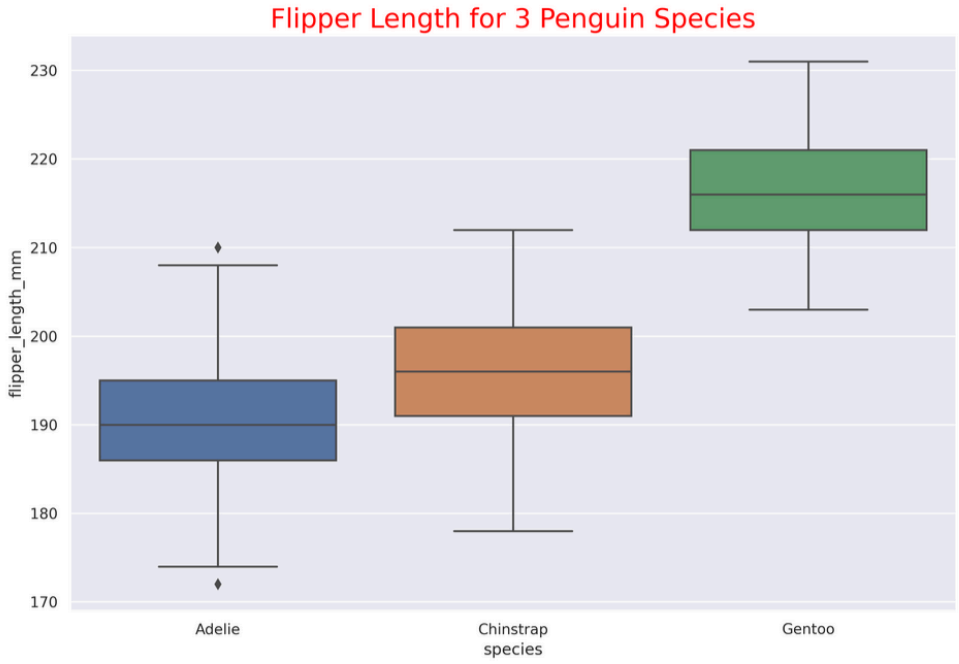
DataGlacier: Week #4

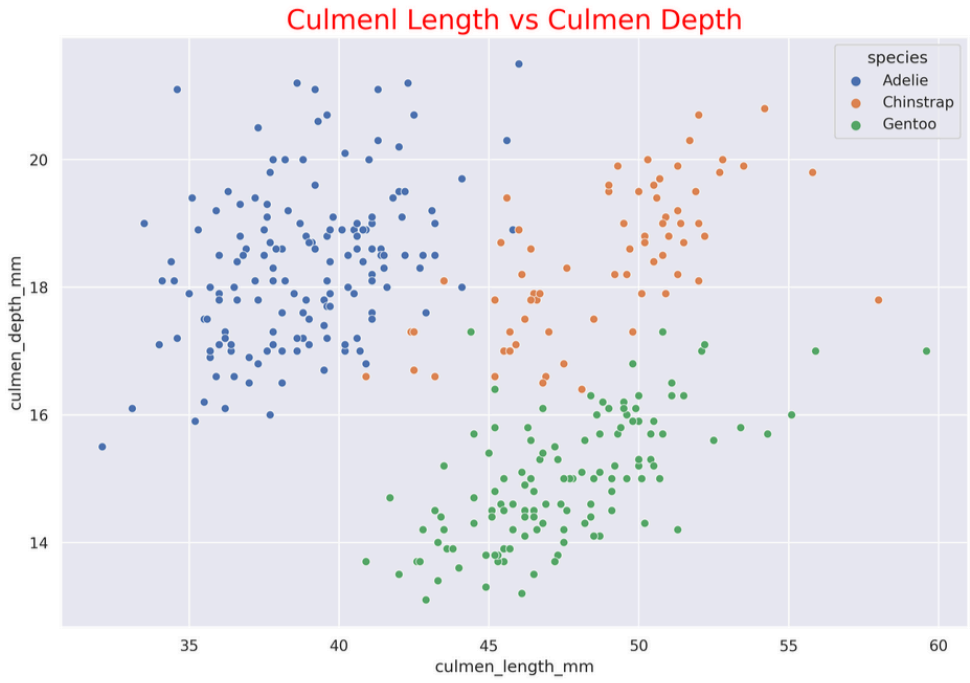
Deployment on Flask

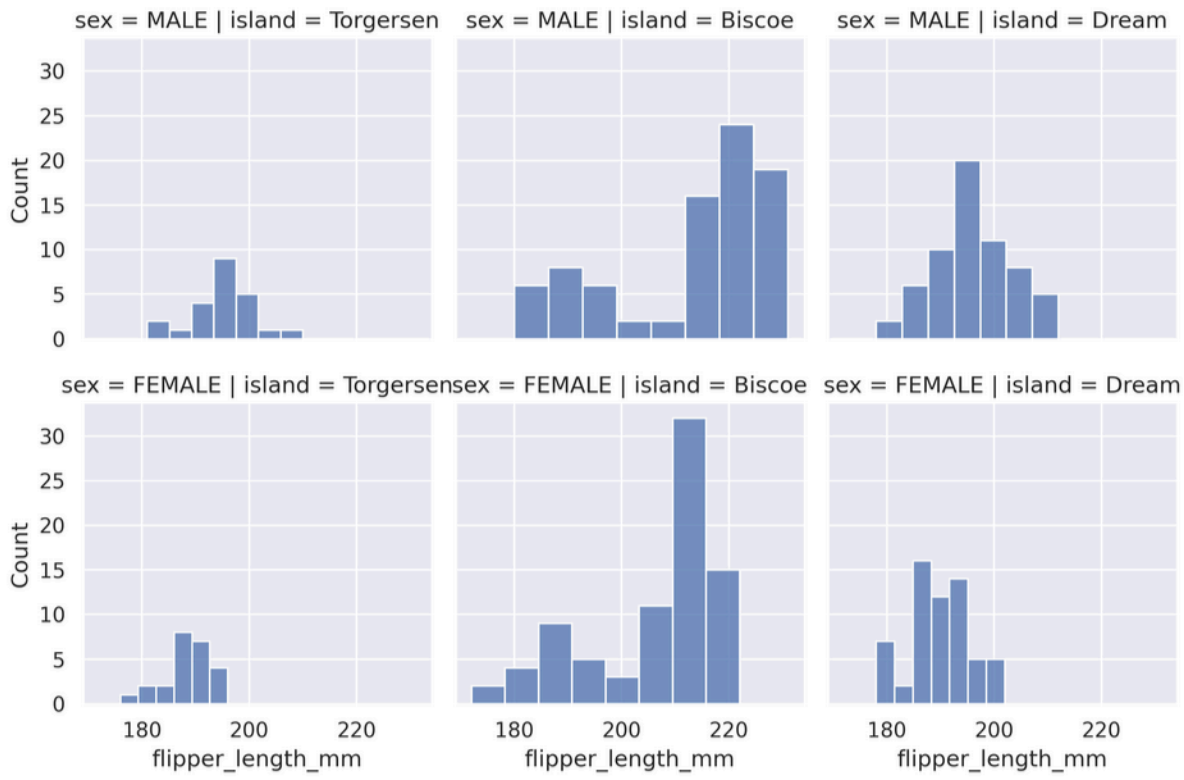
Name: Ethan Dy
Batch Code: LISUM39
Date: November 28, 2024
Submitted to: Github Repository to DataGlacier

Step 1 (Dataset and Model Training):

The Palmer Penguins dataset was used for this project. It includes features such as bill length, bill depth, flipper length, and body mass to classify penguins into three species: **Adelie**, **Chinstrap**, and **Gentoo**.







The dataset is part of the seaborn library's built-in datasets. The data was preprocessed to handle missing values, and a Random Forest Classifier was trained to achieve over 99-100% accuracy.

```python
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
import pickle
import os

# Load the dataset
data = sns.load_dataset("penguins")

# Drop rows with missing values
data = data.dropna()

# Encode the target variable (species)
label_encoder = LabelEncoder()
data["species"] = label_encoder.fit_transform(data["species"])

# Select features and target
X = data[["bill_length_mm", "bill_depth_mm", "flipper_length_mm", "body_mass_g"]]
y = data["species"]

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a Random Forest Classifier
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

The trained model and its metadata were saved using pickle for deployment.

