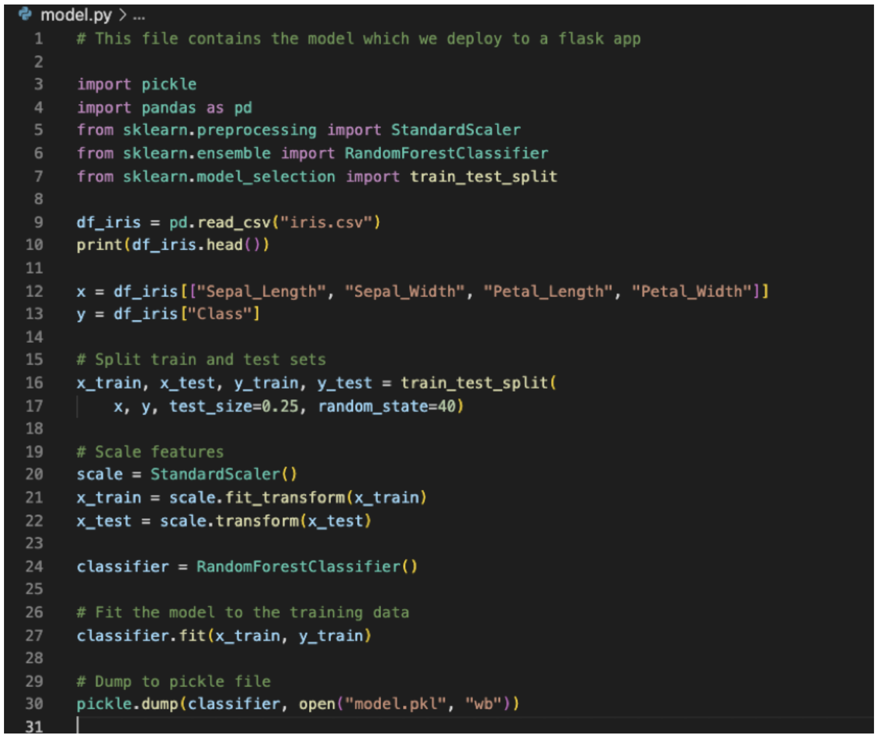
Noah Igram
DataGlacier
LISUM11

Web App Deployment with Heroku

This document outlines the steps that were taken to deploy a Flask web application to the Web using Heroku. The application is a simple web page that displays a flower classification model for a user to interact with. The model uses machine learning to predict a flower's species based on 4 factors: Sepal Width, Sepal Length, Petal Width, and Petal Length.

Flower Classification Model

The model takes the following 4 inputs: Sepal length, Sepal width, Petal length, and Petal Width. It then outputs the predicted flower species out of the following choices: Virginica, Versicolor, Setosa. Below is a snapshot of the python script which creates the model using SkLearn's Random Forest Classifier. The final line of the code uses the pickle module Dump to serialize the data encapsulated in the model.

