VIETNAM NATIONAL UNIVERSITY,

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

Assignment

Flight Delay Prediction

Course: Machine Learning

Instructor: Nguyen Duc Dung

Semester 241 Class CC01

Name	ID
Võ Văn Toàn	2052750

Table of contents

I/ Introduction:

Flight Delay Prediction is a popular project on Kaggle. In this article, I outline the steps taken to analyze the Flight Delay Prediction dataset. The process included data preprocessing steps such as cleaning the data, replacing missing values, and applying normalization where necessary. The dataset was then split into training and testing sets, and a Decision Tree model was built, achieving an accuracy of approximately 99.9%. Following model development, a static HTML page was created to gather user input. The trained model was used to predict whether a flight would be delayed based on the provided input features. Flask was employed to integrate the static page with the model, enabling the display of prediction results to the user.

II/ Main Report:

1/ Importing the Libraries

We begin the analysis by importing the necessary libraries for building the model.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

2. Datasets

We have three datasets available to play around with for modeling and analysis purposes. The first dataset is the flight details consisting of features about a flight scheduled, flight arrival details, flight delay factors, flight source, and destination details.

```
flights=pd.read_csv('flights.csv',low_memory=True)
```

```
flights
```

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT	DESTINATION_A
0	2015	1	1	4	AS	98	N407AS	ANC	
1	2015	1	1	4	AA	2336	N3KUAA	LAX	
2	2015	1	1	4	US	840	N171US	SFO	
3	2015	1	1	4	AA	258	N3HYAA	LAX	
4	2015	1	1	4	AS	135	N527AS	SEA	
...	...	...	...	...	...	...	...	...	
