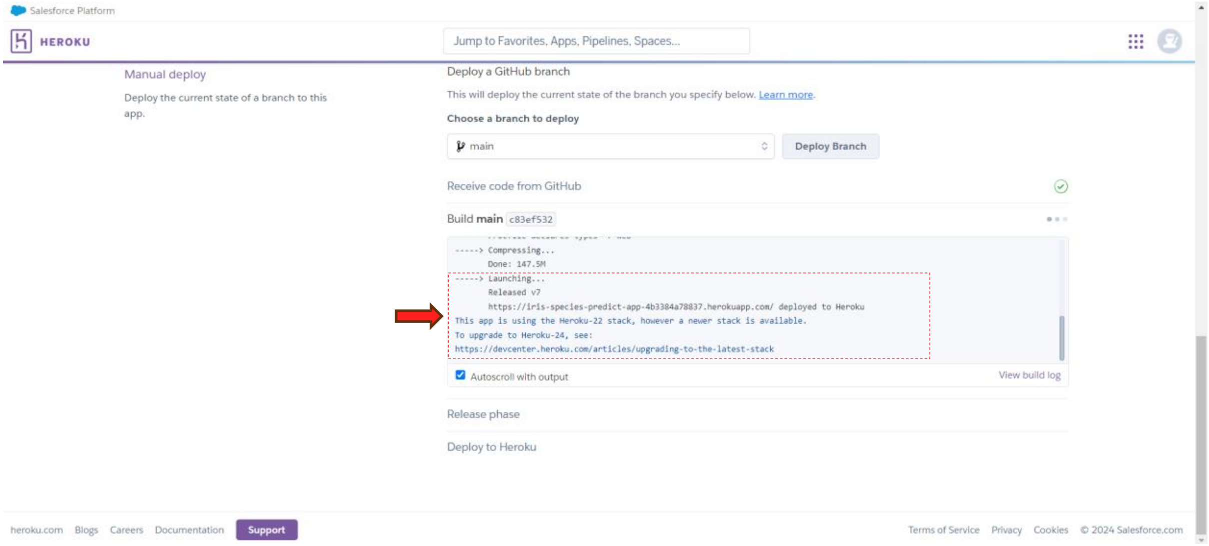
main

Deploy Branch

heroku.com Blogs Careers Documentation Support

Terms of Service Privacy Cookies © 2024 Salesforce.com

After a short while, you should see the deployment status. You can now click "View" or paste the provided link to see your app.



Salesforce Platform

HEROKU

Jump to Favorites, Apps, Pipelines, Spaces...

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** c83ef532

```
-----> Compressing...
       Done: 147.5M
-----> Launching...
       Released v7
       https://iris-species-predict-app-4b3384a78837.herokuapp.com/ deployed to Heroku
This app is using the Heroku-22 stack, however a newer stack is available.
To upgrade to Heroku-24, see:
https://devcenter.heroku.com/articles/upgrading-to-the-latest-stack
```

