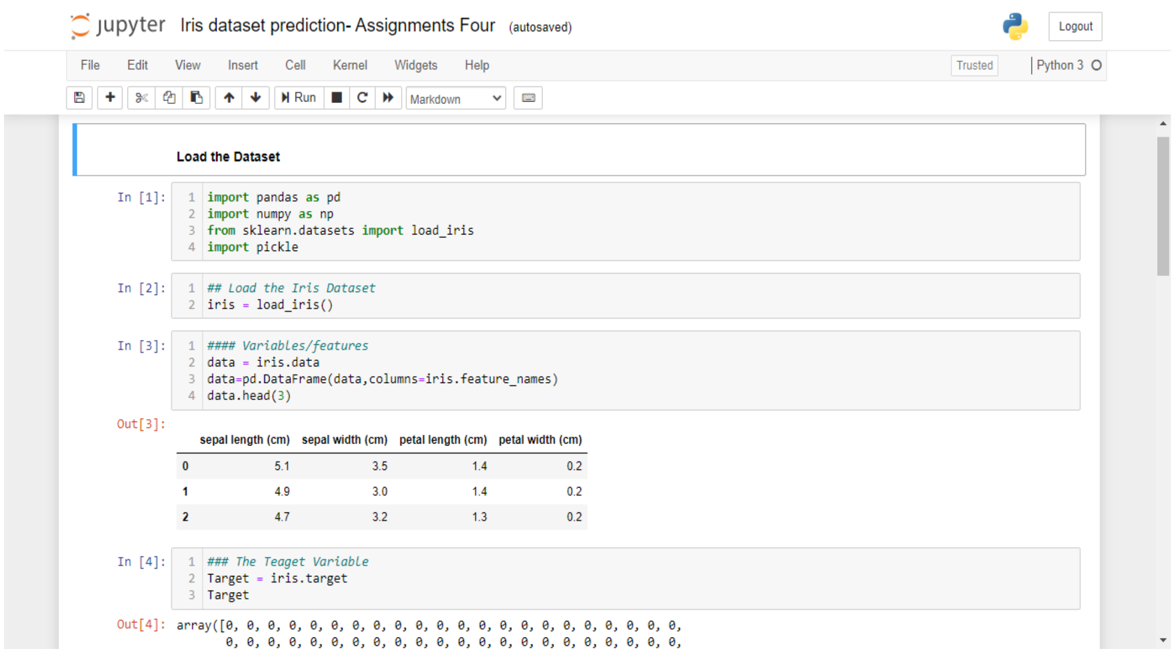
DEPLOYING IN FLASK - ASSIGNMENT 4

Report date:	21/03/2021
Batch code:	LISP01
Version:	1.0
Intern:	Ajaegbu Ebuka Emmanuel
Submitted to:	Data Glacier

STEP BY STEP FOR MODEL DEPLOYMENT

Step One: The model was built in Python using the Random Forest Classifier Algorithm with the help of pandas, NumPy and sklearn



