Final Project
Info 7390 - Advanced Data Science and architecture

Predicting Airline Delays

Presented by:

Dhanisha Damodar Phadate
001859234
phadate.d@husky.neu.edu

Palak Sharma
001828562
sharma.pala@husky.neu.edu

Dharani Thirumalaisamy
001887922
thirumalaisamy.d@husky.neu.edu

INDEX

PART 1 : MODEL DEVELOPMENT

Abstract :

The project's aim is to predict the airline delay based on the historic datasets. In this part of the project , the data is taken from the cloud and pre-processing is carried out to generate the pickle files which when loaded will provide the desired output. The pickle files at the end is pushed to the Amazon's cloud.

a) Data Ingestion :

```
url ='https://storage.googleapis.com/team3/final_flight_delay.csv'
df = urllib.request.urlopen(url)

df  = pd.read_csv(df)
```

The dataset that is used in this project is of 2008's and 2007's. These two datasets and concatenated and pushed to the Google Cloud Platform. The url link provided is the public link of the dataset that is on Google Cloud Platform.

The dataset is ingested into the notebook from the cloud and the pre-processing techniques are carried out.

b) Data Cleaning :

When calculated for the number of missing values , there were a plenty of it.

So , in order to deal with it , the missing values are filled with a method called 'bfill' which will fill the missing values or the 'NaN' values with the previous records.

```
#def cleaning(data):
data = df.fillna(method='bfill',axis=0).fillna('0')
#    return data
#cleaning(df)
```

c) Feature Selection :

In the dataset that we have there are string values for 'Unique Carrier' , 'Dest' and 'Origin'. These values should be encoded before they are used in the prediction model.

There are different methods to do this . For example : Onehotencoding , Labelencoder , but the problem with these two is that they can't be converted back to string format . So , if we display the output in the numeric format , it is not possible for the user to understand that. So , to avoid that :

```
uc = data.UniqueCarrier
uc = uc.replace('9E','1')
uc = uc.replace('AA','2')
uc = uc.replace('AQ','3')
uc = uc.replace('AS','4')
uc = uc.replace('B6','5')
uc = uc.replace('CO','6')
uc = uc.replace('DL','7')
uc = uc.replace('EV','8')
uc = uc.replace('F9','9')
uc = uc.replace('FE','10')
uc = uc.replace('HA','11')
uc = uc.replace('MQ','12')
uc = uc.replace('NW','13')
uc = uc.replace('OH','14')
uc = uc.replace('OO','15')
uc = uc.replace('UA','16')
uc = uc.replace('US','17')
uc = uc.replace('WN','18')
uc = uc.replace('XE','19')
uc = uc.replace('YV','20')
uc = uc.replace('FL','21')
```

```
origin = data['Origin']
origin = origin.replace('ABE','1')
origin = origin.replace('ABI','2')
origin = origin.replace('ABQ','3')
origin = origin.replace('ABY','4')
origin = origin.replace('ACT','5')
origin = origin.replace('ACV','6')
origin = origin.replace('ACY','7')
origin = origin.replace('ADK','8')
origin = origin.replace('ADQ','9')
origin = origin.replace('AEX','10')
origin = origin.replace('AGS','11')
origin = origin.replace('AKN','12')
origin = origin.replace('ALB','13')
origin = origin.replace('ALO','14')
origin = origin.replace('AMA','15')
origin = origin.replace('ANC','16')
origin = origin.replace('APF','17')
origin = origin.replace('ASE','18')
origin = origin.replace('ATL','19')
```

```
dest = data['Dest']
dest = dest.replace('ABE','1')
dest = dest.replace('ABI','2')
dest = dest.replace('ABQ','3')
dest = dest.replace('ABY','4')
dest = dest.replace('ACT','5')
dest = dest.replace('ACV','6')
dest = dest.replace('ACY','7')
dest = dest.replace('ADK','8')
dest = dest.replace('ADQ','9')
dest = dest.replace('AEX','10')
dest = dest.replace('AGS','11')
dest = dest.replace('AKN','12')
dest = dest.replace('ALB','13')
dest = dest.replace('ALO','14')
dest = dest.replace('AMA','15')
dest = dest.replace('ANC','16')
dest = dest.replace('APF','17')
dest = dest.replace('ASE','18')
dest = dest.replace('ATL','19')
dest = dest.replace('ATW','20')
dest = dest.replace('AUS','21')
dest = dest.replace('AVL','22')
dest = dest.replace('AVP','23')
```

```
#def featureselection():

data = data[['Month','DayofMonth','DayOfWeek','uniquecarrier_int','Cancelled','Diverted','CarrierDelay','WeatherDelay',
        'NASDelay','SecurityDelay','LateAircraftDelay','FlightNum','ArrDelay','DepDelay','origin_int','dest_int']]

df_inputs = data[['Month','DayofMonth','DayOfWeek','uniquecarrier_int','origin_int','dest_int']]

df_target_reg = data[['ArrDelay','DepDelay']]

df_target_classi_1 = data[['Cancelled']]
df_target_classi_2 = data[['CarrierDelay','WeatherDelay','NASDelay','SecurityDelay','LateAircraftDelay']]
```

