Week 4 - Flask Deployment

Name: **Asha K C**

Submitted date: **28 March 2025**

Internship Batch: **LISUM43**

Submitted to: **Data Glacier**

Task given:

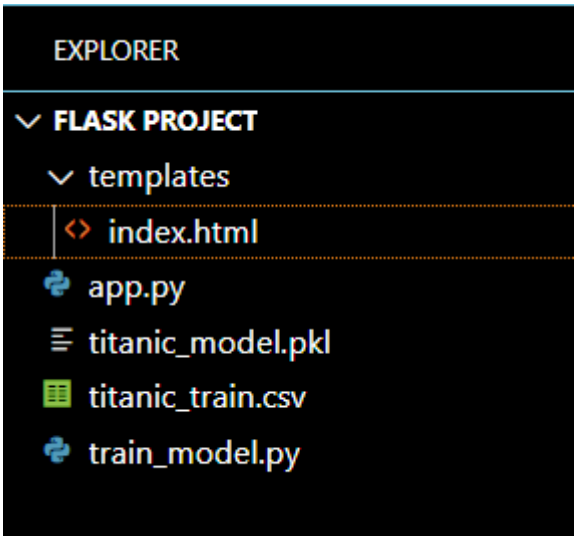
1. Choose any simple data.
2. Create a prediction model. In this assignment, I'm using **Titanic** data from Kaggle to predict whether a passenger survived the disaster or not.
3. Deploy the project in web on Flask

Steps followed for this project:

1. Created **train_model.py** python code. The data within the titanic data set is read, preprocessed and trained with a test set. Then pickling is used to dump the model into **titanic_model.pk** in this python code.
2. Created a html file for a web user interface page - **index.html**. A form is used here for users to input these info - Passenger Class, Age, Number of Siblings aboard, number of parents/children aboard, Fare, Sex, Port of Embarkation.
3. Created **app.py** python code to read user's input and make the prediction using model generated.

   "*Survived*" if prediction == 1 else "*Did not survive*"

Below are screens showing the programs used and their result:



train_model.py > ...

```python
# Importing the libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import pickle

def preprocess_data(df):
    """Fill missing values and convert categorical variables."""
    df['Age'].fillna(df['Age'].median(), inplace=True)
    df['Fare'].fillna(df['Fare'].median(), inplace=True)
    # For the 'Embarked' column fill with mode
    df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)

    # Convert categorical variables using one-hot encoding.
    # We use drop_first=True to avoid multicollinearity.
    df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)
    return df

if __name__ == "__main__":
    # Load your Titanic dataset
    data = pd.read_csv('titanic_train.csv')

    # Select features and target.
    # We ignore columns like 'PassengerId', 'Name', 'Ticket', 'Cabin' here.
    features = ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Sex', 'Embarked']
    X = data[features]
    y = data['Survived']

    # Preprocess features
    X = preprocess_data(X)

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

    # Train the logistic regression model
    model = LogisticRegression(max_iter=1000)
    model.fit(X_train, y_train)

    # Evaluate model accuracy on the test set
    preds = model.predict(X_test)
    acc = accuracy_score(y_test, preds)
    print("Test Accuracy: {:.2f}%".format(acc * 100))

    # Save the model to disk
    pickle.dump(model, open('titanic_model.pkl','wb'))

    print("Model saved as titanic_model.pkl")
```

```
Test Accuracy: 81.01%
Model saved as titanic_model.pkl
```

templates > index.html > html

```html
<!DOCTYPE html>
<html>
<head>
    <title>Titanic Survival Prediction</title>
</head>
<body>
    <h1>Titanic Survival Prediction</h1>
    <form action="/predict" method="post">
        <label for="Pclass">Passenger Class (1, 2, or 3):</label>
        <input type="number" name="Pclass" id="Pclass" min="1" max="3" required>
        <br><br>

        <label for="Age">Age:</label>
        <input type="number" step="any" name="Age" id="Age" required>
        <br><br>

        <label for="SibSp">Number of Siblings Aboard:</label>
        <input type="number" name="SibSp" id="SibSp" required>
        <br><br>

        <label for="Parch">Number of Parents/Children Aboard:</label>
        <input type="number" name="Parch" id="Parch" required>
        <br><br>

        <label for="Fare">Fare:</label>
        <input type="number" step="any" name="Fare" id="Fare" required>
        <br><br>

        <label for="Sex">Sex:</label>
        <select name="Sex" id="Sex">
            <option value="female">Female</option>
            <option value="male">Male</option>
        </select>
        <br><br>

        <label for="Embarked">Port of Embarkation:</label>
        <select name="Embarked" id="Embarked">
            <option value="C">Cherbourg (C)</option>
            <option value="Q">Queenstown (Q)</option>
            <option value="S">Southampton (S)</option>
        </select>
        <br><br>

        <input type="submit" value="Predict">
    </form>
    <br>
    <!-- Display prediction result -->
    {% if prediction_text %}
        <h2>{{ prediction_text }}</h2>
    {% endif %}
</body>
</html>
```

```
 * 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: 853-121-057
```