Load the Dataset

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
import pickle
```

In [2]:
```
## Load the Iris Dataset
iris = load_iris()
```

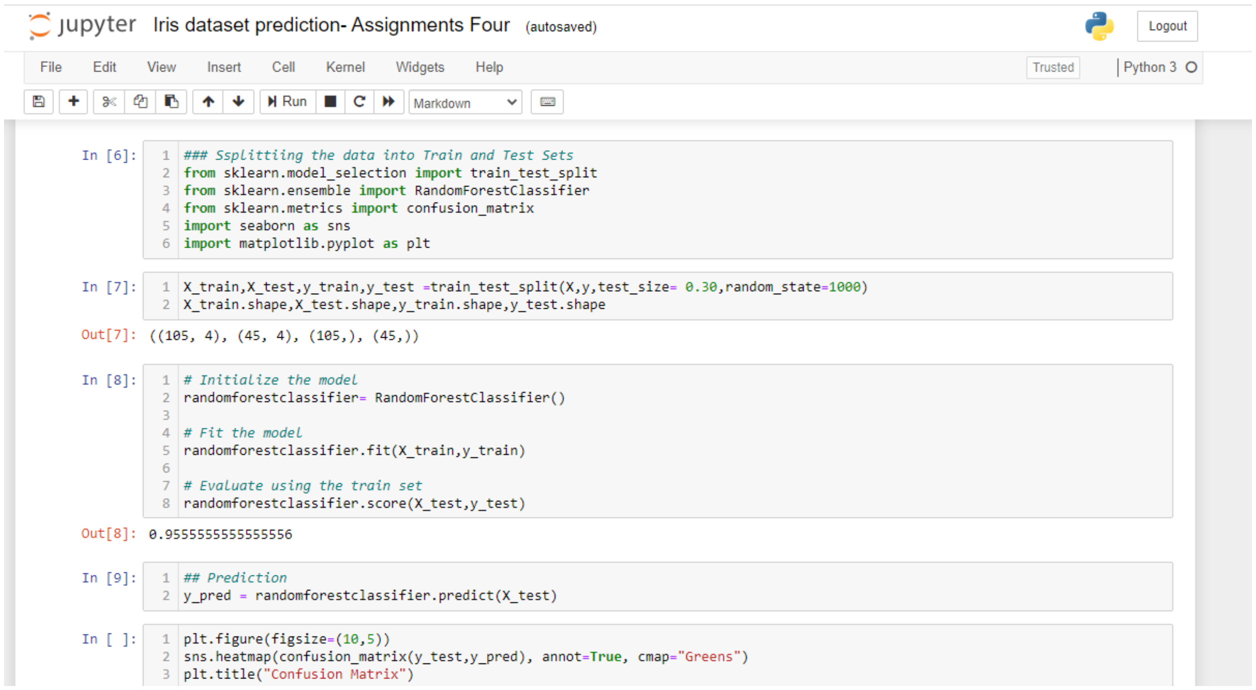
In [3]:
```
#### Variables/features
data = iris.data
data=pd.DataFrame(data,columns=iris.feature_names)
data.head(3)
```

Out[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2

In [4]:
```
### The Teaget Variable
Target = iris.target
Target
```

Out[4]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```


jupyter Iris dataset prediction- Assignments Four (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown

In [6]:
```
### Splitting the data into Train and Test Sets
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

In [7]:
```
X_train,X_test,y_train,y_test =train_test_split(X,y,test_size= 0.30,random_state=1000)
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[7]:
```
((105, 4), (45, 4), (105,), (45,))
```

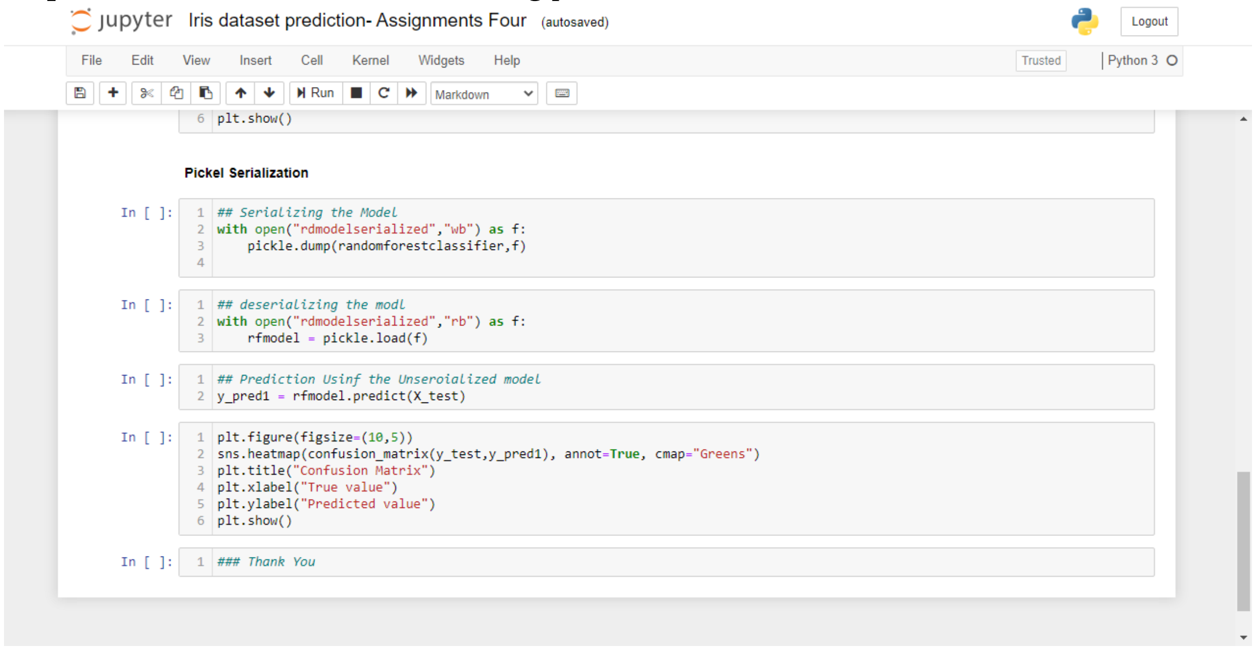
In [8]:
```
# Initialize the model
randomforestclassifier= RandomForestClassifier()