1048570	2015	3	10	2	EV	4122	N11191	RDU	
1048571	2015	3	10	2	UA	1018	N79279	LGA	
1048572	2015	3	10	2	UA	1260	N76508	SAN	
1048573	2015	3	10	2	EV	4349	N14158	MSY	
1048574	2015	3	10	2	MQ	2916	N539MQ	CID	

The second dataset consists of airport details such as airport location, state, and city.

```
airports=pd.read_csv('airports.csv')
airports
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.44040
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.68190
2	ABQ	Albuquerque International Sunport	Albuquerque	NM	USA	35.04022	-106.60919
3	ABR	Aberdeen Regional Airport	Aberdeen	SD	USA	45.44906	-98.42183
4	ABY	Southwest Georgia Regional Airport	Albany	GA	USA	31.53552	-84.19447
...	...	...	...	...	...	...	...
317	WRG	Wrangell Airport	Wrangell	AK	USA	56.48433	-132.36982
318	WYS	Westerly State Airport	West Yellowstone	MT	USA	44.68840	-111.11764
319	XNA	Northwest Arkansas Regional Airport	Fayetteville/Springdale/Rogers	AR	USA	36.28187	-94.30681
320	YAK	Yakutat Airport	Yakutat	AK	USA	59.50336	-139.66023

The third dataset consists of details regarding the airlines.

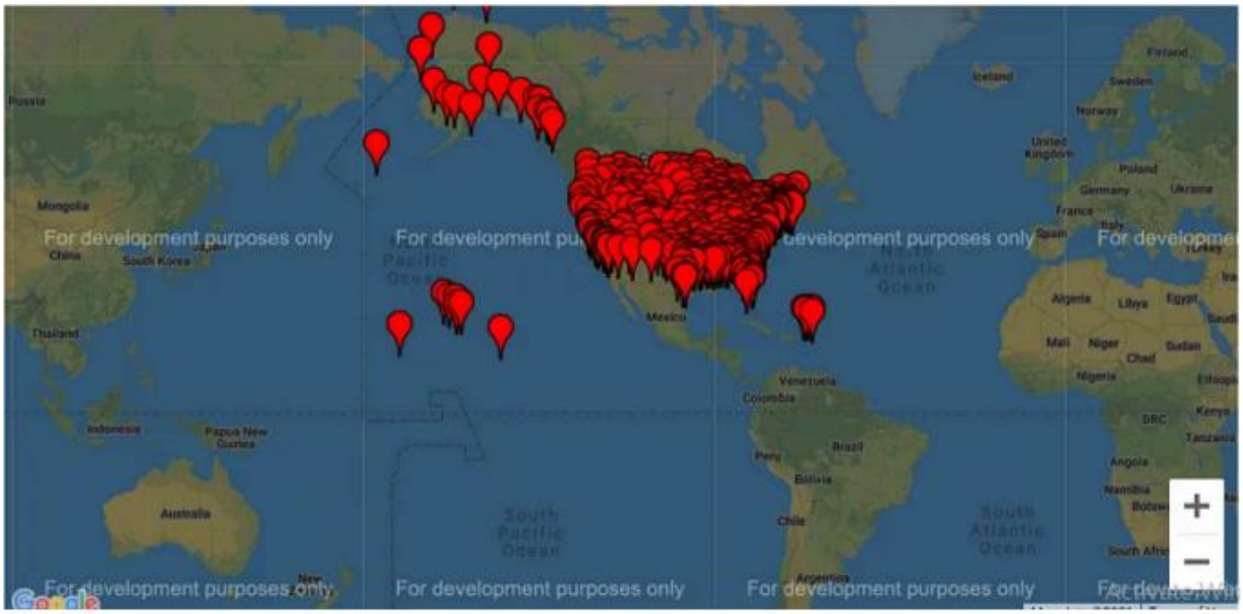
```
airlines=pd.read_csv('airlines.csv')
airlines
```

	IATA_CODE	AIRLINE
0	UA	United Air Lines Inc.
1	AA	American Airlines Inc.
2	US	US Airways Inc.
3	F9	Frontier Airlines Inc.
4	B6	JetBlue Airways
5	OO	Skywest Airlines Inc.
6	AS	Alaska Airlines Inc.
7	NK	Spirit Air Lines
8	WN	Southwest Airlines Co.
9	DL	Delta Air Lines Inc.
10	EV	Atlantic Southeast Airlines
11	HA	Hawaiian Airlines Inc.

Next, I decided to find the location of various airports on google maps. So I used the gmplot to locate the airports based on the Longitude and Latitude values. In the below image I haven't used Google API and so it hasn't loaded correctly. One can google and find ways to use API and integrate with gmplot.