In this problem , it is not necessary to use any automated feature selection tool to select the features that affect the output because we know the answer our self.

Therefore in the final dataset that we are using for the model will be 'data' which has the following labels :

'Month','DayofMonth','DayOfWeek','uniquecarrier_int','Cancelled','Diverted','CarrierDelay','WeatherDelay','NASDelay','SecurityDelay','LateAircraftDelay','FlightNum','ArrDelay','DepDelay','origin_int','dest_int'

The input that we will be taking from the user are :

1. Month - which month he is travelling (1-12)

2. DayofMonth - which date he is travelling (1-31)

3.Dayodweek - which day it is in that particular week (1-7)

4. uniquecarrier_int - which flight he is taking

5. origin_int - origin airport

6. dest_int - destination airport

The first output that the user will be given is : If there is a delay or not?
If 1 - not delayed ; If 0 - delayed.

The second output which the user will be given is : By how much is the arrival delayed and departure delayed?

The third output for the user will be : What type of delay it is ?
There are 5 types of delay : Carrier delay , weather delay , NAS delay , security delay , Late aircraft delay

d)Models Used :

i) For the first output given to the user , the model used is **Naive Bayes classifier** which gives a accuracy of 0.97. Irrespective of the type of algorithm we use , the accuracy is very high. So any algorithm can be used for this prediction.

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()

# Train our classifier
gnb.fit(df_inputs_train,  df_target_train)

pickle.dump(gnb,open('delay.pkl', "wb" ))
#from sklearn.ensemble import RandomForestClassifier
```

```
g = gnb.predict(df_inputs_test)
from sklearn.metrics import accuracy_score
r = accuracy_score(df_target_test,g)
r
```

```
0.97853725205181741
```

accuracy_score simply implies

```
c = confusion_matrix(df_target_test,g)
c
```

```
array([[2365605,       0],
       [  51886,       0]], dtype=int64)
```

since it is a single label output , the confusion matrix doesn't make much sense in this case.

ii) For the second output , the model used is **Random Forest Regressor.**

```
rfr = RandomForestRegressor(n_estimators =10, random_state = 1,n_jobs=-1)
rfr.fit(inputs_train,target_train)
#pickle.dump(rfr,open('delay_value.pkl', "wb" ))
```

```
r = rfr.predict(inputs_test)
rl = r2_score(target_test , r)
rl
```

```
0.0021386247940772818
```

For logistic regression :

```
lr = LinearRegression()
lr.fit(inputs_train,target_train)
```

```
LinearRegression(copy_X=True, fit_i
```

```
l = lr.predict(inputs_test)
l = r2_score(target_test , l)
l
```

```
-0.066348491988103442
```

R2_score value simply implies how well the dataset fits the regression line. Any variability in the dataset will be seen in the output. The greater the r2_score , better the model.

iii) For the third output , the model used is **Random Forest Regressor.**
The reason behind the usage of this algorithm is , this is a multi-label regression problem , so other algorithms can't be used since they don't support multi-class regression problem.

