WEEK 4: Deployment on Flask

Name: Sainad Reddy Naini
Batch Code: LISUM35
Submission Date: 28/07/2024
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

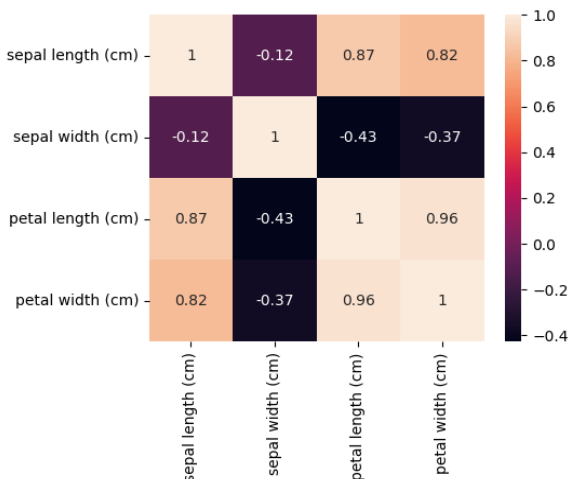
In [6]:
```
#Checking dattypes
iris.dtypes
```
Out[6]:
```
sepal length (cm)     float64
sepal width (cm)      float64
petal length (cm)     float64
petal width (cm)      float64
species              category
dtype: object
```
In [7]:
```
#Sum of Null_values
iris.isnull().sum()
```
Out[7]:
```
sepal 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.

app.py > ...

```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

templates > index.html > ...

```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

static > # styles.css > html

```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;
}
```

0 Kodezi Ln 17, Col 21 Spaces: 4 UTF-8 CRLF

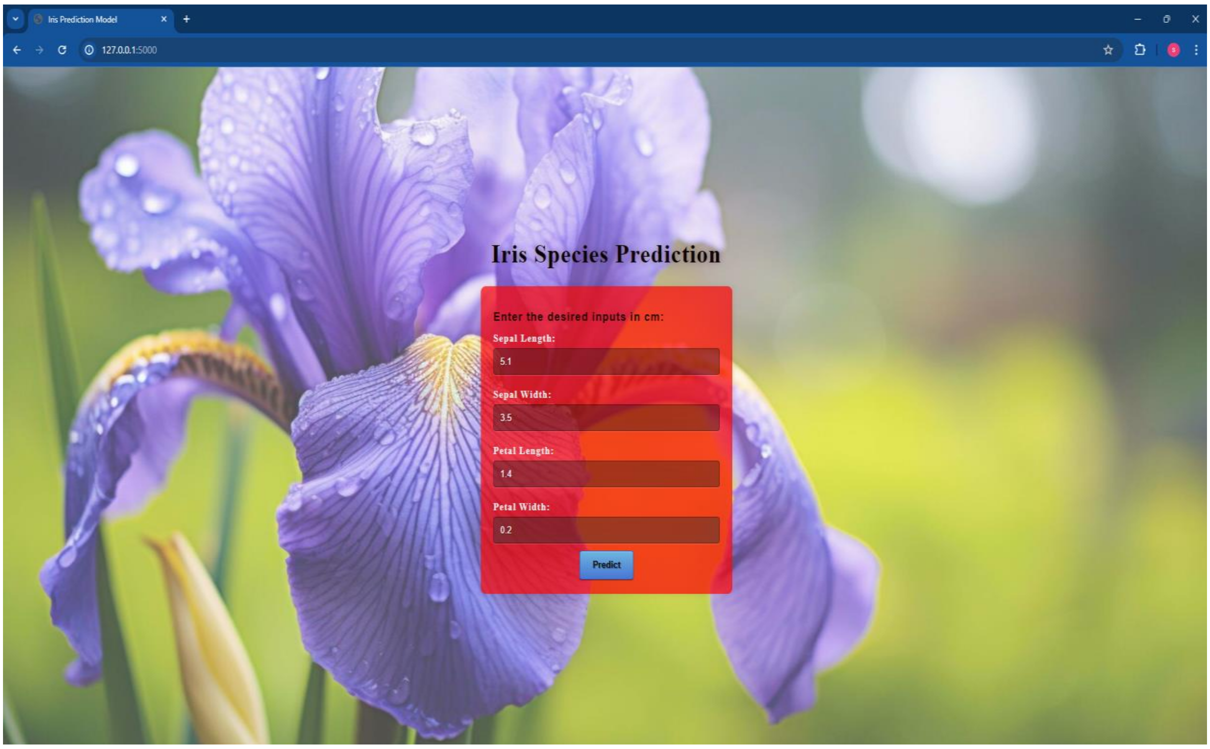
static > # styles.css > html

```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);
}

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

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.

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



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.