```
import gmplot
latitudes=airports.loc[:,'LATITUDE']
longitudes=airports.loc[:,'LONGITUDE']
gmap=gmplot.GoogleMapPlotter(35,102,2)
gmap.scatter(latitudes,longitudes,'red',size=5)
gmap.draw('map/gmplot.html')
```

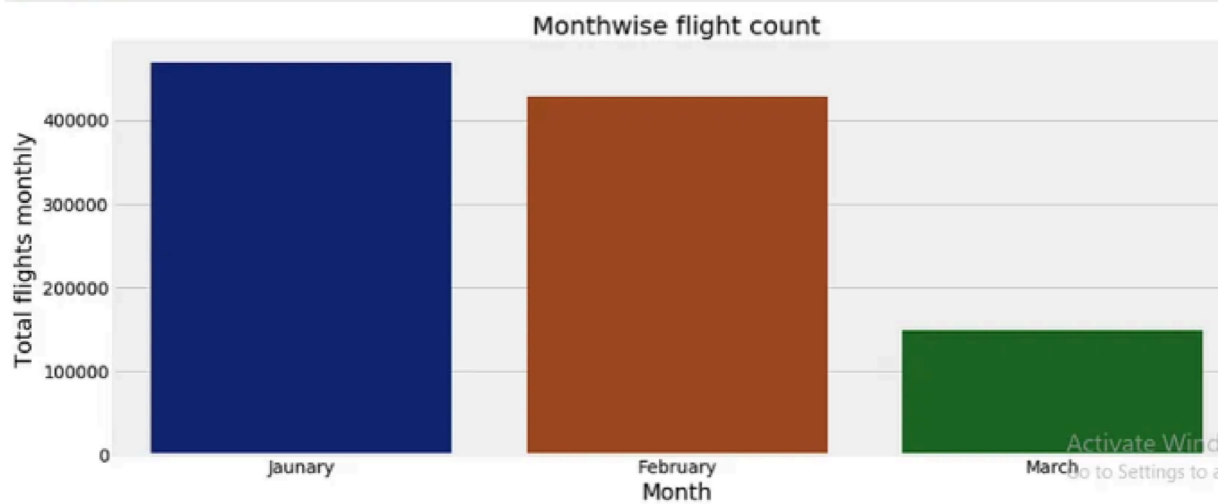
```
from IPython.display import IFrame
IFrame(src='map/gmplot.html',width=900, height=600)
```



Next, we take the month-wise flight details and analyze how many flights take off on monthly basis. Hence we map the month number with a month name. After that, we use the plot function to visualize the flight details month-wise.

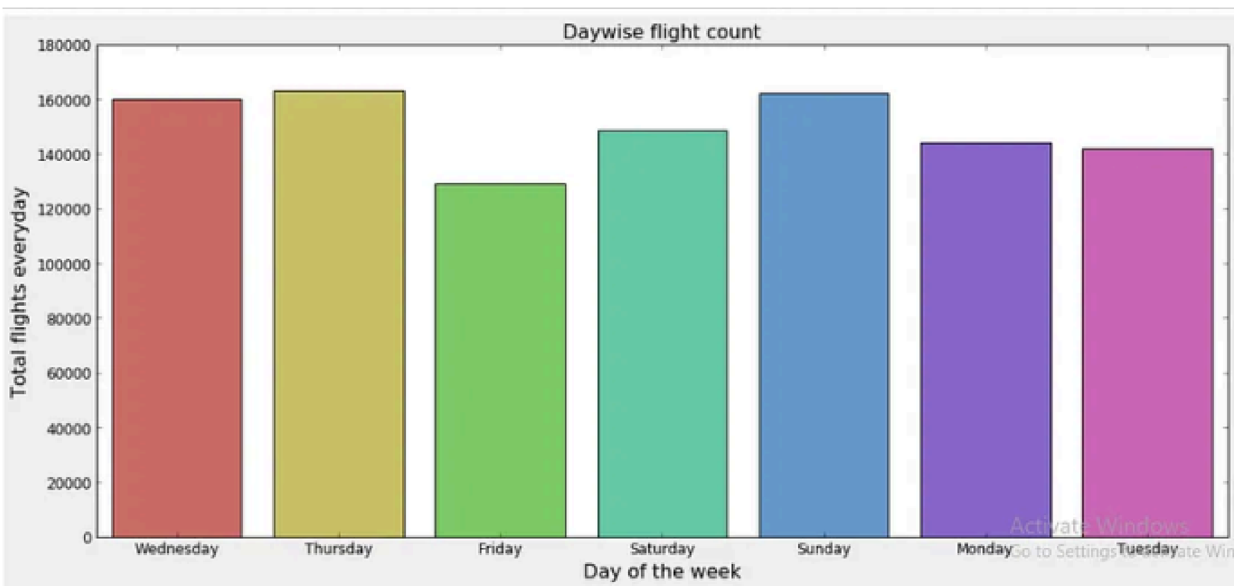
```
flights1=flights[['MONTH','DAY_OF_WEEK']]
flights1['MONTH']=flights1['MONTH'].map({1:'Jaunary',2:'February',3:'March',4:'April',5:'May',6:'June',7:'July',8:'August',
                                        9:'September',10:'October',11:'November',12:'December'})
flights1['DAY_OF_WEEK']=flights1['DAY_OF_WEEK'].map({1:'Sunday',2:'Monday',3:'Tuesday',4:'Wednesday',5:'Thursday',6:'Friday',
                                                    7:'Saturday'})
```