# Fit the model
randomforestclassifier.fit(X_train,y_train)

# Evaluate using the train set
randomforestclassifier.score(X_test,y_test)
```

Out[8]:
```
0.9555555555555556
```

In [9]:
```
## Prediction
y_pred = randomforestclassifier.predict(X_test)
```

In [ ]:
```
plt.figure(figsize=(10,5))
sns.heatmap(confusion_matrix(y_test,y_pred), annot=True, cmap="Greens")
plt.title("Confusion Matrix")
```

Step two: The model was serialized using pickle modules


jupyter Iris dataset prediction- Assignments Four (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown

```
plt.show()
```

Pickel Serialization

In [ ]:
```
## Serializing the Model
with open("rdmodelserialized","wb") as f:
    pickle.dump(randomforestclassifier,f)

```

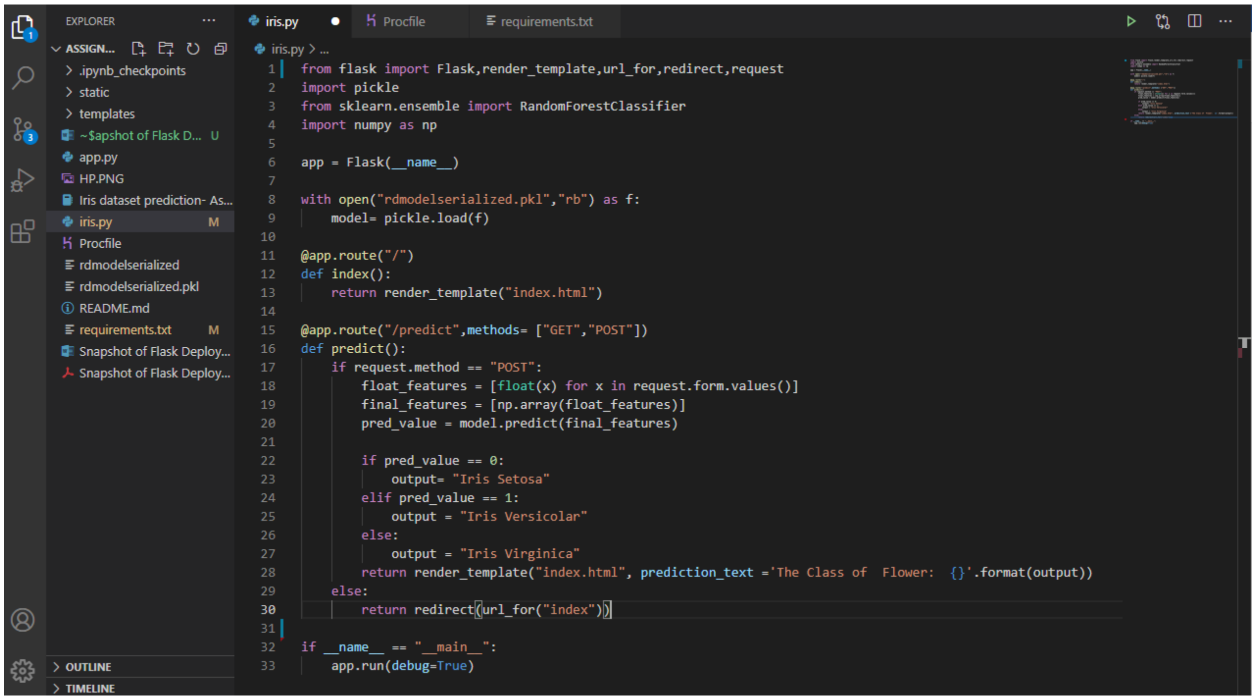
In [ ]:
```
## deserializing the modl
with open("rdmodelserialized","rb") as f:
    rfmodel = pickle.load(f)
```

In [ ]:
```
## Prediction Usinf the Unseroialized model
y_pred1 = rfmodel.predict(X_test)
```

In [ ]:
```
plt.figure(figsize=(10,5))
sns.heatmap(confusion_matrix(y_test,y_pred1), annot=True, cmap="Greens")
plt.title("Confusion Matrix")
plt.xlabel("True value")
plt.ylabel("Predicted value")
plt.show()
```

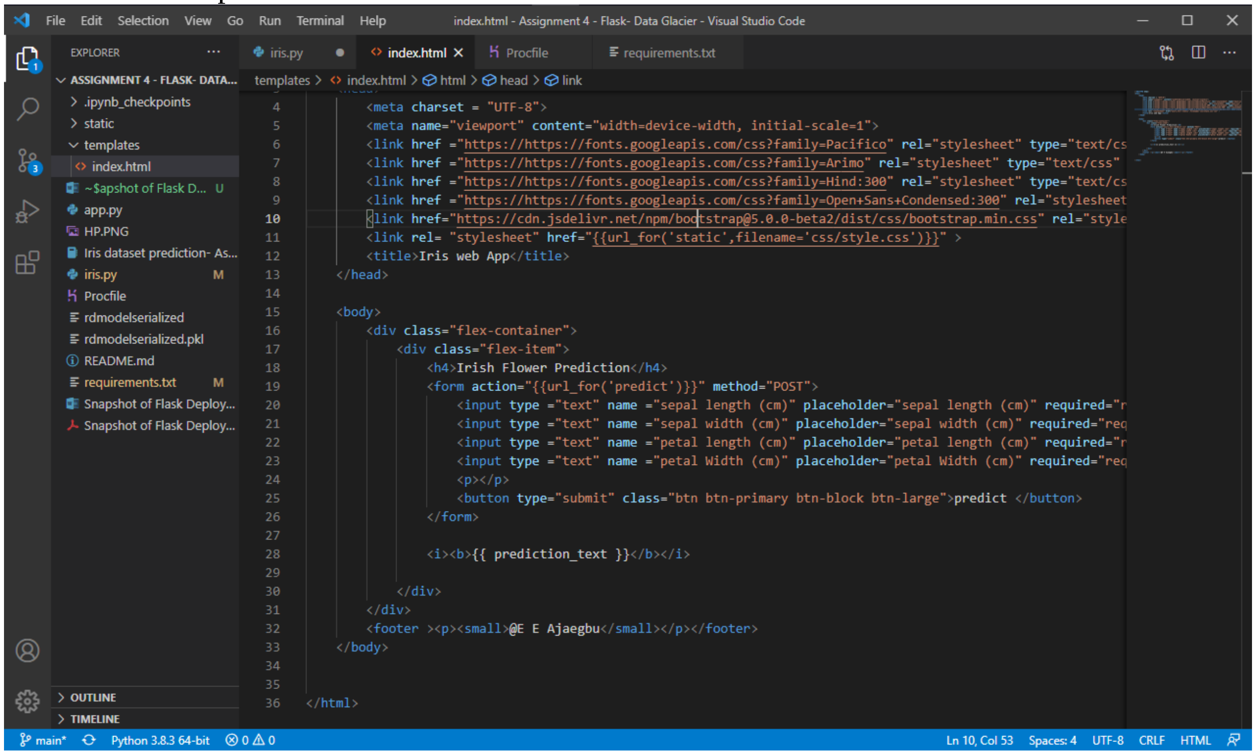
In [ ]:
```
### Thank You
```

Step Three: Flask app was built with the help of the Flask, render_emplate, request, url_for methods. The serialized model was then imported and unserialized and the used for prediction



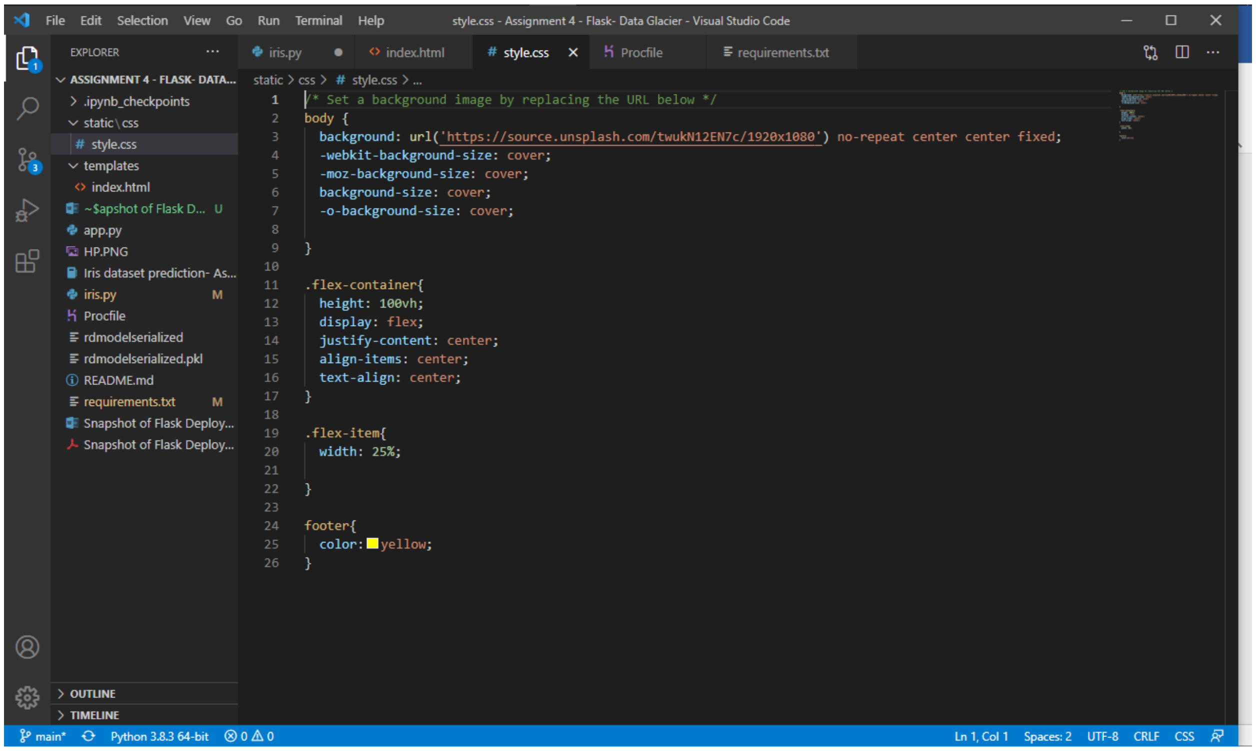
```python
from flask import Flask,render_template,url_for,redirect,request
import pickle
from sklearn.ensemble import RandomForestClassifier
import numpy as np