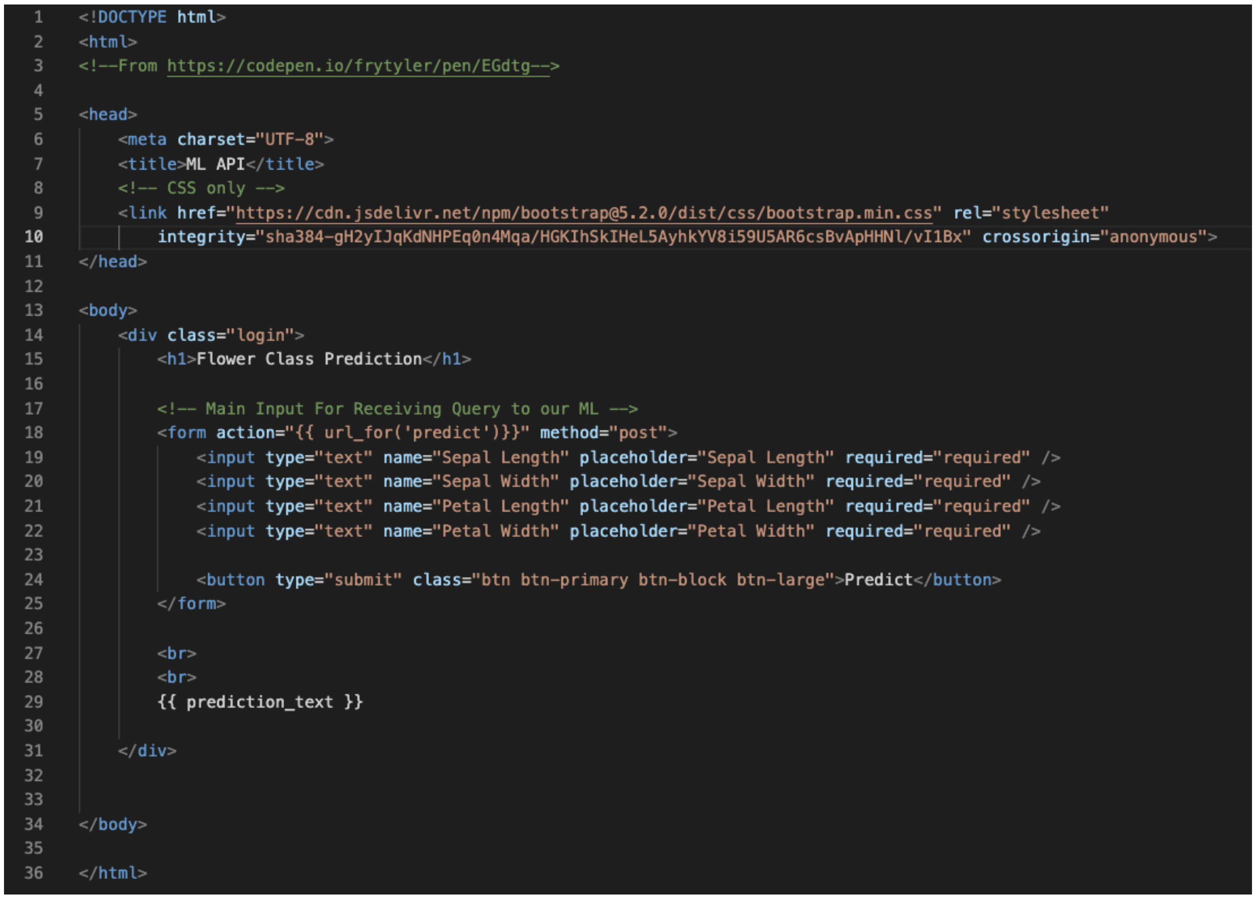


model.py > ...

```python
# This file contains the model which we deploy to a flask app

import pickle
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

df_iris = pd.read_csv("iris.csv")
print(df_iris.head())

x = df_iris[["Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width"]]
y = df_iris["Class"]

# Split train and test sets
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.25, random_state=40)

# Scale features
scale = StandardScaler()
x_train = scale.fit_transform(x_train)
x_test = scale.transform(x_test)

classifier = RandomForestClassifier()

# Fit the model to the training data
classifier.fit(x_train, y_train)

# Dump to pickle file
pickle.dump(classifier, open("model.pkl", "wb"))
```

HTML for Web App

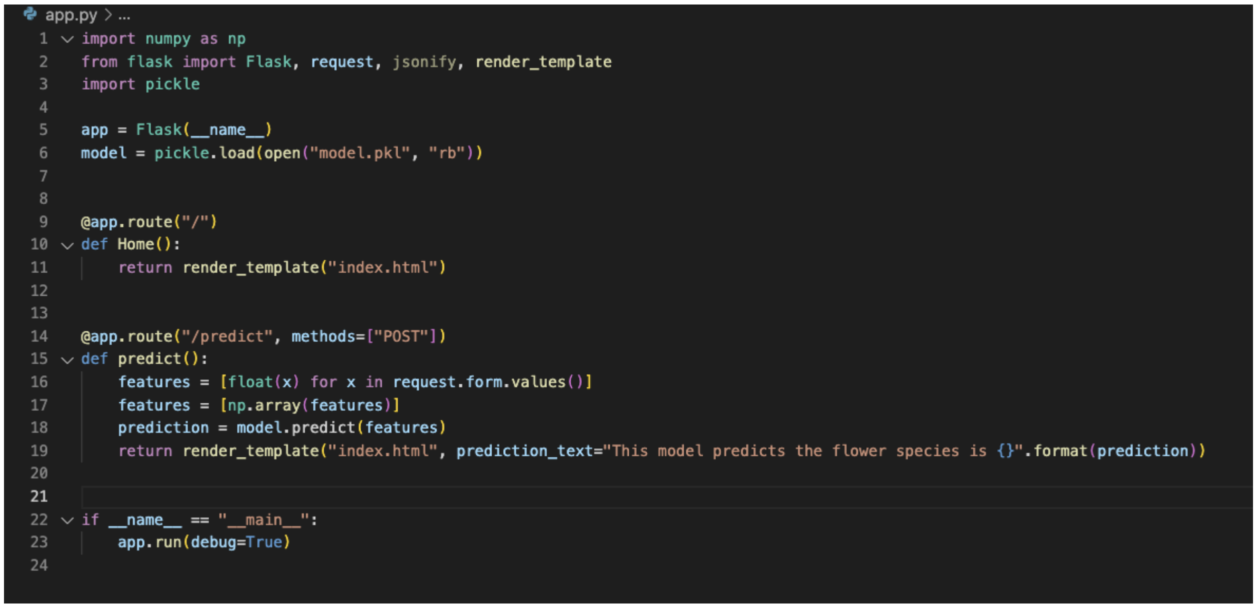
Basic HTML was written to create an interactive web app which allows the user to input custom Sepal length and width, and Petal length and width and receive a prediction from the classification model. Below is a snapshot of the code. Bootstrap was used to help make the website a bit less bland.



```html
<!DOCTYPE html>
<html>
<!--From https://codepen.io/frytyler/pen/EGdtg-->

<head>
    <meta charset="UTF-8">
    <title>ML API</title>
    <!-- CSS only -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/css/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-gH2yIJqKdNHPEq0n4Mqa/HGKIhSkIHeL5AyhkYV8i59U5AR6csBvApHHNl/vI1Bx" crossorigin="anonymous">
</head>

<body>
    <div class="login">
        <h1>Flower Class Prediction</h1>

        <!-- Main Input For Receiving Query to our ML -->
        <form action="{{ url_for('predict')}}" method="post">
            <input type="text" name="Sepal Length" placeholder="Sepal Length" required="required" />
            <input type="text" name="Sepal Width" placeholder="Sepal Width" required="required" />
            <input type="text" name="Petal Length" placeholder="Petal Length" required="required" />
            <input type="text" name="Petal Width" placeholder="Petal Width" required="required" />

            <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
        </form>

        <br>
        <br>
        {{ prediction_text }}

    </div>


</body>

</html>
```