```
plt.figure(figsize=(16,6))
plt.style.use('fivethirtyeight')
ax=sns.countplot('MONTH',data=flights1,palette='dark',)
ax.set_xlabel(xlabel='Month',fontsize=18)
ax.set_ylabel(ylabel='Total flights monthly',fontsize=18)
ax.set_title(label='Monthwise flight count',fontsize=20)
plt.show()
```



Next, we take the day-wise flight details and analyze how many flights take off on daily basis. Hence we map the weekday number with a weekday name. After that, we use the plot function to visualize the flight details daily basis.

```
plt.figure(figsize = (15, 7))
plt.style.use('_classic_test')
sns.countplot(x ='DAY_OF_WEEK',data=flights1,palette='hls')
plt.title('Daywise flight count',fontsize=16)
plt.xlabel('Day of the week', fontsize = 16)
plt.ylabel('Total flights everyday', fontsize = 16)
plt.show()
```



3/ Cleaning the Data

As the first step of cleaning the data, we can begin with dropping the negligible features in the dataset.

```
flights=flights.drop(['YEAR','FLIGHT_NUMBER','AIRLINE','DISTANCE','TAIL_NUMBER','TAXI_OUT',
                                           'SCHEDULED_TIME','DEPARTURE_TIME','WHEELS_OFF','ELAPSED_TIME',
                                           'AIR_TIME','WHEELS_ON','DAY_OF_WEEK','TAXI_IN','CANCELLATION_REASON'],
                                          axis=1)
```

```
flights.head()
```

	MONTH	DAY	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE	DEPARTURE_DELAY	SCHEDULED_ARRIVAL	ARRIVAL_TIME	ARRIVAL_
0	1	1	ANC	SEA	5	-11.0	430	408.0	
1	1	1	LAX	PBI	10	-8.0	750	741.0	
2	1	1	SFO	CLT	20	-2.0	806	811.0	
3	1	1	LAX	MIA	20	-5.0	805	756.0	
4	1	1	SEA	ANC	25	-1.0	320	259.0	

As the second step, we will be replacing the null values with a mean value of the respective features.

```
flights=flights.fillna(flights.mean())
```

```
flights.head()
```

	MONTH	DAY	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE	DEPARTURE_DELAY	SCHEDULED_ARRIVAL	ARRIVAL_TIME	ARRIVAL_
0	1	1	ANC	SEA	5	-11.0	430	408.0	
1	1	1	LAX	PBI	10	-8.0	750	741.0	
2	1	1	SFO	CLT	20	-2.0	806	811.0	
3	1	1	LAX	MIA	20	-5.0	805	756.0	
4	1	1	SEA	ANC	25	-1.0	320	259.0	

As a third step, we will create a dependent feature named result which will classify the flights delayed or not based on the arrival delay being less or more than 15 mins.

```python
result=[]
```

```python
for row in flights['ARRIVAL_DELAY']:
    if row > 15:
        result.append(1)
    else:
        result.append(0)
```

```python
flights['result'] = result
```

```python
flights
```

_TIME	ARRIVAL_DELAY	DIVERTED	CANCELLED	AIR_SYSTEM_DELAY	SECURITY_DELAY	AIRLINE_DELAY	LATE_AIRCRAFT_DELAY	WEATHER_DELAY	result
000000	-22.000000	0	0	13.692554	0.057328	18.203577	22.921458	3.545277	0
000000	-9.000000	0	0	13.692554	0.057328	18.203577	22.921458	3.545277	0
000000	5.000000	0	0	13.692554	0.057328	18.203577	22.921458	3.545277	0
000000	-9.000000	0	0	13.692554	0.057328	18.203577	22.921458	3.545277	0
000000	-21.000000	0	0	13.692554	0.057328	18.203577	22.921458	3.545277	0
...	...	...	...	...	...	...	...	...	...
000000	-16.000000	0	0	13.692554	0.057328	18.203577	22.921458	3.545277	0
000000	-2.000000	0	0	13.692554	0.057328	18.203577	22.921458	3.545277	0

As the fourth step, we will remove a few more non-contributing features from the dataframe.

```python
flights=flights.drop(['ORIGIN_AIRPORT', 'DESTINATION_AIRPORT', 'ARRIVAL_TIME', 'ARRIVAL_DELAY'],axis=1)
flights
```

	MONTH	DAY	SCHEDULED_DEPARTURE	DEPARTURE_DELAY	SCHEDULED_ARRIVAL	DIVERTED	CANCELLED	AIR_SYSTEM_DELAY	SECURITY_DEL
0	1	1	5	-11.000000	430	0	0	13.692554	0.057
1	1	1	10	-8.000000	750	0	0	13.692554	0.057
2	1	1	20	-2.000000	806	0	0	13.692554	0.057
3	1	1	20	-5.000000	805	0	0	13.692554	0.057
4	1	1	25	-1.000000	320	0	0	13.692554	0.057
...	...	...	...	...	...	...	...	...	...
1048570	3	10	1013	-8.000000	1149	0	0	13.692554	0.057
1048571	3	10	1013	-8.000000	1337	0	0	13.692554	0.057
1048572	3	10	1013	-3.000000	1624	0	0	13.692554	0.057
1048573	3	10	1013	-10.000000	1242	0	0	13.692554	0.057
1048574	3	10	1013	11.334851	1115	0	1	13.692554	0.057

As the fifth step, we will normalize the input features by dropping the output column.

