Name: Devin Chau
Report date: 11/26//2024
Internship Batch: LISUM39
Version:<1.0>
Data intake reviewer: Data Glacier
Data storage location: https://github.com/mynameisdevinchau/Data-Glacier-Internship/tree/main/Week%204

1.1 Loading and Exploring the Dataset

The Iris dataset from sklearn.datasets has 3 species of flowers: setosa, virginica, and versicolor. This simple, toy dataset will allow us to understand Flask without overcomplicating the data side of the project.

```
from sklearn.datasets import load_iris

data = load_iris()
df = pd.DataFrame(data=data.data, columns=data.feature_names)
df['species'] = pd.Categorical.from_codes(data.target, categories=data.target_names)
df
```

	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
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

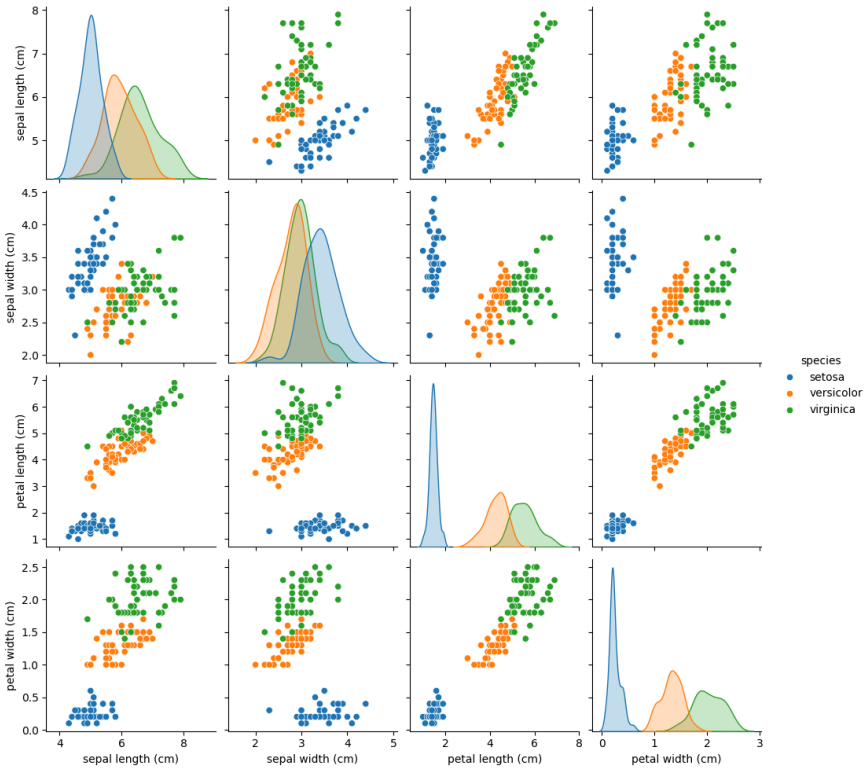
150 rows × 5 columns

We need to check whether the dataset itself has any null values to understand the dataset and avoid future problems.

```
df.isna().sum()
```

	0
sepal length (cm)	0
sepal width (cm)	0
petal length (cm)	0
petal width (cm)	0
species	0

1.2 Visualizing the Dataset



Pairplot of the dataset to visualize the relationships. We can see from the visualizations that the different species tend to group up on the scatter plots.

1.3 Training and Splitting the Model

```
#Building the model
X = df.drop(['species'], axis=1)
y = df['species']
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, train_size=0.70,test_size=0.30, random_state=0)
print(X_train.shape, X_test.shape)
```

```
(105, 4) (45, 4)
```

Used 3 Classification Models

Random Forest Classifier

```
randomForestClassifier = RandomForestClassifier(n_estimators=100, random_state=0)
randomForestClassifier.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=0)
```

KNN

```
kNearestNeighbors = KNeighborsClassifier(n_neighbors=5)
kNearestNeighbors.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

Logistic Regression

```
logisticRegression = LogisticRegression()
logisticRegression.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression()
```

The accuracy we achieved on all 3 models is 97%+.

Used Pickle to save the data for deployment

