Data Intake Report

Name: Deployment on Flask
Report date: 27-April-2025
Internship Batch: LISUM44
Version: 1.0
Data intake by: Yash Parmar
Data intake reviewer: Data Glacier
Data storage location: https://github.com/yparmar2024/Data-Glacier/tree/main/Week%204

Tabular data details:

Total number of observations	150
Total number of files	1
Total number of features	5
Base format of the file	sklearn module
Size of the data	6.0 KB+

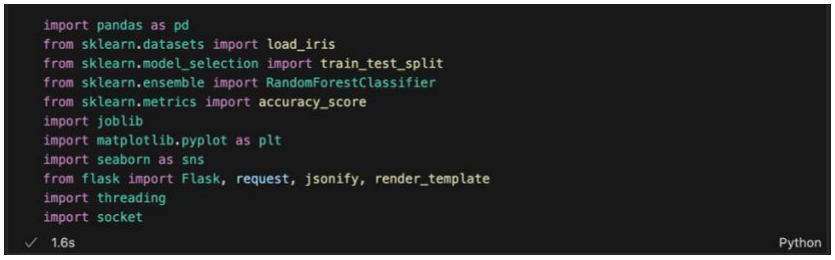
Introduction 1.0:

The Deployment on Flask project focuses on deploying a machine learning model for the Iris dataset classification using Flask. The model is trained using a RandomForestClassifier, and the deployment is done with a web application that allows users to input features and receive predictions on the type of flower.

Building Model 2.0:

Importing Libraries 2.1:

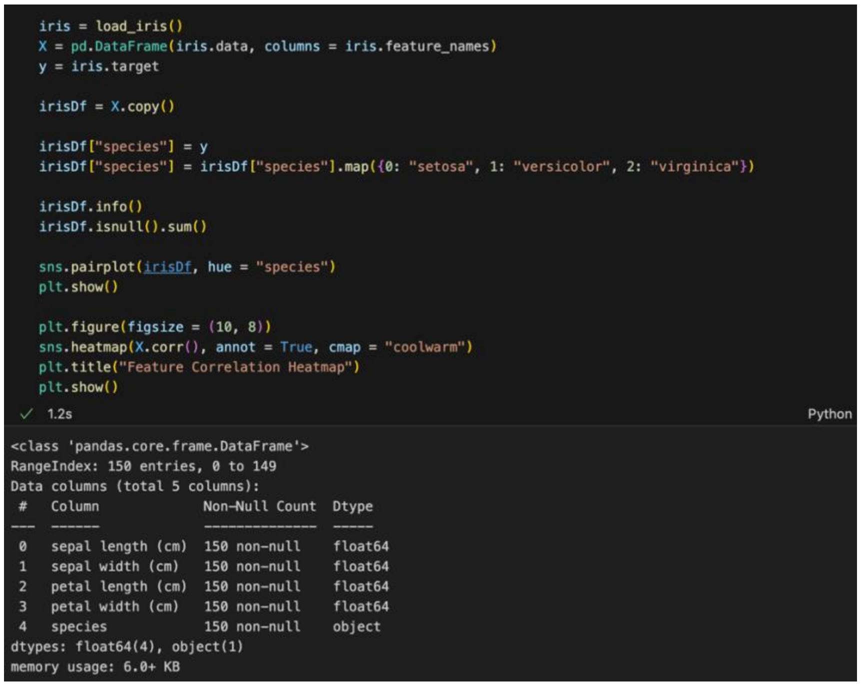
We have to import all the libraries needed, below I've listed all the libraries needed and why we need them:
- Pandas is used for efficient data handling
- load_iris loads Iris which is a toy dataset from the machine learning library, sklearn
- train_test_split streamlines the process of splitting, training and testing the dataset from the machine learning library, sklearn
- RandomForestClassifier handles large datasets with accuracy, ready for training from the machine learning library, sklearn
- accuracy_score computes the accuracy of the developed model from the machine learning library, sklearn
- joblib saves the model after the training is done to proceed with more training later from the Python library, joblibrary
- pyplot allows for visualizing the dataset and findings from the visualization library, matplotlib
- seaborn is used for making the visualizations look more appealing and with ease of use along with matplotlib
- Flask, request, jsonify, and render_template are basic modules needed to execute a Flask app from the web app library, Flask
- Threading allows for the app to have multiple requests and operations
- Socket allows for real-time communication between client and server



```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import joblib
import matplotlib.pyplot as plt
import seaborn as sns
from flask import Flask, request, jsonify, render_template
import threading
import socket
```

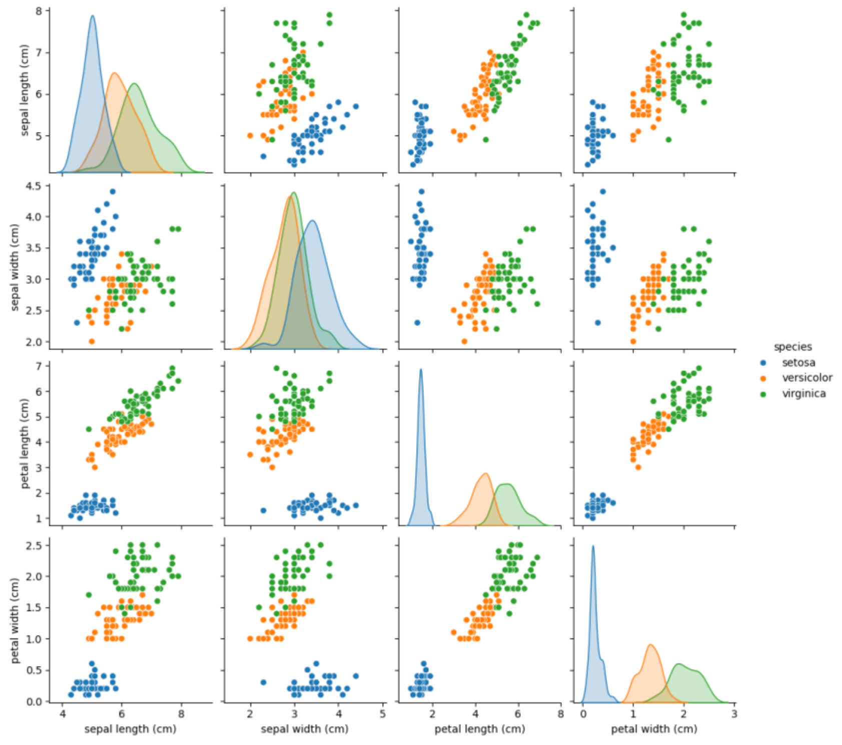
Exploratory Data Analysis 2.2:

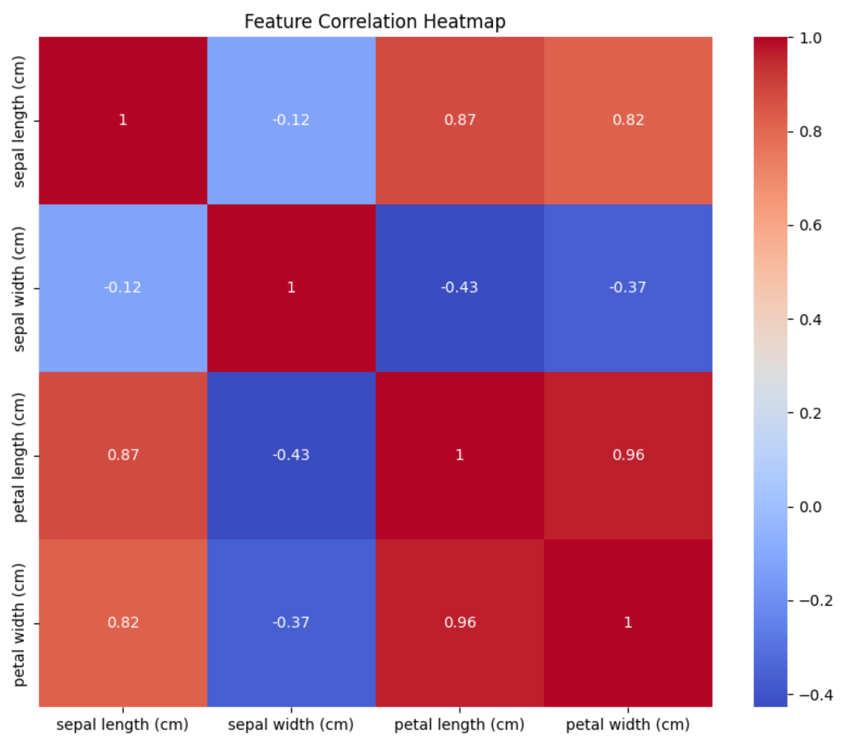
This code loads the Iris dataset from the sklearn library and puts it into a data frame using the Pandas library. After, instead of numerical values, there are strings assigned to each true classification label. After analyzing the data frame and checking for null values, the data is then plotted using a pair plot, which shows how different measurements relate to each other and how they separate the species and a correlation heatmap which shows which features are most strongly correlated. These visualizations help understand the dataset's structure and relationships before building any machine learning models.



```python
iris = load_iris()
X = pd.DataFrame(iris.data, columns = iris.feature_names)
y = iris.target

irisDf = X.copy()

irisDf["species"] = y
irisDf["species"] = irisDf["species"].map({0: "setosa", 1: "versicolor", 2: "virginica"})

irisDf.info()
irisDf.isnull().sum()

sns.pairplot(irisDf, hue = "species")
plt.show()

plt.figure(figsize = (10, 8))
sns.heatmap(X.corr(), annot = True, cmap = "coolwarm")
plt.title("Feature Correlation Heatmap")
plt.show()
```