app = Flask(__name__)

with open("rdmodelserialized.pkl","rb") as f:
    model= pickle.load(f)

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

@app.route("/predict",methods= ["GET","POST"])
def predict():
    if request.method == "POST":
        float_features = [float(x) for x in request.form.values()]
        final_features = [np.array(float_features)]
        pred_value = model.predict(final_features)

        if pred_value == 0:
            output= "Iris Setosa"
        elif pred_value == 1:
            output = "Iris Versicolar"
        else:
            output = "Iris Virginica"
        return render_template("index.html", prediction_text ='The Class of  Flower:  {}'.format(output))
    else:
        return redirect(url_for("index"))

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

Step Four: The index.html file was created for building the structure of the websites. this file is stored in the templates folder



```html
        <meta charset = "UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link href ="https://https://fonts.googleapis.com/css?family=Pacifico" rel="stylesheet" type="text/cs
        <link href ="https://https://fonts.googleapis.com/css?family=Arimo" rel="stylesheet" type="text/css"
        <link href ="https://https://fonts.googleapis.com/css?family=Hind:300" rel="stylesheet" type="text/cs
        <link href ="https://https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300" rel="stylesheet
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/css/bootstrap.min.css" rel="style
        <link rel= "stylesheet" href="{{url_for('static',filename='css/style.css')}}" >
        <title>Iris web App</title>
    </head>

    <body>
        <div class="flex-container">
            <div class="flex-item">
                <h4>Irish Flower Prediction</h4>
                <form action="{{url_for('predict')}}" method="POST">
                    <input type ="text" name ="sepal length (cm)" placeholder="sepal length (cm)" required="r
                    <input type ="text" name ="sepal width (cm)" placeholder="sepal width (cm)" required="req
                    <input type ="text" name ="petal length (cm)" placeholder="petal length (cm)" required="r
                    <input type ="text" name ="petal Width (cm)" placeholder="petal Width (cm)" required="req
                    <p></p>
                    <button type="submit" class="btn btn-primary btn-block btn-large">predict </button>
                </form>

                <i><b>{{ prediction_text }}</b></i>

            </div>
        </div>
        <footer ><p><small>@E E Ajaegbu</small></p></footer>
    </body>


</html>
```

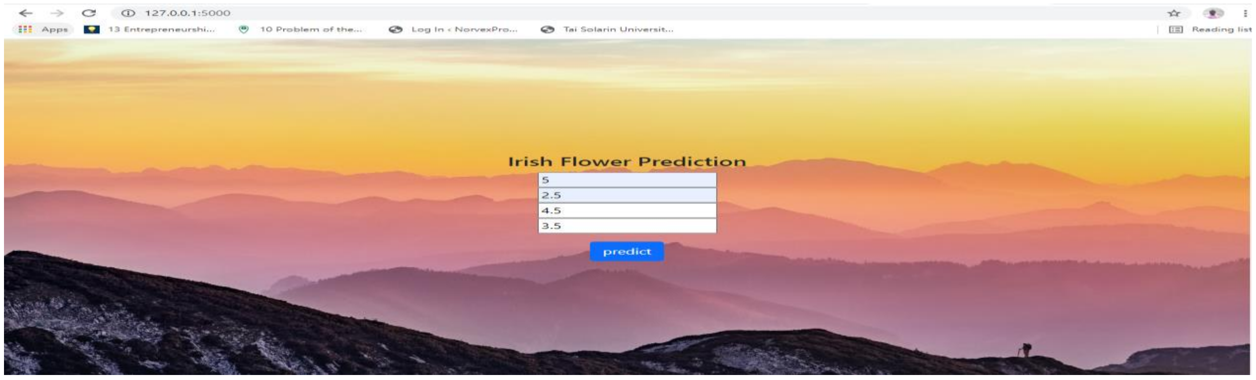
Step Five: The CSS file was created for styling the web app and for also bootstrapping.