```
with open('random_forest.pkl', 'wb') as file:
        pickle.dump(randomForestClassifier, file)

with open('knn.pkl', 'wb') as file:
        pickle.dump(kNearestNeighbors, file)

with open('logistic_regression.pkl', 'wb') as file:
        pickle.dump(logisticRegression, file)
```

2.1 Deploying the Model on Flask

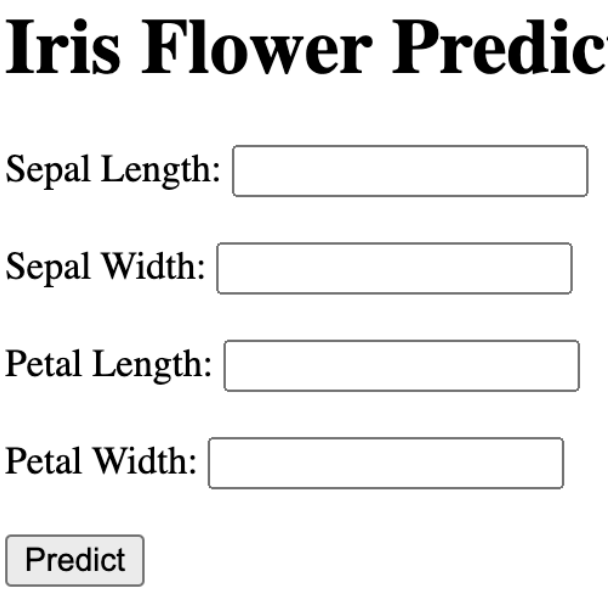
We initialized the Flask framework to deploy the model. We used the .pkl files that contain the data from the model we trained earlier.

```
app = Flask(__name__, template_folder='templates')

with open('random_forest.pkl', 'rb') as file:
    randomForestClassifier = pickle.load(file)

with open('knn.pkl', 'rb') as file:
    kNearestNeighbors = pickle.load(file)

with open('logistic_regression.pkl', 'rb') as file:
    logisticRegression = pickle.load(file)
```

```
app = Flask(__name__, template_folder='templates')

with open('random_forest.pkl', 'rb') as file:
    randomForestClassifier = pickle.load(file)

with open('knn.pkl', 'rb') as file:
    kNearestNeighbors = pickle.load(file)

with open('logistic_regression.pkl', 'rb') as file:
    logisticRegression = pickle.load(file)

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

@app.route('/predict', methods=['POST'])
def predict():
    int_features = [float(x) for x in request.form.values()]
    final_features = [np.array(int_features)]

    rfc_prediction = randomForestClassifier.predict(final_features)[0]
    knn_prediction = kNearestNeighbors.predict(final_features)[0]
    lr_prediction = logisticRegression.predict(final_features)[0]

    return render_template('index.html',
                           rfc_prediction=rfc_prediction,
                           knn_prediction=knn_prediction,
                           lr_prediction=lr_prediction)

if __name__ == "__main__":
    app.run(port=5000, debug=True)
```

3.1 Front End Development

Simple HTML web app to display and have the sepal length, sepal width, petal length, and petal width to predict what the flower is based on the values inputted



These are the values I get after plugging in 3, 2, 1, and 2 respectively to retrieve those predictions

Iris Flower Prediction

Sepal Length:

Sepal Width:

Petal Length:

Petal Width:

Predict

Random Forest Classifier Prediction: setosa

K Nearest Neighbors Prediction: setosa

Logistic Regression Prediction: setosa

HTML code

```
<!DOCTYPE html>
<html>
<head>
    <meta charset = "UTF-8">
    <title>Iris Flower Prediction</title>
</head>
<body>
    <h1>Iris Flower Prediction</h1>
    <form action="{{ url_for('predict')}}"method="POST">
        <label for="sepal_length">Sepal Length:</label>
        <input type="number" name="sepal_length" id="sepal_length" required><br><br>

        <label for="sepal_width">Sepal Width:</label>
        <input type="number" name="sepal_width" id="sepal_width" required><br><br>

        <label for="petal_length">Petal Length:</label>
        <input type="number" name="petal_length" id="petal_length" required><br><br>

        <label for="petal_width">Petal Width:</label>
        <input type="number" name="petal_width" id="petal_width" required><br><br>

        <button type="submit">Predict</button>
    </form>

    {% if rfc_prediction %}
        <h2>Random Forest Classifier Prediction: {{ rfc_prediction }}</h2>
    {% endif %}

    {% if knn_prediction %}
        <h2>K Nearest Neighbors Prediction: {{ knn_prediction }}</h2>
    {% endif %}

    {% if lr_prediction %}
        <h2>Logistic Regression Prediction: {{ lr_prediction }}</h2>
    {% endif %}
</body>
</html>
```