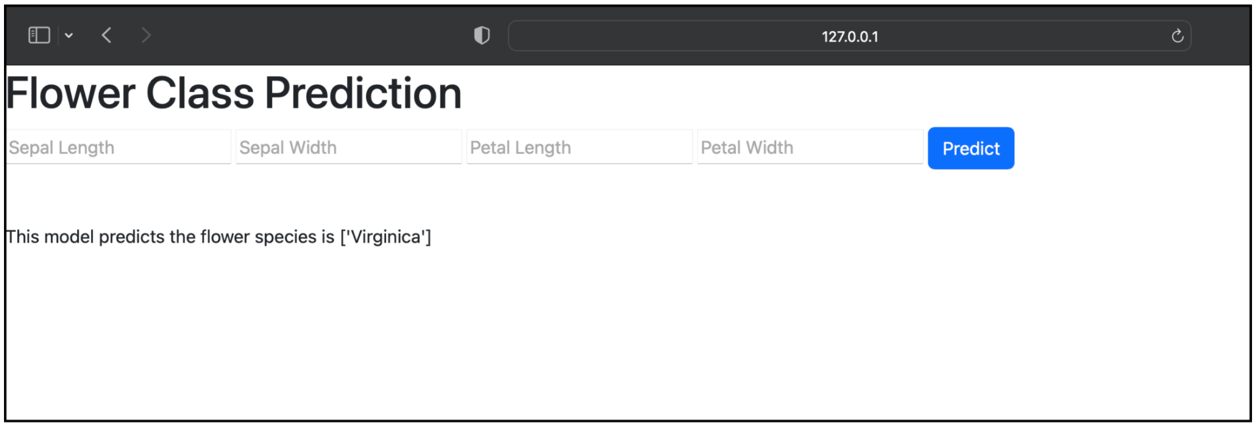
Web Deployment with Flask

Flask was used to deploy the web app to a local server for personal interaction and testing. Below shows the code which makes this happen.



app.py > ...

```python
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle

app = Flask(__name__)
model = pickle.load(open("model.pkl", "rb"))


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


@app.route("/predict", methods=["POST"])
def predict():
    features = [float(x) for x in request.form.values()]
    features = [np.array(features)]
    prediction = model.predict(features)
    return render_template("index.html", prediction_text="This model predicts the flower species is {}".format(prediction))


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

```

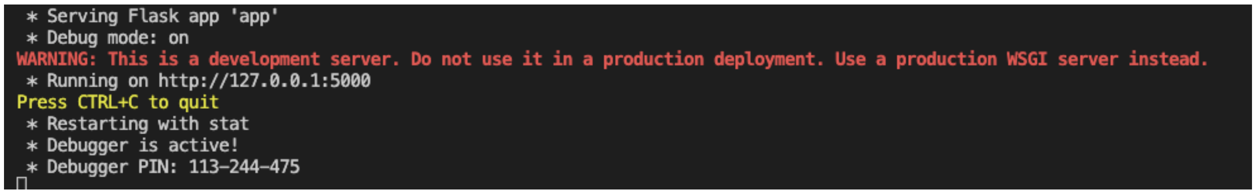


Flower Class Prediction

Sepal Length | Sepal Width | Petal Length | Petal Width | Predict

This model predicts the flower species is ['Virginica']

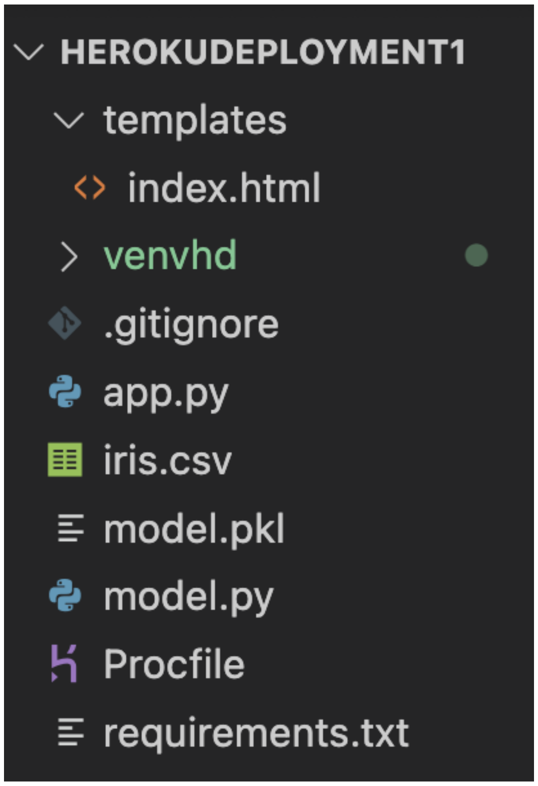
The above image is a snapshot of the web app created. It allows for the user to interact and predict a flower species and was run on a local server. Lastly, below is a snapshot of the terminal which demonstrates what happens when the Flask app is deployed to the local server.



```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 113-244-475
```

Deploying the Flask application with Heroku

Heroku is a service created by Salesforce that allows users to run web applications on their hosted domains. It makes it easy for applications like those created with Flask to be deployed to the Web so that anyone can have access.



```
HEROKUDEPLOYMENT1
  templates
    index.html
  venvhd
  .gitignore
  app.py
  iris.csv
  model.pkl
  model.py
  Procfile
  requirements.txt
```