```
29      # Save the model
30      os.makedirs("datasets", exist_ok=True)
31      model_filename = "datasets/penguin_model.pkl"
32      with open(model_filename, "wb") as file:
33          pickle.dump(model, file)
34
35      # Save feature names
36      feature_names = X.columns.to_list()
37      features_filename = "datasets/penguin_features.pkl"
38      with open(features_filename, "wb") as file:
39          pickle.dump(feature_names, file)
40
41      # Save label encoder
42      encoder_filename = "datasets/label_encoder.pkl"
43      with open(encoder_filename, "wb") as file:
44          pickle.dump(label_encoder, file)
45
46      # Print model accuracy
47      accuracy = model.score(X_test, y_test)
48      print(f"Model accuracy: {accuracy * 100:.2f}%")
```

Step 2 (Flask App Development):

A Flask web application was created to allow users to input penguin features and receive predictions.

The application loads the trained model and features for real-time predictions. The app.py script was developed to handle user inputs, predict species, and display the results dynamically.

First, we initialized a Flask app to serve as the web framework for deploying the model.

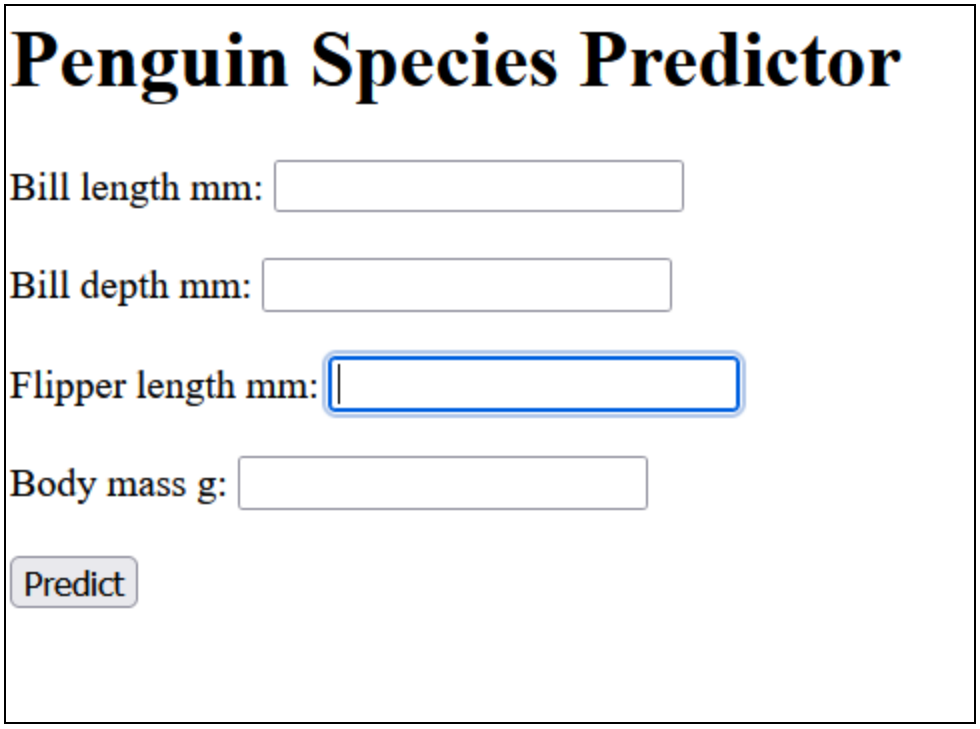
Next, we opened the .pkl files (containing the trained model, feature names, and label encoder) using Python's pickle module. These files were generated during the training process and are essential for making predictions in the Flask app.