```
x_train , x_test , y_train , y_test = train_test_split(df_inputs, df_target_reg_2, test_size=0.30,random_state=1)
rfr_2 = RandomForestRegressor(n_estimators =5, random_state = 1,n_jobs=-1)
rfr_2.fit(x_train,y_train)
pickle.dump(rfr_2,open('delay_type_value.pkl', "wb" ))
```

The pickle files of each model is created and sent to cloud.

e) Cloud upload :

```
buck_name="team3"#enter bucket name
Input_location = 'us-west-2'
S3_client = boto3.client('s3', Input_location, aws_access_key_id= Access_key, aws

if Input_location == 'us-west-2':
        S3_client.create_bucket(
            Bucket=buck_name
        )
else:
    S3_client.create_bucket(
            Bucket=buck_name,
            CreateBucketConfiguration={'LocationConstraint': Input_location},
        )

S3_client.upload_file('delay.pkl', buck_name, 'delay.pkl')
S3_client.upload_file('delay_value.pkl', buck_name, 'delay_value.pkl')
S3_client.upload_file('delay_type.pkl', buck_name, 'delay_type.pkl')
S3_client.upload_file('delay_type.pkl', buck_name, 'delay_type_value.pkl')
```

By dockerizing the whole code , access_key ,secret_key can be given which will push the files to the Amazon cloud.

f) Dockerization:

1. TO BUILD AND RUN THE DOCKER IMAGE LOCALLY
   vim dockerfile

2. TO BUILD THE IMAGE
   docker build -f dockerfile -t finalproject

3. TO RUN THE IMAGE TO CREATE CONTAINER
   docker run -e Access_key=Access_key -e Secret_key=Secret_key -ti finalproject

4. TO TAG THE IMAGE
   docker tag <image id> dhanisha/finalproject

5. TO PUSH THE DOCKER IMAGE TO DOCKER HUB
   docker push dhanisha/finalproject

6. TO PULL THE DOCKER IMAGE FROM THE DOCKER HUB
   docker pull dhanisha/finalproject

7. TO RUN THE IMAGE PULLED FROM DOCKER HUB
   docker run -e Access_key=Access_key -e Secret_key=Secret_key -ti finalproject

PART 2 : WeB App Deployment

Abstract:

The project's aim is to predict the airline delay based on the historic datasets. In this part of the project, all the prediction values obtained from the pipeline are used as a backend code for our web app. The web App is created in python using Flask and jinja templates. Features like css, REST APIs , javascript and Bootstrap are used for the development of a fully deployed App. The App is deployed on EC2 instance which allows users to get the predicted value of whether the Flight is delay and which type of delay it is. And also the average arrival and departure time.

A) FIRING the EC2 instance:

The following things are required:

1) Aws login account.
2) Putty Key Generator (to use the pem and ppk files)
3) Putty shell.
4) The following commands to make the environment ready for the App to be deployed:
    - Changing the version of python to python3 as in ubuntu the default version is python 2.7 using the below commands:
      GO TO THE ROOT and check the version using : python –version
      Execute nano ~/ .bashrc and add the lines :
      alias python=python3
      alias python='/usr/bin/python3'
      Execute source ~/.bashrc and check the new version of python
    - Installing all the required packages : pandas, skipy, numpy, sklearn, boto3, sqlalchemy, bokeh, flask, pip3.
    - sudo apt-get install apache2 mysql-client mysql-server
    - sudo apt-get install libapache2-mod-wsgi
    - sudo a2enmod wsgi
    - cd /var/www
    - sudo mkdir FlaskApp
    - cd FlaskApp

- sudo mkdir FlaskApp
- cd FlaskApp
- sudo mkdir static
- sudo mkdir templates
- sudo nano __init__.py

5) Three main Files are created and added for the complete configuration
   a) nano /var/www/FlaskApp/FlaskApp/__init__.py
   b) sudo nano /etc/apache2/sites-available/FlaskApp.conf
   c) sudo nano /var/www/FlaskApp/flaskapp.wsgi

6) Giving the following permissions :

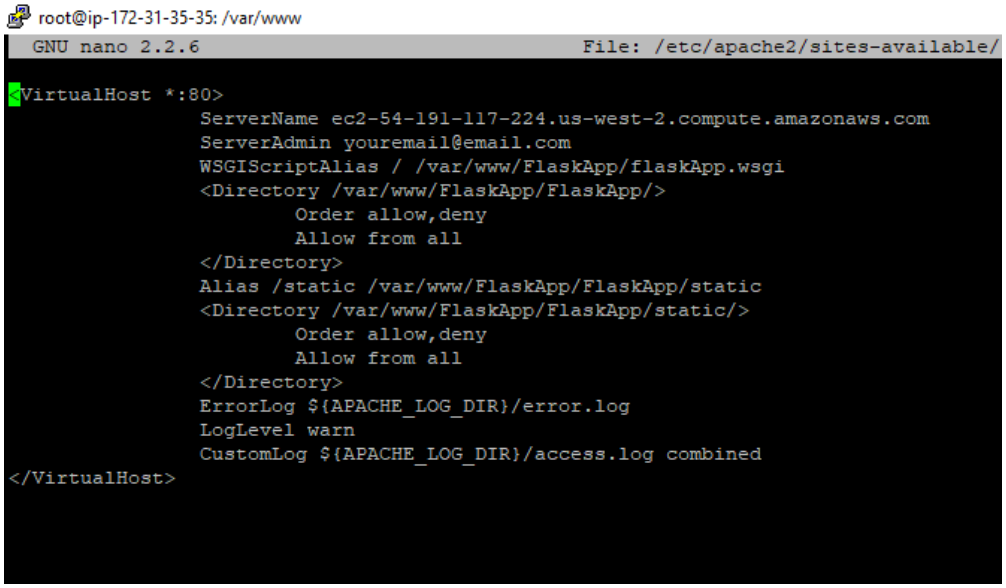
   sudo chown -R ubuntu:ubuntu /var/www
   sudo chown -R ubuntu:ubuntu /var/www/FlaskApp/FlaskApp
   sudo chown -R ubuntu:ubuntu/var/www/FlaskApp/FlaskApp/static

   sudo chown -R ubuntu:ubuntu /var/www/FlaskApp/FlaskApp/templates

   sudo chmod a+rwx /var/www/FlaskApp

   sudo chmod a+rwx /var/www/FlaskApp/FlaskApp
   sudo chmod a+rwx /var/www/FlaskApp/FlaskApp/static

7) CONFig File:



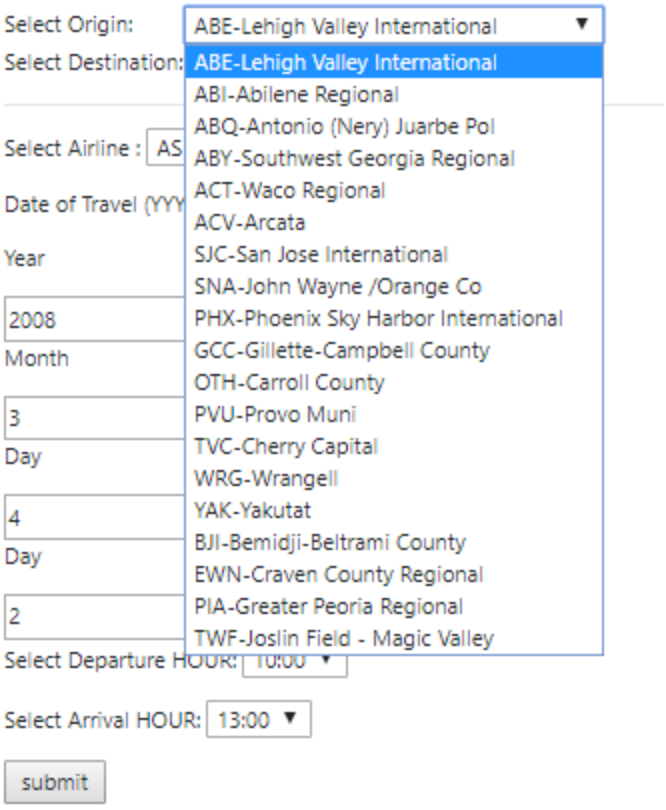
8) flaskapp.wsgi file