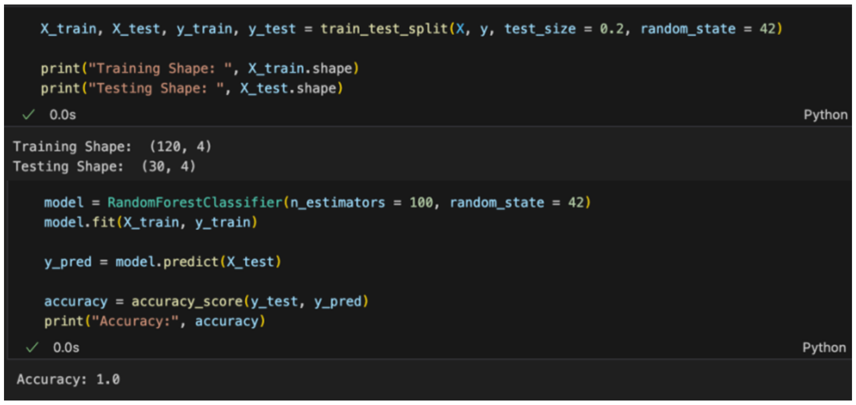
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   species            150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```





Training Model 2.3:

This part is handling the machine learning workflow for the Iris dataset. First, it splits the x and y sets of data into training (80%) and testing (20%) sets in order to learn on some of the data and test on the rest. Check the shapes of each set to ensure compatibility of the model. A Random Forest classifier with 100 decision trees is then trained on the training data. The model makes predictions on the test set, achieving an accuracy score that measures its performance. The accuracy of the machine learning model is then displayed through its module.



```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)

print("Training Shape: ", X_train.shape)
print("Testing Shape: ", X_test.shape)
```

✓ 0.0s Python

```
Training Shape:  (120, 4)
Testing Shape:  (30, 4)
```

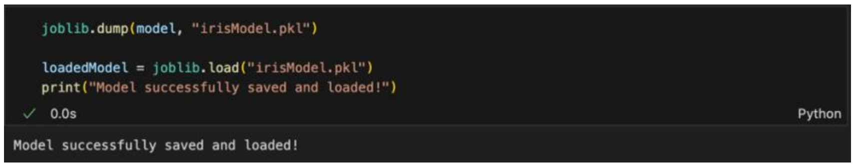
```
model = RandomForestClassifier(n_estimators = 100, random_state = 42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

✓ 0.0s Python