File Edit Selection View Go Run Terminal Help
style.css - Assignment 4 - Flask- Data Glacier - Visual Studio Code
EXPLORER
ASSIGNMENT 4 - FLASK- DATA...
.ipynb_checkpoints
static \ css
style.css
templates
index.html
~$apshot of Flask D... U
app.py
HP.PNG
Iris dataset prediction- As...
iris.py M
Procfile
rdmodelserialized
rdmodelserialized.pkl
README.md
requirements.txt M
Snapshot of Flask Deploy...
Snapshot of Flask Deploy...
OUTLINE
TIMELINE
iris.py
index.html
style.css
Procfile
requirements.txt
static > css > # style.css > ...
main* Python 3.8.3 64-bit ⊗ 0 ⚠ 0
Ln 1, Col 1 Spaces: 2 UTF-8 CRLF CSS

```css
/* Set a background image by replacing the URL below */
body {
  background: url('https://source.unsplash.com/twukN12EN7c/1920x1080') no-repeat center center fixed;
  -webkit-background-size: cover;
  -moz-background-size: cover;
  background-size: cover;
  -o-background-size: cover;

}

.flex-container{
  height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
  text-align: center;
}

.flex-item{
  width: 25%;

}

footer{
  color: yellow;
}
```

Step Six: Making Prediction Using the App

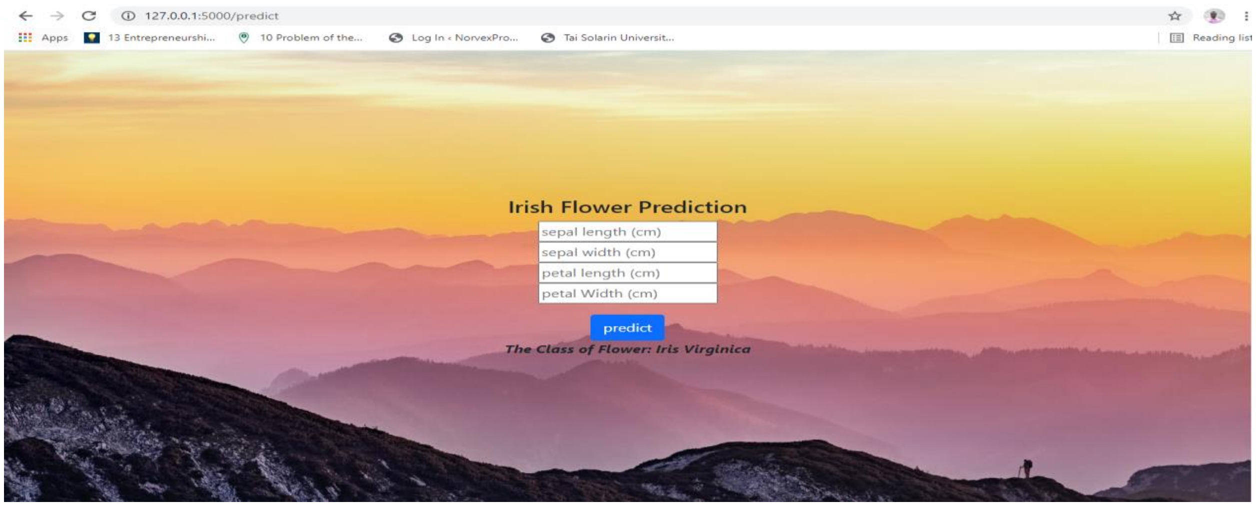
Before



Irish Flower Prediction

5
2.5
4.5
3.5

After



Step Seven: The files will then uploaded to GitHub

Step Eight: The web application will be deployed in the cloud.- Heroku