☑ Autoscroll with output

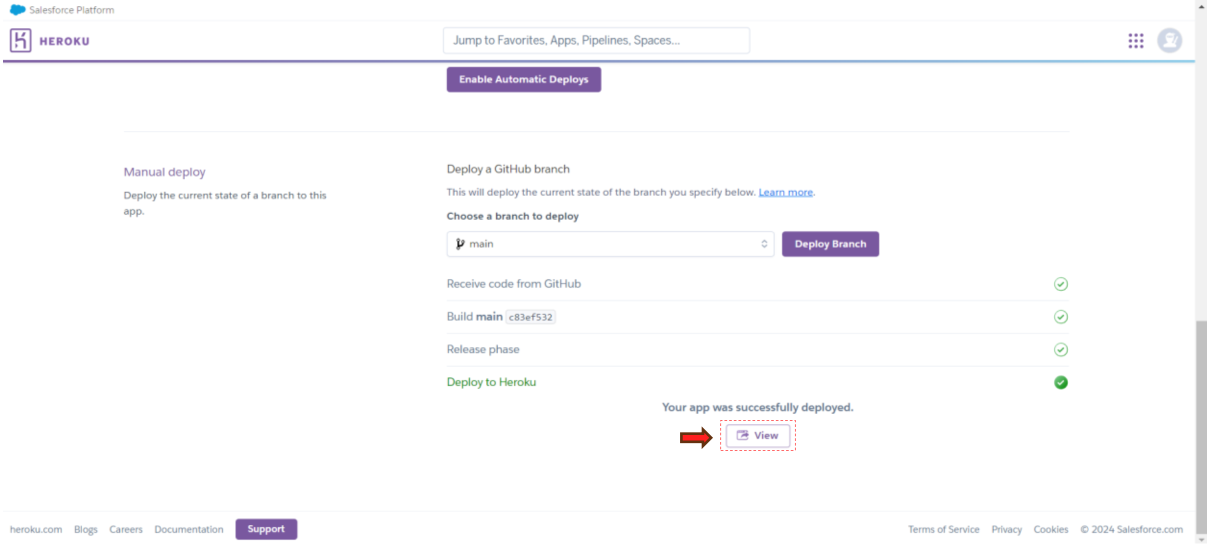
View build log

Release phase

Deploy to Heroku

heroku.com Blogs Careers Documentation Support

Terms of Service Privacy Cookies © 2024 Salesforce.com



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 ✓

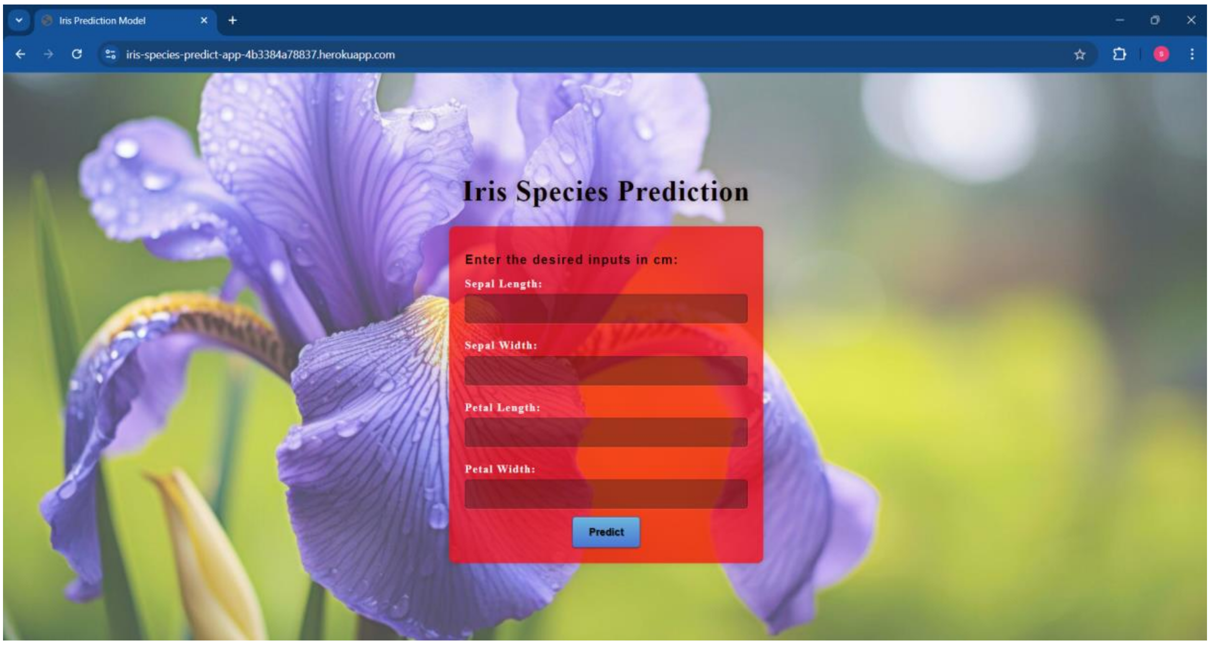
Build main c83ef532 ✓

Release phase ✓

Deploy to Heroku ✓

Your app was successfully deployed.

6.6 Testing the App



Iris Species Prediction

Enter the desired inputs in cm:

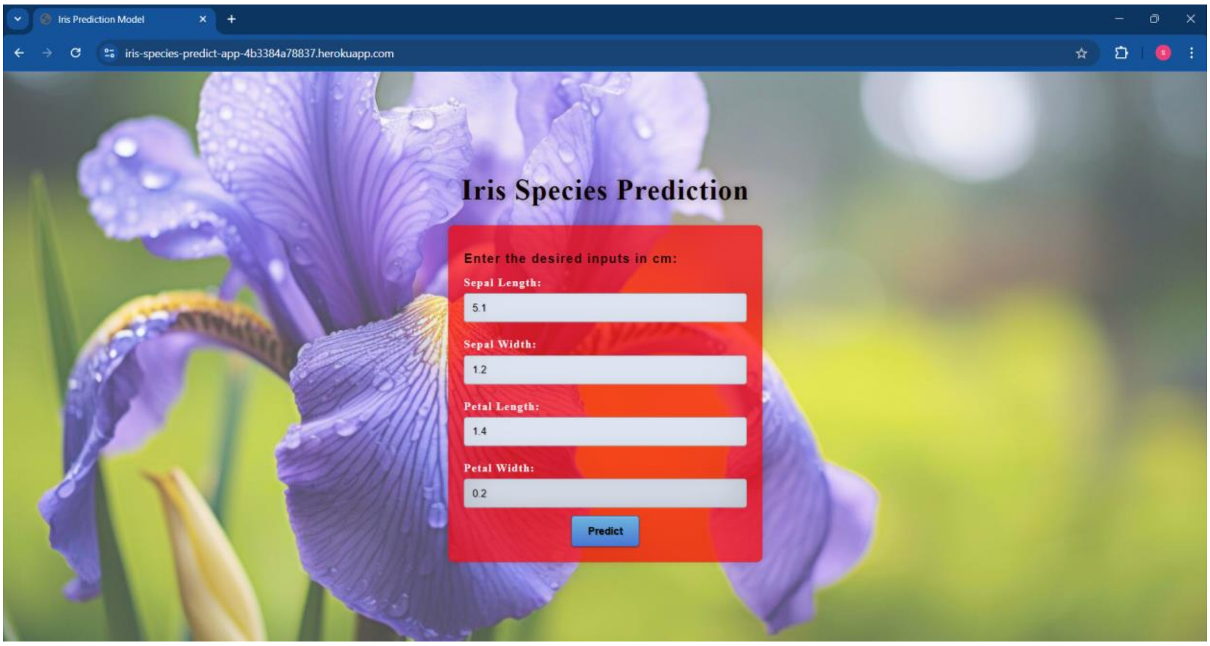
Sepal Length:

Sepal Width:

Petal Length:

Petal Width:

Enter the inputs in the web app.



iris-species-predict-app-4b3384a78837.herokuapp.com

Iris Species Prediction

Enter the desired inputs in cm:

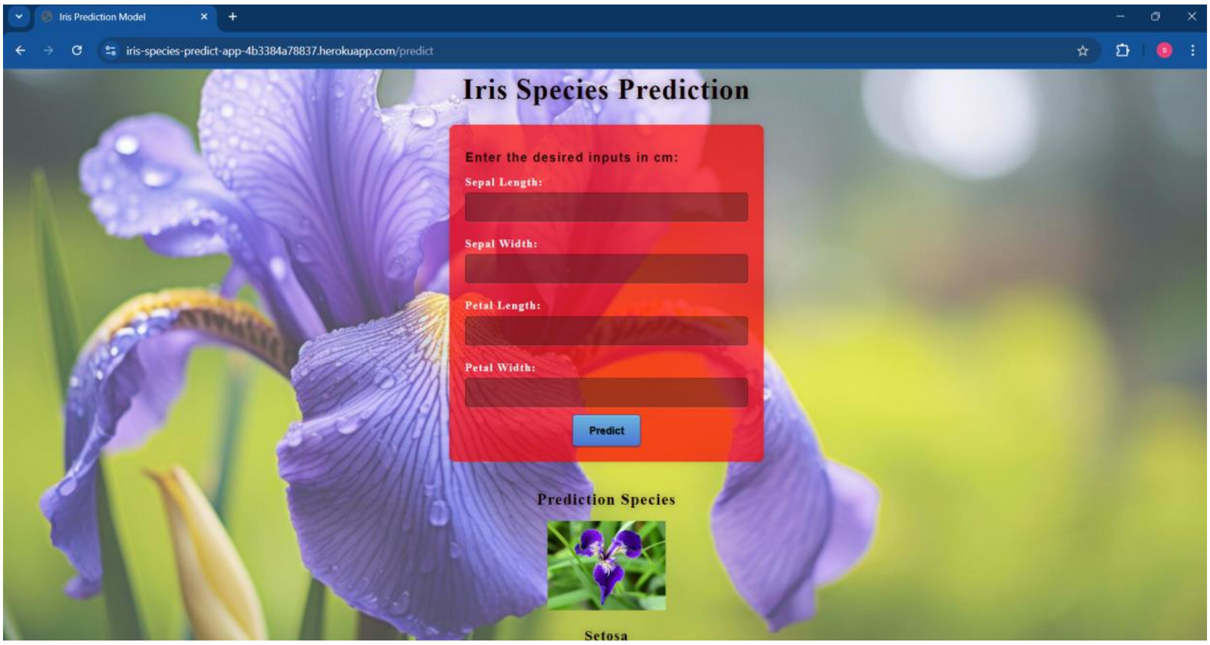
Sepal Length: 5.1

Sepal Width: 1.2

Petal Length: 1.4

Petal Width: 0.2

View the output.



iris-species-predict-app-4b3384a78837.herokuapp.com/predict

Iris Species Prediction

Enter the desired inputs in cm:

Sepal Length:

Sepal Width:

Petal Length:

Petal Width:

Prediction Species

Setosa

This completes the detailed deployment process on Heroku using the GitHub integration method.