```
Accuracy: 1.0
```

Saving Model 2.4:

This part is simply saving the model for future training or predictions that way it doesn't get lost and retraining does not need to be done. Verify that the model is successfully saved and loaded through a simple load and print statement.



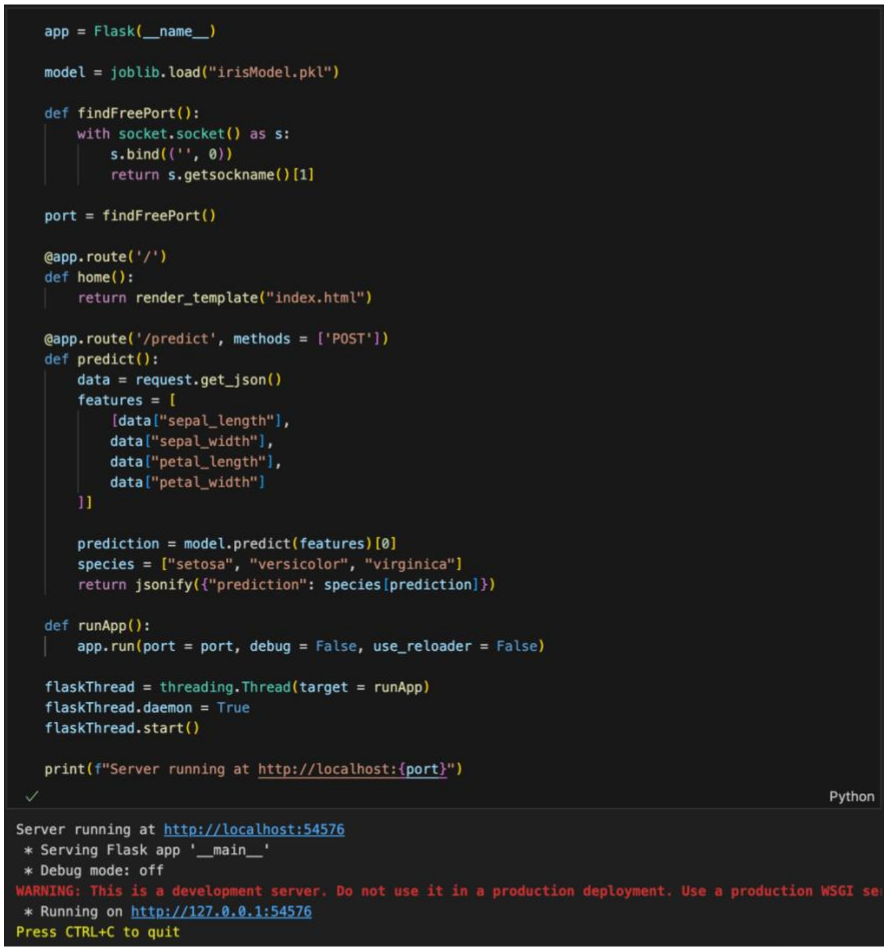
```python
joblib.dump(model, "irisModel.pkl")

loadedModel = joblib.load("irisModel.pkl")
print("Model successfully saved and loaded!")
```
✓ 0.0s Python

```
Model successfully saved and loaded!
```

Building Flask App 3.0:

Importing Libraries 3.1:

Let’s start building the Flask web app, start by initializing the app and then loading the model so that it’s usable. With the idea that not every port is not always being used on each device, a free port function can be written to find a free port on each device. This can be used to start the app route and load in the home page where the HTML file is rendered from the templates folder which was coded. After, the predict function can be written to help process each request based on the trained model from earlier. There can be a function written to run the app which will run the app on a free port using a Daemon thread. Then, it will output the link to click on for the web page to the user.



```python
app = Flask(__name__)

model = joblib.load("irisModel.pkl")

def findFreePort():
    with socket.socket() as s:
        s.bind(('', 0))
        return s.getsockname()[1]

port = findFreePort()

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

@app.route('/predict', methods = ['POST'])
def predict():
    data = request.get_json()
    features = [
        [data["sepal_length"],
        data["sepal_width"],
        data["petal_length"],
        data["petal_width"]
    ]]

    prediction = model.predict(features)[0]
    species = ["setosa", "versicolor", "virginica"]
    return jsonify({"prediction": species[prediction]})

def runApp():
    app.run(port = port, debug = False, use_reloader = False)

flaskThread = threading.Thread(target = runApp)
flaskThread.daemon = True
flaskThread.start()

print(f"Server running at http://localhost:{port}")
```
✓ Python

```
Server running at http://localhost:54576
 * Serving Flask app '__main__'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI se
 * Running on http://127.0.0.1:54576
Press CTRL+C to quit
```