```python
sc=StandardScaler()
X=flights.drop(columns='result')
Y=flights['result']
X=sc.fit_transform(X)
```

4/ Model Creation

We split the data into training and test set and then train the data using the Decision Tree Classifier.

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=42)
```

```
from sklearn.tree import DecisionTreeClassifier
clf=DecisionTreeClassifier()
clf.fit(X_train,Y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
clf.score(X_test,Y_test)
```

```
0.9995279307631786
```

5/ Hyperparameter Tunning

We use the GridSearchCV function to tune the parameters and try to find the best combination that gives us maximum accuracy in the model. So we first fit the training data on the various parameter values and try to find the best combination of parameters. Once the combination of best params is obtained we then train the Decision Tree model based on it. Once the model is ready we apply it to the test dataset.

```
grid_param = {
'criterion': ['gini', 'entropy'],
'max_depth' : range(30,31,1),
'min_samples_leaf' : range(35,38,1),
'min_samples_split': range(35,38,1),
'splitter' : ['best', 'random']
}
```

```
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(estimator=clf,
    param_grid=grid_param,
    cv=5,
    n_jobs =-1)
```

```
grid_search.fit(X_train,Y_train)
```

```
best_parameters = grid_search.best_params_
print(best_parameters)
```

```
{'criterion': 'entropy', 'max_depth': 30, 'min_samples_leaf': 35, 'min_samples_split': 35, 'splitte
r': 'best'}
```

```
grid_search.best_score_
```

```
0.9990773192189399
```

```
clf = DecisionTreeClassifier(criterion = 'entropy', max_depth =30, min_samples_leaf= 35, min_samples_sp
clf.fit(X_train,Y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=30, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=35, min_samples_split=35,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
clf.score(X_test,Y_test)
```

```
0.9992513649476671
```

6/ Saving the model

Once the model is designed we then save it in a file that appears in an unreadable format. Later when we want to apply the model on an unseen dataset then we can directly call the model saved in the file and generate the output as 0 or 1 which indicates whether the flight will be delayed based on the input features.

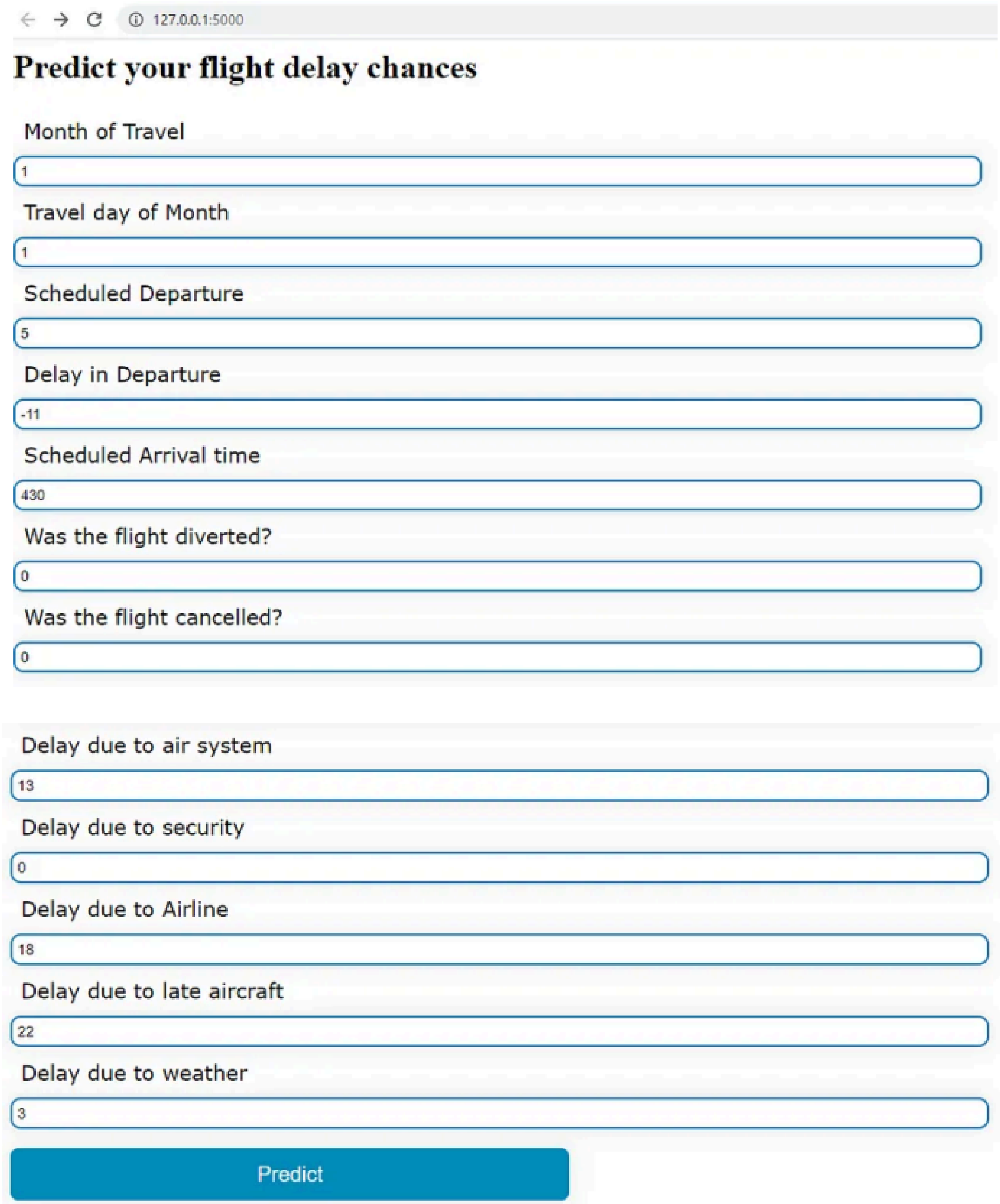
```
filename = 'finalized_model.sav'
pickle.dump(clf, open(filename, 'wb'))
```

```
loaded_model = pickle.load(open(filename, 'rb'))
print('HI there')
prediction=loaded_model.predict([[1,1,5,-11,430,0,0,13,0,
                                  18,22,3]])
print(prediction)
```

```
HI there
[0]
```