```
root@ip-172-31-35-35: /var/www/FlaskApp
  GNU nano 2.2.6                                        File: flaskApp.wsgi

#!/usr/bin/python3
import sys
import logging
logging.basicConfig(stream=sys.stderr)
sys.path.insert(0,"/var/www/FlaskApp/")

from FlaskApp import app as application
application.secret_key = 'gnbvnbfggcbvcmhgfdhchgf'
```

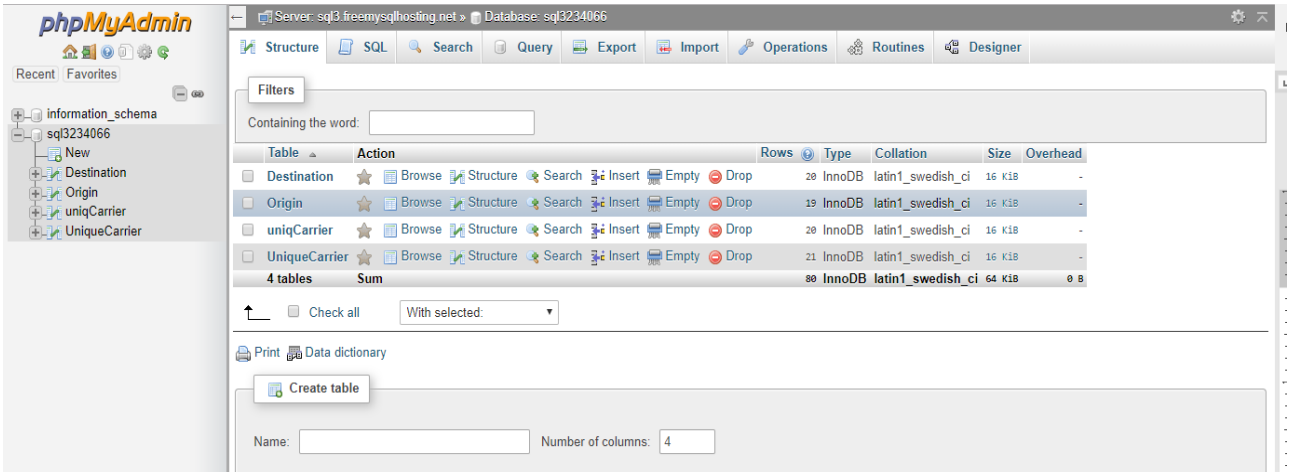
B) Deploying a Flask App: An App is up and running on : ec2-54-191-117-224.us-west-2.compute.amazonaws.com

C) Using REST APIs:

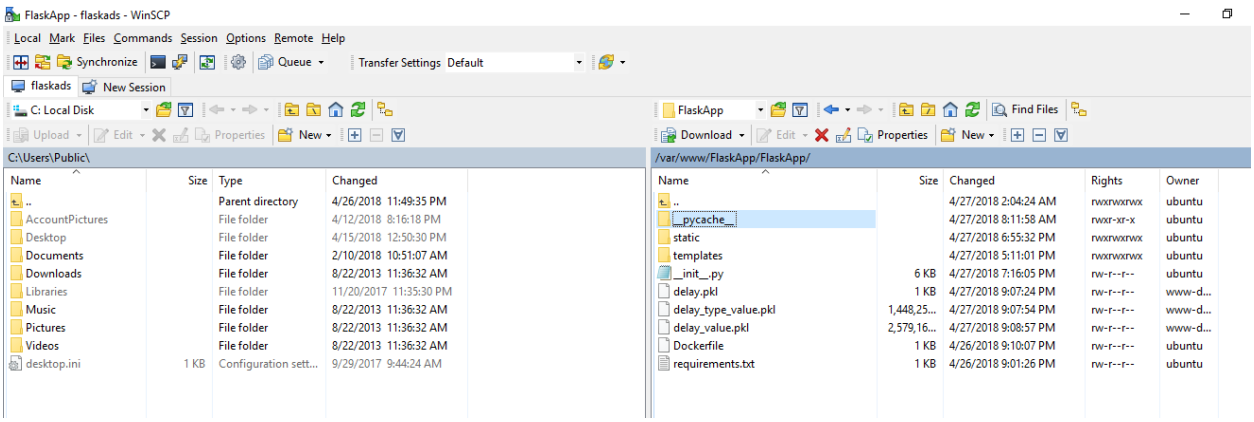
We have made the dynamic web pages which will auto fill the dropdown of the form using data from third