```python
from flask import Flask, request, render_template
import pickle

app = Flask(__name__)

# Load the model, feature names, and label encoder
with open("datasets/penguin_model.pkl", "rb") as f:
    model = pickle.load(f)

with open("datasets/penguin_features.pkl", "rb") as f:
    feature_names = pickle.load(f)

with open("datasets/label_encoder.pkl", "rb") as f:
    label_encoder = pickle.load(f)

```

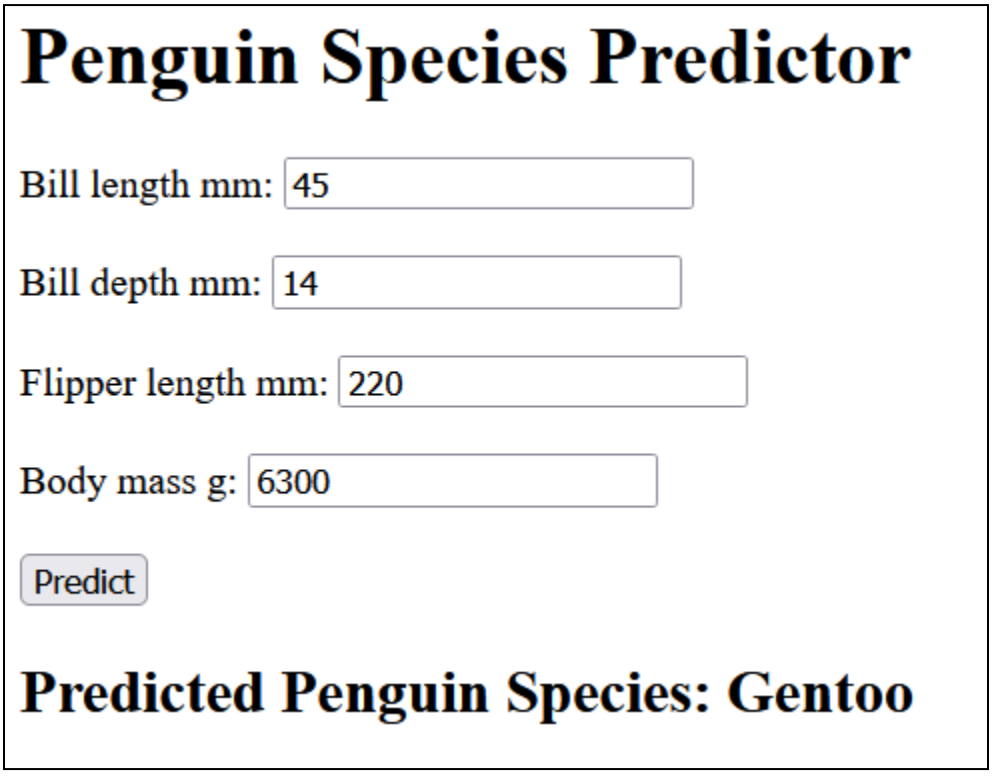
```python
@app.route("/", methods=["GET", "POST"])
def home():
    if request.method == "POST":
        try:
            user_inputs = {feature: request.form.get(feature) for feature in feature_names}

            # Convert inputs to float for prediction
            features = [float(user_inputs[f]) for f in feature_names]

            prediction = model.predict([features])[0]
            species = label_encoder.inverse_transform([prediction])[0]

            return render_template(
                "index.html",
                prediction=f"Predicted Penguin Species: {species}",
                feature_names=feature_names,
                user_inputs=user_inputs
            )
        except ValueError:
            return render_template(
                "index.html",
                error="Please provide valid numeric inputs.",
                feature_names=feature_names,
                user_inputs=request.form
            )

    # Render the initial page with empty inputs
    return render_template("index.html", feature_names=feature_names, user_inputs={})

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

Step 3 (Frontend Development):

A user-friendly HTML form was developed to capture penguin features: bill length, bill depth, flipper length, and body mass.
The form pre-fills the user inputs on submission. Error messages were added to handle invalid or missing inputs.