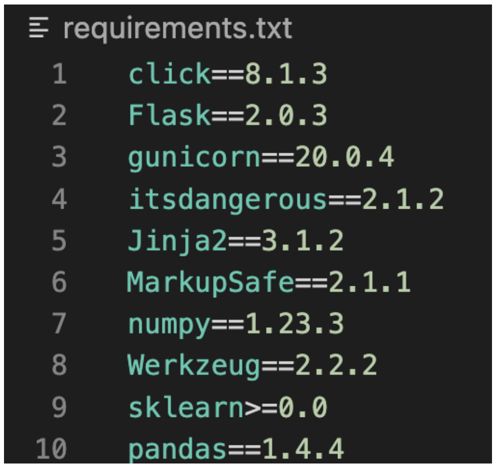
Here is a snapshot of the project folder which contains the Flask application, the model (python script and pickled file) and its accompanying .csv file, the HTML for the web app, as well as two other necessary files for the Heroku deployment. These two files are Procfile and requirements.txt.

Procfile tells Heroku that it will use gunicorn as its Web Server Gateway Interface HTTP server, and specifies that this is a web application. It is only one line of code:

```
web: gunicorn app:app
```

requirements.txt contains the relevant python packages needed to run the web app. This allows Heroku to create the Flask app and use the model with the right package versions, all on its own.

Requirements.txt



```
requirements.txt
click==8.1.3
Flask==2.0.3
gunicorn==20.0.4
itsdangerous==2.1.2
Jinja2==3.1.2
MarkupSafe==2.1.1
numpy==1.23.3
Werkzeug==2.2.2
sklearn>=0.0
pandas==1.4.4
```

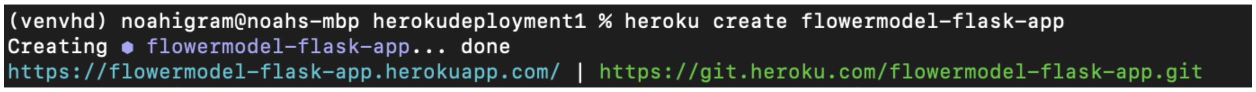
Here is a snapshot of requirements.txt, which contains the necessary Python packages and versions to allow the Flask app to be successfully deployed with Heroku.

Deploying with the Heroku Command Line Interface

Heroku.com allows users to deploy by connecting to a github account and deploying from that repository, however I decided to use the Heroku CLI for a surefire way of successfully deploying the app to the Web.

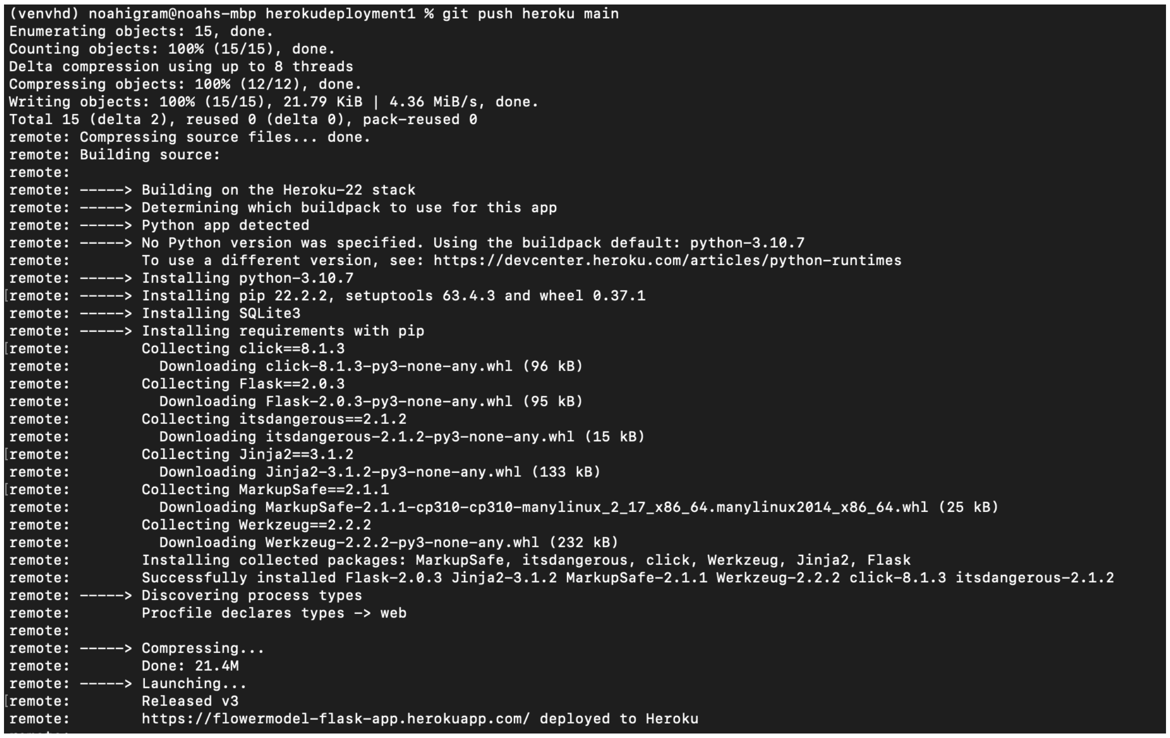
After creating a local git repository, git was used with the Heroku CLI to deploy the web app to https://flowermodel-flask-app.com/

