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/yparmar2024/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:

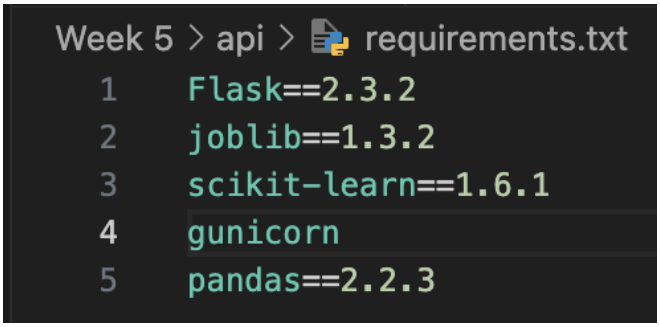
To start setting up the deployment, I used Week 4's Flask Web App code as a basis and rewrote it so that it would be hosted on Render. I didn't have to use the find free port function from the Week 4 since Render provides the user with a domain therefore I didn't need socket. I didn't need a run app 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.

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.

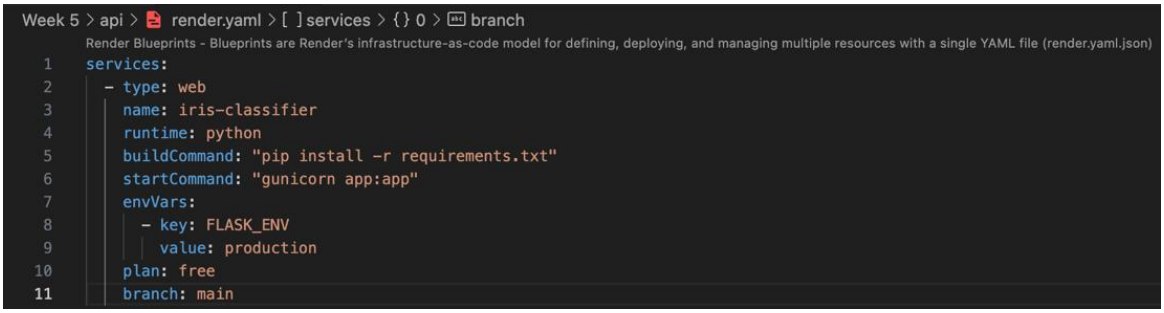
Creating Requirements 2.3:

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.



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.



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.

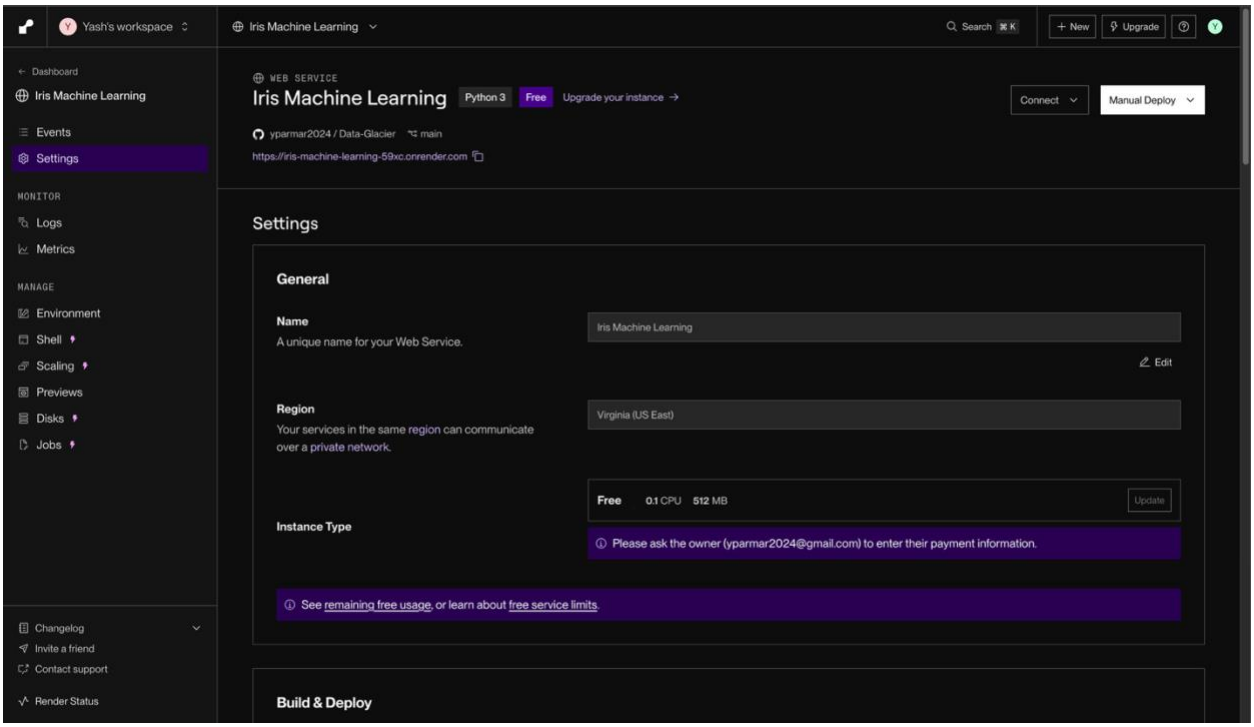
Week 5 > api > trainModel.py > ...

```python
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
import joblib
import os

Windsurf: Refactor | Explain | Generate Docstring | X
def trainSaveModel():
    # Load iris dataset to be trained on
    iris = load_iris()
    # Use all of the Iris dataset for training now that the model has been created
    X, y = iris.data, iris.target

    # Train model using RandomForestClassifier
    model = RandomForestClassifier(n_estimators = 100, random_state = 42)
    model.fit(X, y)

    # Save model and let user know it's saved
    modelPath = os.path.join(os.path.dirname(__file__), "irisModel.pkl")
    joblib.dump(model, modelPath)
    print("Production model saved to {modelPath}")

if __name__ == "__main__":
    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.



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.