Filled-in data for classification:

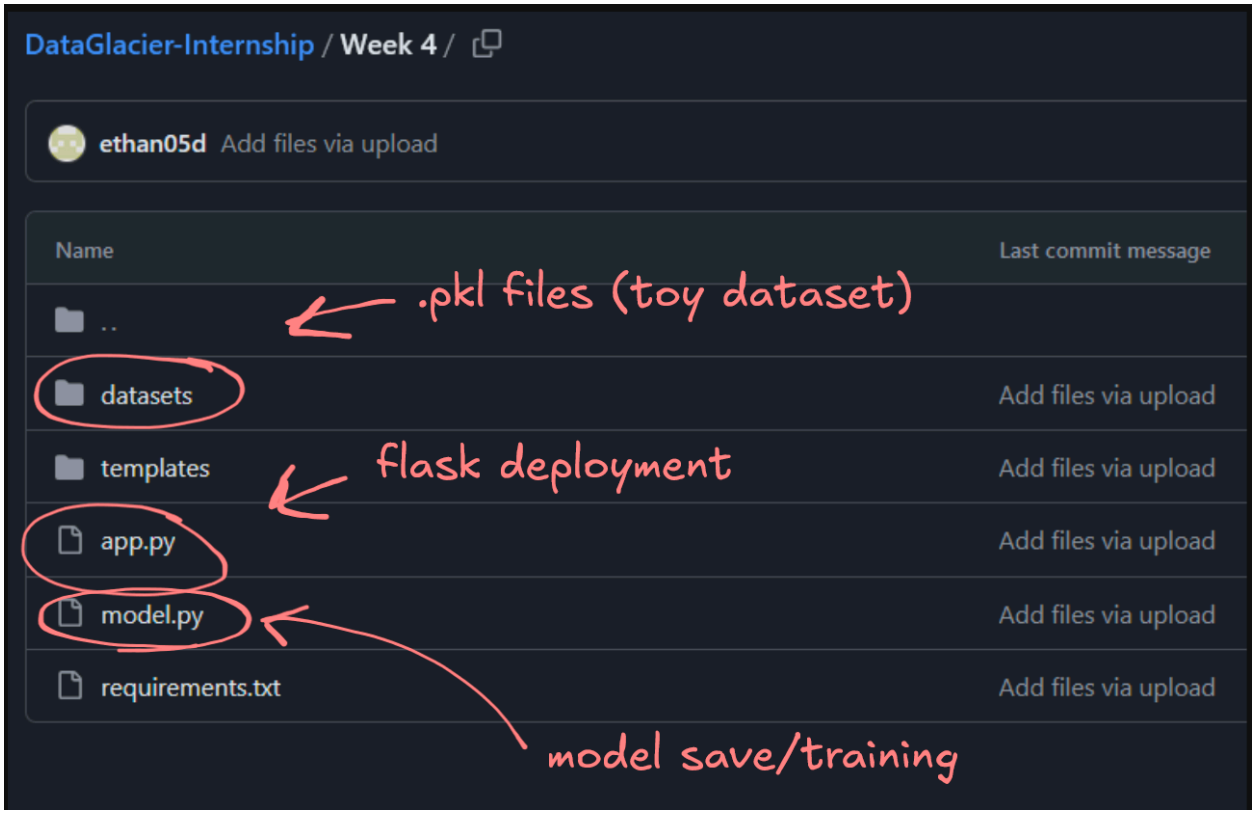


Here is the HTML code utilizing JINJA templates to dynamically display the information:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Penguin Species Predictor</title>
</head>
<body>
    <h1>Penguin Species Predictor</h1>
    <form method="POST">
        {% for feature in feature_names %}
            <label for="{{ feature }}">{{ feature.replace('_', ' ').capitalize() }}:</label>
            <input
                type="text"
                id="{{ feature }}"
                name="{{ feature }}"
                value="{{ user_inputs.get(feature, '') }}"
            ><br><br>
        {% endfor %}
        <button type="submit">Predict</button>
    </form>
    {% if prediction %}
        <h2>{{ prediction }}</h2>
    {% endif %}
    {% if error %}
        <p style="color: red;">{{ error }}</p>
    {% endif %}
</body>
</html>
```

Step 4 (Submission):

Submitted within GitHub -> [Github Repository]


DataGlacier-Internship / Week 4 /
ethan05d Add files via upload
Name
Last commit message
.pkl files (toy dataset)
..
datasets
Add files via upload
templates
Add files via upload
flask deployment
app.py
Add files via upload
model.py
Add files via upload
requirements.txt
Add files via upload
model save/training