

Name: Nathan Adam

Batch code: LISUM01

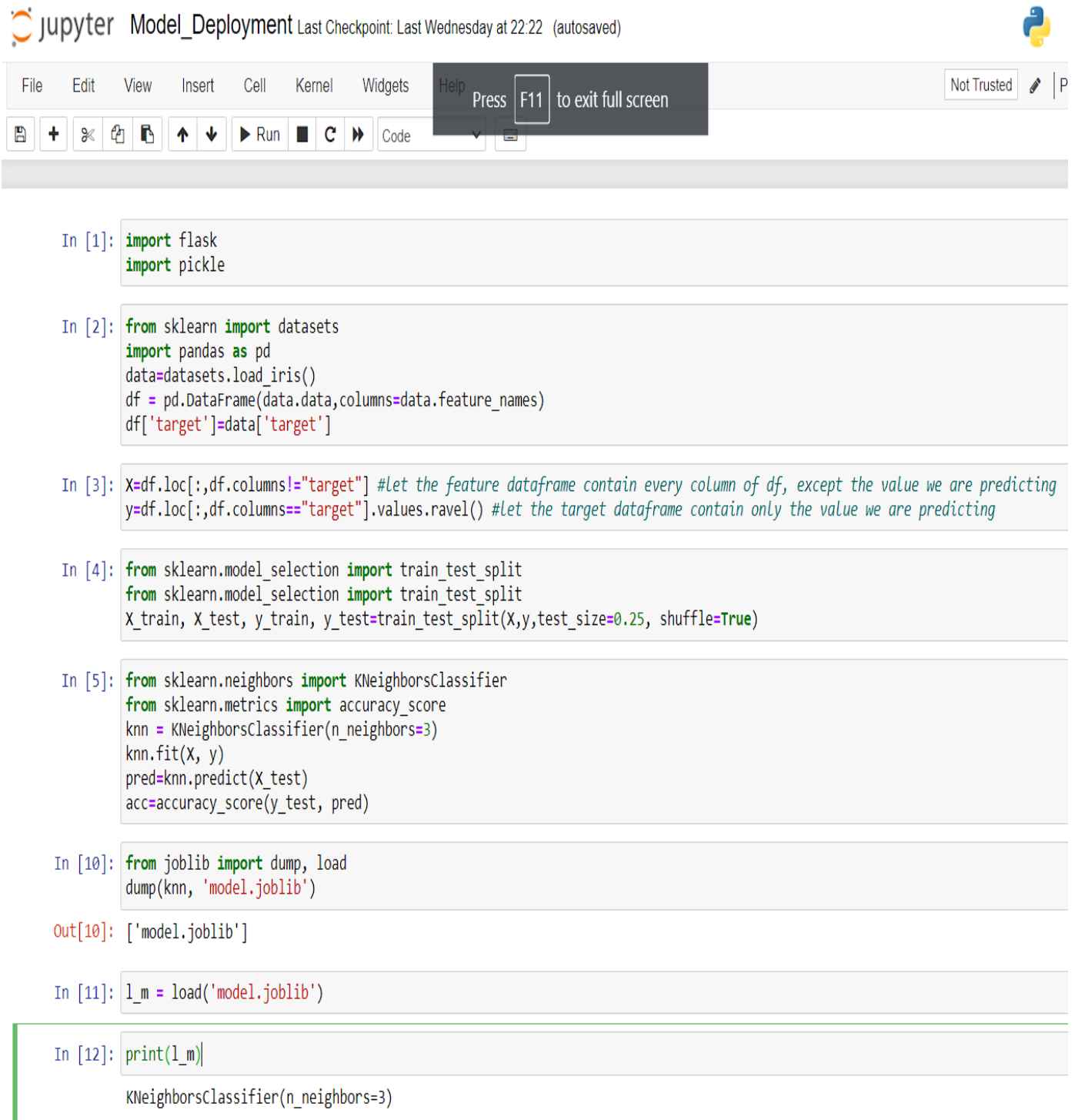
Submission date: 7th July 2021

Submitted to: Week 5: Cloud and API deployment

https://github.com/N-A-ML/Data_Glacier_Cloud_and_API_Deployment_Week_5 (on GitHub)

Note: the app was deployed on the cloud with Heroku in week 4.

Select data (iris dataset), create and save a simple model (knn classifier):



The image shows a Jupyter Notebook interface with the title 'Model_Deployment'. The top bar includes the Jupyter logo, the title, and a status message 'Last Checkpoint: Last Wednesday at 22:22 (autosaved)'. On the right, there is a 'Not Trusted' warning and a Python logo. The menu bar includes File, Edit, View, Insert, Cell, Kernel, and Widgets. A tooltip indicates 'Press F11 to exit full screen'. The toolbar contains icons for saving, adding cells, running, and other standard Jupyter actions. The code area contains 12 input cells:

```
In [1]: import flask
import pickle

In [2]: from sklearn import datasets
import pandas as pd
data=datasets.load_iris()
df = pd.DataFrame(data.data,columns=data.feature_names)
df['target']=data['target']

In [3]: X=df.loc[:,df.columns!="target"] #let the feature dataframe contain every column of df, except the value we are predicting
y=df.loc[:,df.columns=="target"].values.ravel() #let the target dataframe contain only the value we are predicting

In [4]: from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.25, shuffle=True)

In [5]: from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X, y)
pred=knn.predict(X_test)
acc=accuracy_score(y_test, pred)

In [10]: from joblib import dump, load
dump(knn, 'model.joblib')

Out[10]: ['model.joblib']

In [11]: l_m = load('model.joblib')

In [12]: print(l_m)

KNeighborsClassifier(n_neighbors=3)
```

Create html and css files:

index.html - Notepad

File Edit Format View Help

```
<html>
<head>
<link rel= "stylesheet" type= "text/css" href= "{{ url_for('static',filename='styles/styles.css') }}">
<link rel="stylesheet" type="text/css" href="//fonts.googleapis.com/css?family=Playfair+Display" />
<title> Predict the type of iris flower </title>
</head>

<body>
<h1> Predict the type of iris flower (Setosa, Versicolor, or Virginica) using a K nearest neighbors classifier (k=3)</h1>
<div class="wrapper">
<div class="form">

<form action = "{{ url_for('predict')}}" method="post">
    <input type="text" name="sepal_length" placeholder= "Sepal Length(cm)" required="required" /> <br> <br>
    <input type="text" name="sepal_width" placeholder= "Sepal Width(cm)" required="required" /> <br> <br>
    <input type="text" name="petal_length" placeholder= "Petal Length(cm)" required="required" /> <br> <br>
    <input type="text" name="petal_width" placeholder= "Petal Width(cm)" required="required" /> <br> <br>
    <button type="submit"> Predict </button>

|
<br>
<br>
{{ prediction_text }}
</form>

</div>
    <div class="image">
    
    </div>

</div>
</body>
</html>
```

*styles.css - Notepad

File Edit Format View Help

```
* {
    font-family:"Playfair Display";
}
body {
    background-color: lightblue;
}

h1 {
font-size:3.5em;
margin-left:5%;
margin-right:5%;
}

form input, button {
font-size:1.5em;
}
form {
font-size:1.5em;
}


.wrapper {
display:flex;
justify-content:space-evenly;
margin-top:5%;
}

.form {

}
|
.image img{
    max-height:50vh;
    height:auto;
    width:auto;
}

}
```

Use flask so the web app can be deployed locally. Images are included in the app:












 jupyter Deployment_flask.py a few seconds ago

File Edit View Language

Press **F11** to exit full screen

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[ ]:
5  import numpy as np
6  from flask import Flask, request, render_template
7  import joblib
8  from joblib import load
9  from sklearn.neighbors import KNeighborsClassifier
10 import os
11 images_folder=os.path.join('static', 'images')
12 app=Flask(__name__)
13 app.config['UPLOAD_FOLDER'] = images_folder
14 model=load('model.joblib')
15
16 @app.route('/')
17 def home():
18     return render_template('index.html')
19 @app.route('/predict', methods=['POST'])
20 def predict():
21     features=[float(x) for x in request.form.values()]
22     final_features=[np.array(features)]
23     prediction=model.predict(final_features)
24     pred_round=round(prediction[0])
25     output=""
26     if pred_round==0:
27         output+="Setosa"
28         file = os.path.join(app.config['UPLOAD_FOLDER'], 'setosa.jpg')
29     elif pred_round==1:
30         output+="Versicolor"
31         file = os.path.join(app.config['UPLOAD_FOLDER'], 'versicolor.jpg')
32     else:
33         output+="Virginica"
34         file = os.path.join(app.config['UPLOAD_FOLDER'], 'virginica.jpg')
35
36     return render_template('index.html', prediction_text='This iris flower is {}'.format(output),
37                           iris=file
38                           )
39 if __name__=="__main__":
40     app.run(port=5000, debug=True, use_reloader=False)
41 |
42 # In[16]:
43
```

Generate Procfile (and enter the name of the app), requirements.txt and runtime.txt, and structure the files and folders correctly:

Add folder in library ▾		Give access to ▾	New folder	
Name		Date modified	Type	Size
 .git		02/07/2021 19:14	File folder	
 .ipynb_checkpoints		30/06/2021 20:30	File folder	
 static		02/07/2021 17:04	File folder	
 templates		30/06/2021 21:58	File folder	
 Deployment_flask.ipynb		02/07/2021 19:30	IPYNB File	5 KB
 Deployment_flask.py		02/07/2021 19:30	Python File	2 KB
 model.joblib		30/06/2021 22:08	JOBLIB File	14 KB
 Model_Deployment.ipynb		30/06/2021 22:22	IPYNB File	7 KB
 Procfile		01/07/2021 17:28	File	1 KB
 requirements.txt		01/07/2021 18:00	Text Document	1 KB
 runtime.txt		01/07/2021 17:29	Text Document	1 KB

Ensure that relevant packages are installed in the working directory (e.g., gunicorn) and upload the files and folders to GitHub. Link the GitHub repository to Heroku and troubleshoot any problems by checking the logs.

Salesforce Platform

HEROKU

Jump to Favorites, Apps, Pipelines, Spaces...

Personal



>

predict3iris

☆

Open app

More

GitHub  N-A-ML/Data_Glacier_Deployment_on_Flask_Week_4  main

Overview Resources **Deploy** Metrics Activity Access Settings

Add this app to a pipeline

Create a new pipeline or choose an existing one and add this app to a stage in it.

Add this app to a stage in a pipeline to enable additional features

Pipelines let you connect multiple apps together and promote code between them.


[Learn more.](#)


Pipelines connected to GitHub can enable review apps, and create apps for new pull requests.


[Learn more.](#)

Choose a pipeline

Deployment method



 Heroku Git
Use Heroku CLI

 GitHub
Connected


 Container Registry
Use Heroku CLI



App connected to GitHub

Code diffs, manual and auto deploys are available for this app.


Connected to  N-A-ML/Data_Glacier_Deployment_on_Flask_Week_4 by  N-A-ML

Disconnect...

 Releases in the [activity feed](#) link to GitHub to view commit diffs

 Automatically deploys from  main


Automatic deploys

 You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please

Finally, launch the app on the cloud with Heroku and test it:


Predict the type of iris flower (Setosa, Versicolor, or Virginica) using a K nearest neighbors classifier (k=3)

This iris flower is Setosa



The app is working as intended.

A simple python file was created. This will help us test whether the predictions are working as expected:

 jupyter File_for_testing_output_in_Postman.py a few seconds ago

```
File Edit View Language

5 import numpy as np
6 import pandas as pd
7 from flask import Flask, jsonify, request, render_template
8 import joblib
9 from joblib import load
10 from sklearn.neighbors import KNeighborsClassifier
11 import os
12 app=Flask(__name__)
13
14 @app.route('/', methods= ['GET','POST'])
15 def home():
16     if(request.method=="GET"):
17         data= "hello world"
18         return jsonify({'data':data})
19 @app.route('/predict/')
20 def predict():
21     model=load('model.joblib')
22     sepal_length=request.args.get('sepal_length')
23     sepal_width=request.args.get('sepal_width')
24     petal_length=request.args.get('petal_length')
25     petal_width=request.args.get('petal_width')
26
27     test_df= pd.DataFrame({'sepal length':[sepal_length], 'sepal width':[sepal_width], 'petal length':[petal_length], 'petal width':
[petal_width]})
28
29     model_prediction=model.predict(test_df)
30     model_prediction=np.around(model_prediction, 2)
31
32     output=""
33     if model_prediction[0]==0:
34         output+="Setosa"
35
36     elif model_prediction[0]==1:
37         output+="Versicolor"
38
39     else:
40         output+="Virginica"
41
42
43     return jsonify({'Iris Type': str(output)})
44 if __name__=="__main__":
45     app.run(debug=True,
46             use_reloader=False
47     )
```

I used Postman to check whether each page and each type of prediction was working correctly:

Homepage:

A screenshot of the Postman application showing a GET request to the URL `http://127.0.0.1:5000/`. The interface includes a top bar with a 'Save' button and a 'Send' button. Below the URL bar, there are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Params' tab is active, showing a table with columns 'KEY', 'VALUE', and 'DESCRIPTION'. The table contains one row with 'Key' and 'Value'. The 'Body' tab is also visible, showing a JSON response: `{ "data": "hello world" }`. The status bar at the bottom indicates 'Status: 200 OK', 'Time: 7 ms', and 'Size: 173 B'.

KEY	VALUE	DESCRIPTION
Key	Value	Description

```
{
  "data": "hello world"
}
```

Predictions for the 3 different types of iris:

A screenshot of the Postman application showing a GET request to the URL `http://127.0.0.1:5000/predict/?sepal_length=4.5&sepal_width=2.3&petal_length=1.3&petal_width=0.3`. The interface includes a top bar with a 'Save' button and a 'Send' button. Below the URL bar, there are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Params' tab is active, showing a table with columns 'KEY', 'VALUE', and 'DESCRIPTION'. The table contains four rows with 'sepal_length', 'sepal_width', 'petal_length', and 'petal_width'. The 'Body' tab is also visible, showing a JSON response: `{ "Iris Type": "Setosa" }`. The status bar at the bottom indicates 'Status: 200 OK', 'Time: 9 ms', and 'Size: 173 B'.

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> sepal_length	4.5	
<input checked="" type="checkbox"/> sepal_width	2.3	
<input checked="" type="checkbox"/> petal_length	1.3	
<input checked="" type="checkbox"/> petal_width	0.3	
Key	Value	Description

```
{
  "Iris Type": "Setosa"
}
```

http://127.0.0.1:5000/predict/?sepal_length=5.6&sepal_width=2.5&petal_length=3.9&petal_width=1.1

Save



GET http://127.0.0.1:5000/predict/?sepal_length=5.6&sepal_width=2.5&petal_length=3.9&petal_width=1.1

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	sepal_length	5.6			
<input checked="" type="checkbox"/>	sepal_width	2.5			
<input checked="" type="checkbox"/>	petal_length	3.9			
<input checked="" type="checkbox"/>	petal_width	1.1			
	Key	Value	Description		

Body Cookies Headers (4) Test Results

Status: 200 OK Time: 8 ms Size: 177 B Save Response

Pretty Raw Preview Visualize JSON



```
1
2  "Iris Type": "Versicolor"
3
```

Bootcamp Runner Trash

http://127.0.0.1:5000/predict/?sepal_length=5.7&sepal_width=2.5&petal_length=5&petal_width=2

Save



GET http://127.0.0.1:5000/predict/?sepal_length=5.7&sepal_width=2.5&petal_length=5&petal_width=2

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	sepal_length	5.7			
<input checked="" type="checkbox"/>	sepal_width	2.5			
<input checked="" type="checkbox"/>	petal_length	5			
<input checked="" type="checkbox"/>	petal_width	2			
	Key	Value	Description		

Body Cookies Headers (4) Test Results

Status: 200 OK Time: 7 ms Size: 176 B Save Response

Pretty Raw Preview Visualize JSON



```
1
2  "Iris Type": "Virginica"
3
```

Bootcamp Runner Trash

The iris types were predicted correctly and everything is working as intended.