party hosted SQL data base. Rest API calls will be sent to the server and a page will be rendered dynamically.



D) Creating an SQL database:



E) File Folder using WinSCP



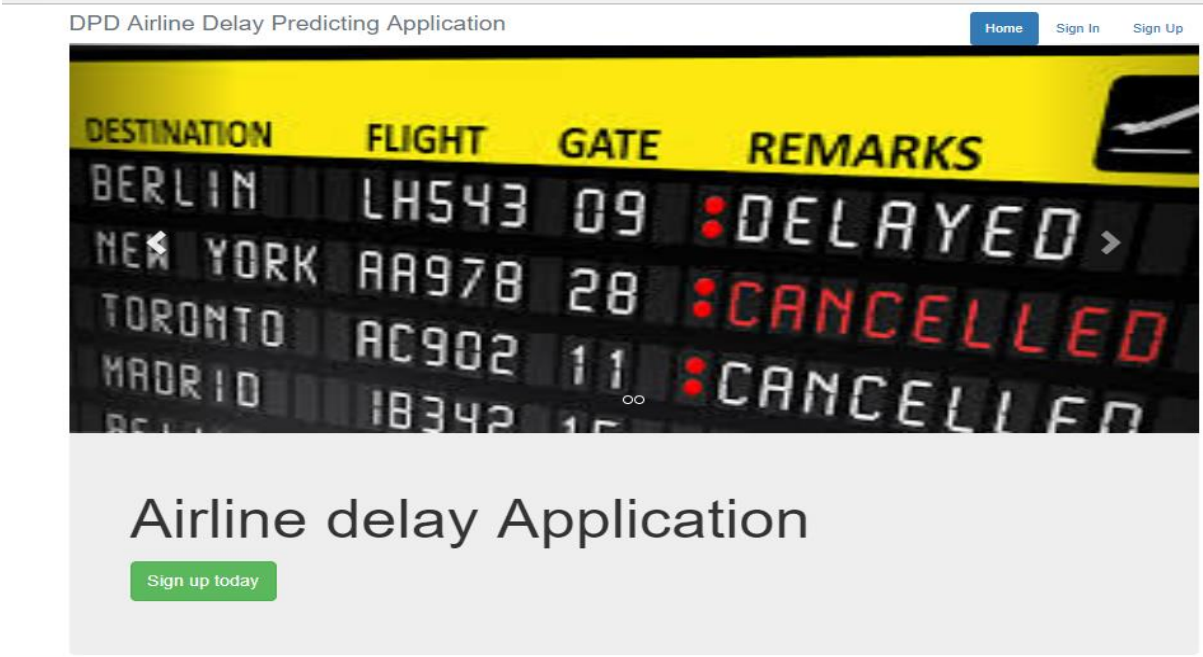
File Structure :

/var/www/FlaskApp/FlaskApp/__init__.py
/var/www/FlaskApp/flaskapp.wsgi
/var/www/FlaskApp/FlaskApp/static
/var/www/FlaskApp/FlaskApp/templates

F) Fully Functional App Deployed

Domain name: http://ec2-54-191-117-224.us-west-2.compute.amazonaws.com

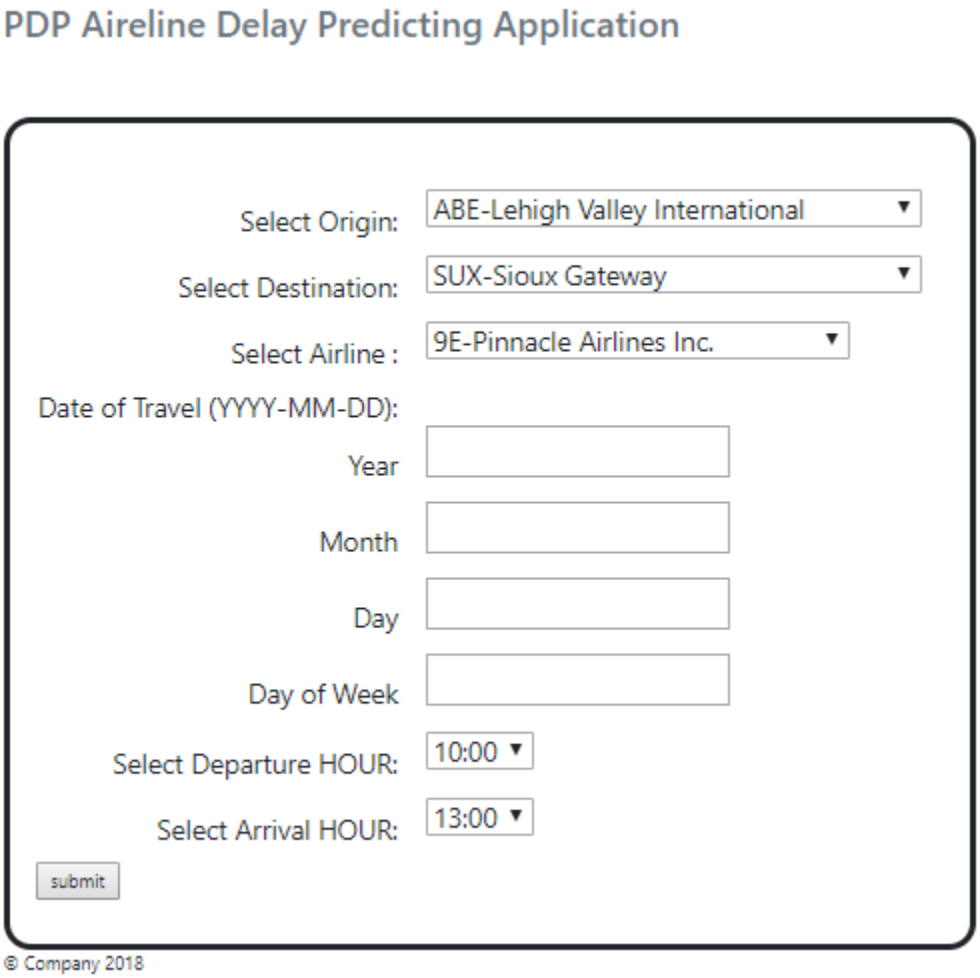
Home page:



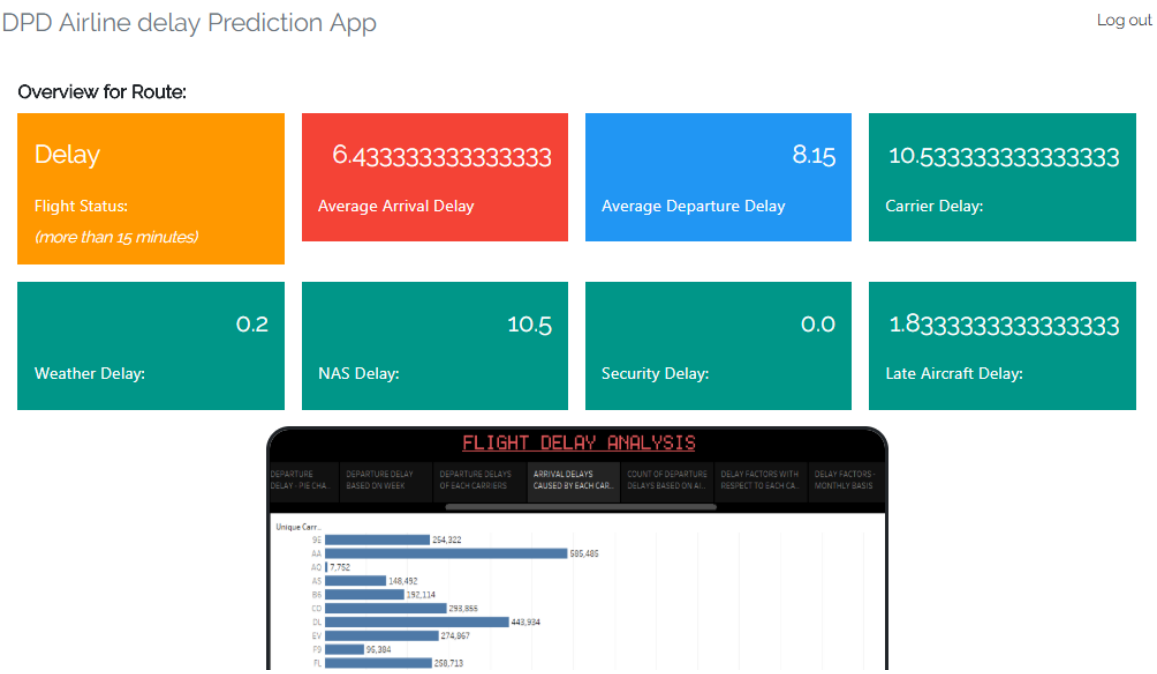
Login page:

Register page:

Form page:



Predict page:



G) Dockerization:

1. TO BUILD AND RUN THE DOCKER IMAGE LOCALLY
   vim dockerfile

2. TO BUILD THE IMAGE
   docker build -f dockerfile -t flask-image:latest

3. TO RUN THE IMAGE TO CREATE CONTAINER
   docker run -ti flask-image

4. TO TAG THE IMAGE
   docker tag <image id> dhanisha/finalprojectflask

5. TO PUSH THE DOCKER IMAGE TO DOCKER HUB
   docker push dhanisha/finalprojectflask

6. TO PULL THE DOCKER IMAGE FROM THE DOCKER HUB
   docker pull dhanisha/finalprojectflask

7. TO RUN THE IMAGE PULLED FROM DOCKER HUB

```
docker run -ti flasj-images
```

Conclusion: We could successfully deploy an Airline delay prediction application for the user on AWS ec2 instances.