

# Data Intake Report

Name: Cloud and API Deployment

Report date: 4-May-2025

Internship Batch: LISUM44

Version: 1.0

Data intake by: Yash Parmar

Data intake reviewer: Data Glacier

Data storage location: <https://github.com/y Parmar2024/Data-Glacier/tree/main/Week%205>

## 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 Cloud and API Deployment project focuses on deploying a machine learning model for the Iris dataset classifications using Flask and a cloud deployment platform such as Render. This is the same model that was created, trained, and saved from Week 4.

## Setting up for Deployment 2.0:

### Creating the App 2.1:

To start setting up the deployment, I used Week 4's Flask Web App code as a basis and rewrote it so that it would be hosted on Render. I didn't have to use the find free port function from the Week 4 since Render provides the user with a domain therefore I didn't need socket. I didn't need a run app function since Render handles that so I also didn't need thread, therefore I just needed the functions home and predict for any basic web app. The rest of the code was very similar where we render the HTML template from the previous week, and use predict to get predictions from the model to output to the user.

```
Week 5 > api > app.py > _
1 from flask import Flask, request, jsonify, render_template
2 import joblib
3 import pandas as pd
4
5 app = Flask(__name__)
6 model = joblib.load("irisModel.pkl")
7
8 @app.route("/")
9 def home():
10     return render_template("index.html")
11
12 @app.route("/predict", methods = ["POST"])
13 def predict():
14     data = request.get_json()
15
16     features = pd.DataFrame([
17         data["sepal_length"],
18         data["sepal_width"],
19         data["petal_length"],
20         data["petal_width"]
21     ], columns = ["sepal length (cm)", "sepal width (cm)", "petal length (cm)", "petal width (cm)"])
22
23     prediction = model.predict(features)[0]
24     species = ["setosa", "versicolor", "virginica"]
25     return jsonify({"prediction": species[prediction]})
26
27 if __name__ == "__main__":
28     app.run(debug = True)
```

## Saving the Model 2.2:

Since we saved the model last week into our Week 4 folder, I just decided to copy it over. Other ways to do this is to just retrain the model and have it saved into this folder, I decided to take the easier way.

## Creating Requirements 2.3:

We need to set up the requirements file so the user knows which version of each module they should be using, which the machine automatically does. For this, I simply put the versions which I used when creating the model.

```
Week 5 > api > requirements.txt
1  Flask==2.3.2
2  joblib==1.3.2
3  scikit-learn==1.6.1
4  gunicorn
5  pandas==2.2.3
```

## Creating a Configuration File 2.4:

We need to create a configuration file to allow for quick deployment which states features such as build command, start command, plan, branch, etc. These allow for Render to know the different configurations to apply to the web app.

```
Week 5 > api > render.yaml > [ ] services > { } 0 > branch
Render Blueprints - Blueprints are Render's infrastructure-as-code model for defining, deploying, and managing multiple resources with a single YAML file (render.yaml.json)
1  services:
2    - type: web
3      name: iris-classifier
4      runtime: python
5      buildCommand: "pip install -r requirements.txt"
6      startCommand: "gunicorn app:app"
7      envVars:
8        - key: FLASK_ENV
9          value: production
10     plan: free
11     branch: main
```

## Creating a Model Training Script 2.5:

Although it's not needed, for future training for the model to increase the accuracy, we can add a script for training the model more. This is simply using the train, test, split method but instead of training it on 20% of the dataset, we now train it on 100% since we already trained the model. Then, we save the model to be able to be reused for the next deployment or updated.

```

Week 5 > api > trainModel.py > ...
1  from sklearn.datasets import load_iris
2  from sklearn.ensemble import RandomForestClassifier
3  import joblib
4  import os
5
Windsurf: Refactor | Explain | Generate Docstring | X
6  def trainSaveModel():
7      # Load iris dataset to be trained on
8      iris = load_iris()
9      # Use all of the Iris dataset for training now that the model has been created
10     X, y = iris.data, iris.target
11
12     # Train model using RandomForestClassifier
13     model = RandomForestClassifier(n_estimators = 100, random_state = 42)
14     model.fit(X, y)
15
16     # Save model and let user know it's saved
17     modelPath = os.path.join(os.path.dirname(__file__), "irisModel.pkl")
18     joblib.dump(model, modelPath)
19     print("Production model saved to {modelPath}")
20
21 if __name__ == "__main__":
22     trainSaveModel()

```

## Deployment through Render 3.0:

### Setting up the Render Service 3.1:

Now, that we have the app all ready to be deployed with the right files, we can get ready to deploy it. Let's use Render, since it provides a free tier for a web service. We can choose the github repository where the files are saved and put in the settings needed for the web service.

The screenshot shows the Render dashboard interface. On the left is a sidebar with navigation options: Dashboard, Iris Machine Learning, Events, Settings (highlighted), MONITOR (Logs, Metrics), and MANAGE (Environment, Shell, Scaling, Previews, Disks, Jobs). The main content area is titled 'Iris Machine Learning' and shows it's a 'WEB SERVICE' on 'Python 3' with a 'Free' plan. It includes a 'Connect' button and a 'Manual Deploy' dropdown. Below this, the 'Settings' section is visible, containing a 'General' tab with fields for 'Name' (Iris Machine Learning), 'Region' (Virginia (US East)), and 'Instance Type' (Free, 0.1 CPU, 512 MB). A message prompts the user to ask the owner (yparmar2024@gmail.com) to enter their payment information. At the bottom, there's a 'Build & Deploy' section.

### Viewing the Web App 3.2:

After deploying the web app and viewing the domain provided by Render, we can see that our web app was successfully deployed. The web app has a title, “Iris Species Predictor”, to let the user know what this web app is about. There’s also a note, “Note: This web app is on a free tier, therefore the first prediction may take a bit longer due to server spin-up. Subsequent predictions should be faster”, to let the user know that since this is a free tier, there will be spin-up lag in predictions. There’s also four input options: “Sepal Length (cm)”, “Sepal Width (cm)”, “Petal Length (cm)”, and “Petal Width (cm)” for the user to type inputs into for the prediction of the flower. There’s also a button to predict, “Predict Species”, which should be pressed after inserting the four inputs. It’s a simple web app, but it gets the job done; to predict the classification of the Iris species from the different characteristics through machine learning.

## Iris Species Predictor

Note: This web app is on a free tier, therefore the first prediction may take a bit longer due to server spin-up. Subsequent predictions should be faster.

Sepal Length (cm):

Sepal Width (cm):

Petal Length (cm):

Petal Width (cm):

Predict Species

**Predicted Species:** **virginica**