First the Heroku app was created:



```
(venvhd) noahigram@noahs-mbp herokudeployment1 % heroku create flowermodel-flask-app
Creating ● flowermodel-flask-app... done
https://flowermodel-flask-app.herokuapp.com/ | https://git.heroku.com/flowermodel-flask-app.git
```

Next, git was used to push the local repository to the Heroku repository:



```
(venvhd) noahigram@noahs-mbp herokudeployment1 % git push heroku main
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 8 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (15/15), 21.79 KiB | 4.36 MiB/s, done.
Total 15 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Building on the Heroku-22 stack
remote: -----> Determining which buildpack to use for this app
remote: -----> Python app detected
remote: -----> No Python version was specified. Using the buildpack default: python-3.10.7
remote:        To use a different version, see: https://devcenter.heroku.com/articles/python-runtimes
remote: -----> Installing python-3.10.7
[remote: -----> Installing pip 22.2.2, setuptools 63.4.3 and wheel 0.37.1
remote: -----> Installing SQLite3
remote: -----> Installing requirements with pip
[remote:        Collecting click==8.1.3
remote:          Downloading click-8.1.3-py3-none-any.whl (96 kB)
remote:        Collecting Flask==2.0.3
remote:          Downloading Flask-2.0.3-py3-none-any.whl (95 kB)
remote:        Collecting itsdangerous==2.1.2
remote:          Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
[remote:        Collecting Jinja2==3.1.2
remote:          Downloading Jinja2-3.1.2-py3-none-any.whl (133 kB)
[remote:        Collecting MarkupSafe==2.1.1
remote:          Downloading MarkupSafe-2.1.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (25 kB)
remote:        Collecting Werkzeug==2.2.2
remote:          Downloading Werkzeug-2.2.2-py3-none-any.whl (232 kB)
remote:        Installing collected packages: MarkupSafe, itsdangerous, click, Werkzeug, Jinja2, Flask
remote:        Successfully installed Flask-2.0.3 Jinja2-3.1.2 MarkupSafe-2.1.1 Werkzeug-2.2.2 click-8.1.3 itsdangerous-2.1.2
remote: -----> Discovering process types
remote:        Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:        Done: 21.4M
remote: -----> Launching...
[remote:        Released v3
remote:        https://flowermodel-flask-app.herokuapp.com/ deployed to Heroku
```

The app was deployed to the desired address, and it can now be accessed by typing heroku open in the terminal, or by navigating to the link above.

Making Changes and Redeploying

Some mistakes were made in originally creating the Flask app with versions of packages that agree with Heroku, so a few changes were made to the local repository throughout this process. Each time, the changes were stages, committed, and pushed to the Heroku repo. Each time the web app was redeployed (for testing purposes) using the same process as shown above.