7/ Creating the Static HTML Templates

We create a templates folder that consists of two HTML files. One of them is the page that displayed the fields which are submitted as inputs from the user. The other page is the Results.html page which consists of the result which is obtained from the model output. The Registration HTML page consists of information such as a month, flight scheduled arrival and departure timing, flight delay reasons, whether the flight got canceled or not, etc.



The results HTML page looks as shown below:


127.0.0.1:5000/predict
Flight Delay Prediction
Your flight wont get delayed

8/ Integrating the model and Templates using Flask

We use the flask to integrate the HTML templates with the model in the backend. The values which are taken as input from the user are redirected using the POST method in Flask to the model. The model then will generate the output and pass it as the input value to the Results HTML page. The GET method will call the results template and push the prediction obtained from the model output.

```
@app.route('/predict',methods=['GET','POST'])
def index():
 if request.method=='POST':
  try:
   month = int(request.form['month'])
   day   = int(request.form['day'])
   schdl_dep = float(request.form['schdl_dep'])
   dep_delay = float(request.form['dep_delay'])
   schdl_arriv = float(request.form['schdl_arriv'])
   divrtd = int(request.form['divrtd'])
   cancld = int(request.form['cancld'])
   air_sys_delay = float(request.form['air_sys_delay'])
   secrty_delay = float(request.form['secrty_delay'])
   airline_delay = float(request.form['airline_delay'])
   late_air_delay = float(request.form['late_air_delay'])
   wethr_delay  = float(request.form['wethr_delay'])
```

```
filename = 'finalized_model.sav'
loaded_model = pickle.load(open(filename, 'rb'))
```

```
import numpy as np
prediction=loaded_model.predict([[month,day,schdl_dep,dep_delay,schdl_arriv,divrtd,cancld,air_sys_delay,secrty_delay,
                                  airline_delay,late_air_delay,wethr_delay]])
for i in prediction:
 if i==1:
  prediction='will be'
 else:
  prediction='wont get'
```

```
  return render_template('results.html',prediction=prediction)
 except Exception as e:
  print('The Exception message is: ',e)
  return 'something is wrong'
else:
 return render_template('index.html'),
```

9/ Github link

This is the link to the project: