Data Science Internship at Data Glacier

Week 4: Deployment on Flask

Name: Jesumbo Joseph Oludipe

Batch code: LISUM19

Submission date: 4th April 2023

Submitted to: Data Glacier

Table of Contents

1. Introduction

In this project, a machine-learning model was deployed using the Flask Framework. The Titanic passenger data (name, age, price of the ticket, etc) is used to try to predict who will survive and who will die.

2. Data Information

The data was obtained from Kaggle. There are two files in the data:

1. train.csv

train.csv contains the details of a subset of 891 passengers on board the Titanic ship.

```
In [2]:
train_data = pd.read_csv ("/kaggle/input/titanic/train.csv")
train_data.head()
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Figure 1. Training dataset

2. test.csv

Using the patterns found in train.csv, a prediction of whether the other 418 passengers on board (in test.csv) survived will be. The “Survived” column is not present in the test data.

```
In [3]:
test_data = pd.read_csv("/kaggle/input/titanic/test.csv")
test_data.head()
```

Out[3]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

Figure 2. Test dataset

3. Building a Model

3.1 Import Required Libraries and Dataset

In this part, libraries and datasets which contain the training data are imported

In [1]:

```
# Import pakages and libraries

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
#
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

Figure 3. Importing packages and libraries

In [2]:

```
#Importing the model training dataset
train_data = pd.read_csv ("/kaggle/input/titanic/train.csv")
train_data.head()
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Figure 4. Importing the training dataset.

In [3]:

```
#Importing the test dataset
test_data = pd.read_csv("/kaggle/input/titanic/test.csv")
test_data.head()
```

Out[3]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

Figure 5. Importing the test dataset.

3.2 Exploring Patterns

The code below calculates the percentage of female and male passengers (in **train.csv**) who survived.

In [4]:
```
#exploring patterns
#percentage of woemn that survived
women = train_data.loc[train_data.Sex == 'female']["Survived"]
rate_women = sum(women)/len(women)

print("% of women who survived:", rate_women)
```

```
% of women who survived: 0.7420382165605095
```

In [5]:
```
#percentage of men that survived
men = train_data.loc[train_data.Sex == 'male']["Survived"]
rate_men = sum(men)/len(men)

print("% of men who survived:", rate_men)
```

```
% of men who survived: 0.18890814558058924
```

Figure 6. Exploring the survival patterns

From this, almost 75% of the women on board survived, whereas only 19% of the men lived to tell about it. But this gender-based submission bases its predictions on only a single column. By considering multiple columns, more complex patterns that can potentially yield better-informed predictions can be discovered.

3.3 Building model

I decided to employ a **random forest model**. This model is constructed of several "trees" that will individually consider each passenger's data and vote on whether the individual survived. Then, the random forest model makes a democratic decision: the outcome with the most votes wins!

The code cell below looks for patterns in four different columns (**"Pclass"**, **"Sex"**, **"SibSp"**, and **"Parch"**) of the data. It constructs the trees in the random forest model based on patterns in the **train.csv** file, before generating predictions for the passengers in **test.csv**. The code also saves these new predictions in a CSV file **submission.csv**.

In [6]:

```
#Using random forest model
from sklearn.ensemble import RandomForestClassifier

y = train_data["Survived"]

features = ["Pclass", "Sex", "SibSp", "Parch"]
X = pd.get_dummies(train_data[features])
X_test = pd.get_dummies(test_data[features])

model = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=1)
model.fit(X, y)
predictions = model.predict(X_test)

output = pd.DataFrame({'PassengerId': test_data.PassengerId, 'Survived': predictions})
output.to_csv('submission.csv', index=False)
print("Your submission was successfully saved!")
```

```
Your submission was successfully saved!
```

Figure 7. Building the Model

3.4 Results

The results of the model are displayed below

Output Data

submission.csv (2.84 kB)

PassengerId	Survived
892	0
893	1
894	0
895	0
896	1
897	0
898	1
899	0
900	1
901	0

Figure 8. The output of the model

4. Turning Model into a Web Application

A web application that consists of a simple web page was developed; it possesses a form that allows the input of passenger information ("Pclass", "Sex", "SibSp", and "Parch"). After submitting the info to the web application, it will render it on a new page that gives the result: “This passenger survived” or “This passenger did not survive!”.

4.1 Index.html

The following are the contents of the index.html file that will render a text form where a user can enter the passenger info.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Passenger Info Form</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
  </head>
  <body>
    <h1>Passenger Info Form</h1>
    <form action="#" method="POST">
      <label for="pclass">Pclass:</label>
      <select id="pclass" name="pclass">
        <option value="1">1st Class</option>
        <option value="2">2nd Class</option>
        <option value="3">3rd Class</option></select
      ><br /><br />

      <label for="sex">Sex:</label>
      <input type="radio" id="male" name="sex" value="male" />
      <label for="male">Male</label>
      <input type="radio" id="female" name="sex" value="female" />
      <label for="female">Female</label><br /><br />

      <label for="sibsp">SibSp:</label>
      <input type="number" id="sibsp" name="sibsp" /><br /><br />

      <label for="parch">Parch:</label>
      <input type="number" id="parch" name="parch" /><br /><br />

      <input type="submit" value="Submit" />
    </form>
  </body>
</html>
```

Figure 9. Index.html

4.2 Style.css

In the header section of home.html, we loaded the style.css file. CSS is to determine how the look and feel of HTML documents. style.css has to be saved in a sub-directory called static, which is the default directory where Flask looks for static files such as CSS.

```css
body {
  font-family: Arial, sans-serif;
  background-color: #f2f2f2;
  color: #333;
  margin: 0;
  padding: 0;
}

h1 {
  text-align: center;
}

form {
  margin: 0 auto;
  width: 50%;
  border: 2px solid #ccc;
  padding: 20px;
}

label {
  display: block;
  font-weight: bold;
  margin-bottom: 10px;
}

input[type="radio"] {
  margin-right: 5px;
}

input[type="number"] {
  width: 50px;
}
```

Figure 10. Style.css

4.3 Result.html

There is also a result.html that gives the result if “This passenger survived” or “This passenger did not survive!”.

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Survival Results</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
  </head>
  <body>
    <h1>Survival Results</h1>
    <div class="result">
      <p>This Passenger Survived!</p>
    </div>
  </body>
</html>
```

Figure 11. Result.html

4.4 Running Procedure

Once we have done all of the above, we can start running the API. Now we could open a web browser and navigate to http://127.0.0.1:5000/, we should see a simple website with the content as below.

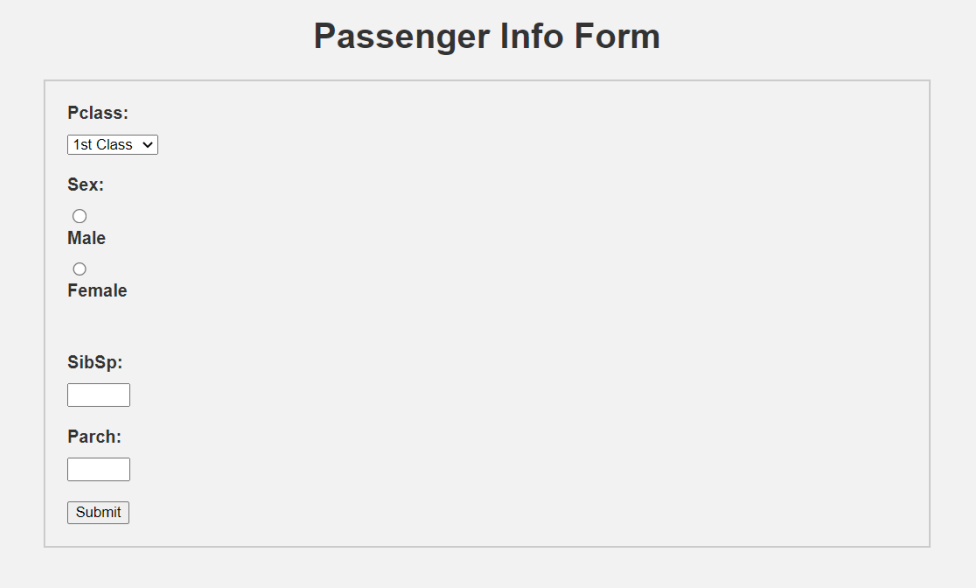


Figure 12. Passenger Info Form Web Application

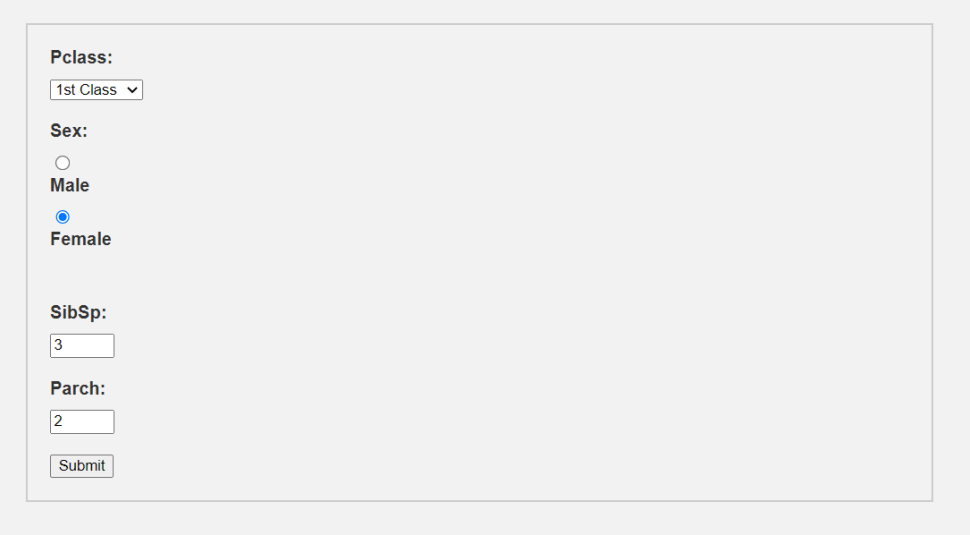


Figure 13. Input passenger info in the form

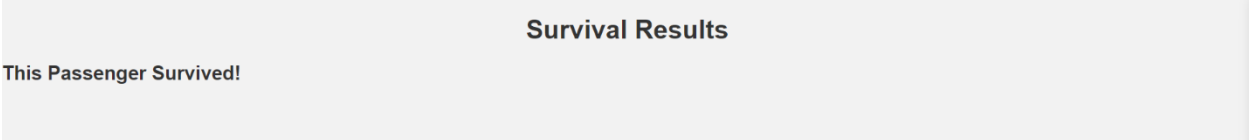


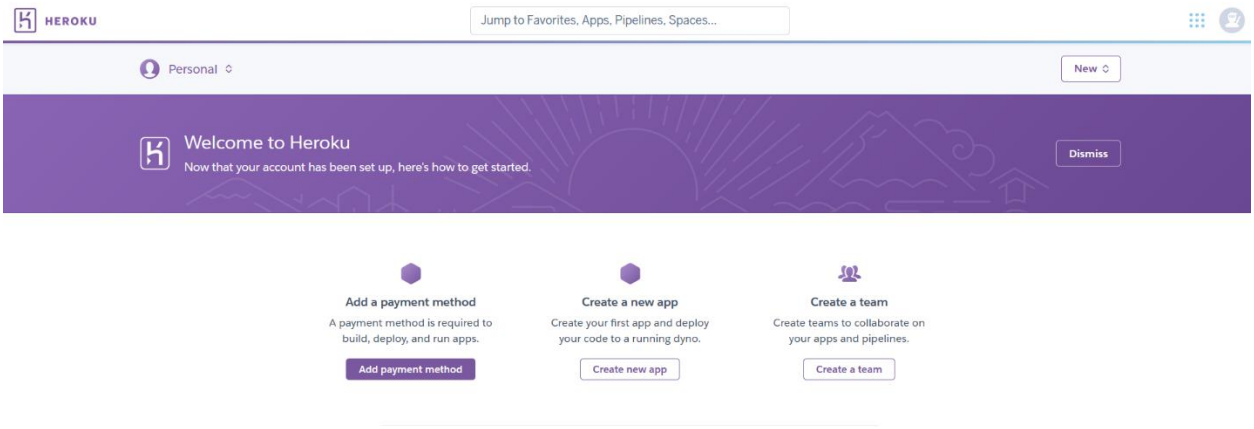
Figure 14. Result of the given input

5. Model deployment using Heroku

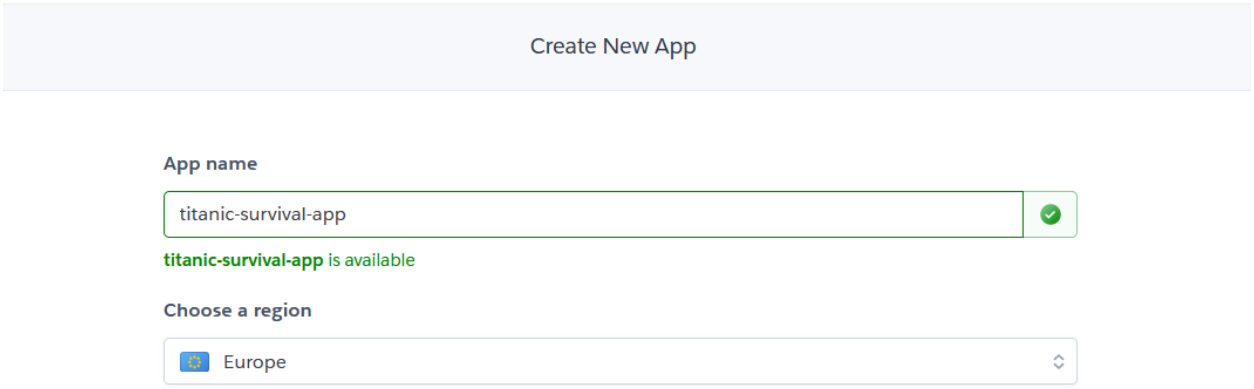
The model has been trained, the machine learning pipeline has been set up, and the application has been tested locally. The next step is to upload the application source code from the GitHub repository to an Heroku account.

The steps for Model Deployment Using Heroku include

1. After signing up on heroku.com, create new app.



2. Enter App name and region



3. Connect to GitHub repository and published the app.

The app is published at:

http://titanic-survival-app.herokuapp.com/