app.py > predict

```python
import pandas as pd
from flask import Flask, request, render_template
import pickle

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

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

@app.route('/predict',methods=['POST'])
def predict():
    try:
        # Get form data
        pclass = int(request.form['Pclass'])
        age = float(request.form['Age'])
        sibsp = int(request.form['SibSp'])
        parch = int(request.form['Parch'])
        fare = float(request.form['Fare'])
        sex = request.form['Sex']  # Expected to be 'male' or 'female'
        embarked = request.form['Embarked']  # Expected values: 'C', 'Q', or 'S'

        # Create a DataFrame for the input features
        input_df = pd.DataFrame({
            'Pclass': [pclass],
            'Age': [age],
            'SibSp': [sibsp],
            'Parch': [parch],
            'Fare': [fare],
            'Sex': [sex],
            'Embarked': [embarked]
        })

        # Preprocess input similar to training
        # (Fill missing values if necessary; here we assume values are provided)
        input_df = pd.get_dummies(input_df, columns=['Sex', 'Embarked'], drop_first=True)

        # We expect the same features as used during training:
        # ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Sex_male', 'Embarked_Q', 'Embarked_S']
        expected_cols = ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Sex_male', 'Embarked_Q', 'Embarked_S']
        for col in expected_cols:
            if col not in input_df.columns:
                input_df[col] = 0  # Add missing binary columns with default value zero
        # Ensure columns are in the same order as expected
        input_df = input_df[expected_cols]
```

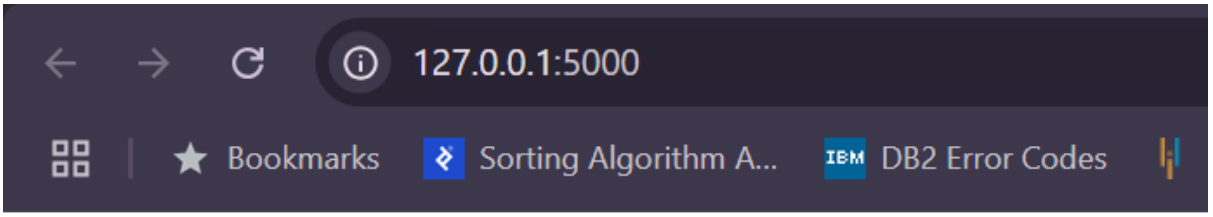
```python

        # Make prediction
        prediction = model.predict(input_df)[0]
        prediction_text = "Survived" if prediction == 1 else "Did not survive"

        # Render result on the same page (you can customize this as needed)
        return render_template('index.html', prediction_text=f"Prediction: {prediction_text}")
    except Exception as e:
        return render_template('index.html', prediction_text=f"Error: {str(e)}")

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

Web screens:



Titanic Survival Prediction

Passenger Class (1, 2, or 3):

Age:

Number of Siblings Aboard:

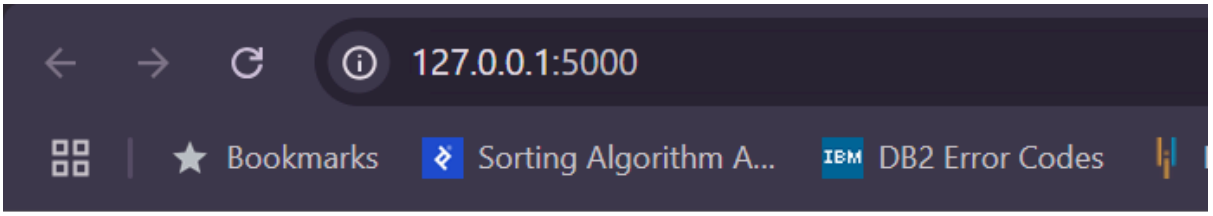
Number of Parents/Children Aboard:

Fare:

Sex: Female

Port of Embarkation: Cherbourg (C)

Predict



Titanic Survival Prediction

Passenger Class (1, 2, or 3): 1

Age: 45

Number of Siblings Aboard: 0

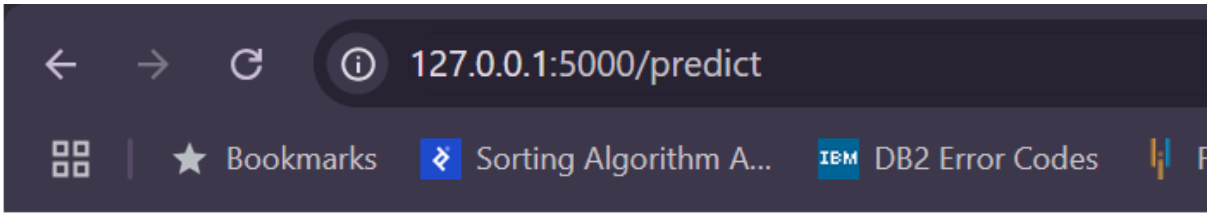
Number of Parents/Children Aboard: 0

Fare: 5489

Sex: Male

Port of Embarkation: Queenstown (Q)

Predict



Titanic Survival Prediction

Passenger Class (1, 2, or 3):

Age:

Number of Siblings Aboard:

Number of Parents/Children Aboard:

Fare:

Sex: Female

Port of Embarkation: Cherbourg (C)

Predict

Prediction: Survived