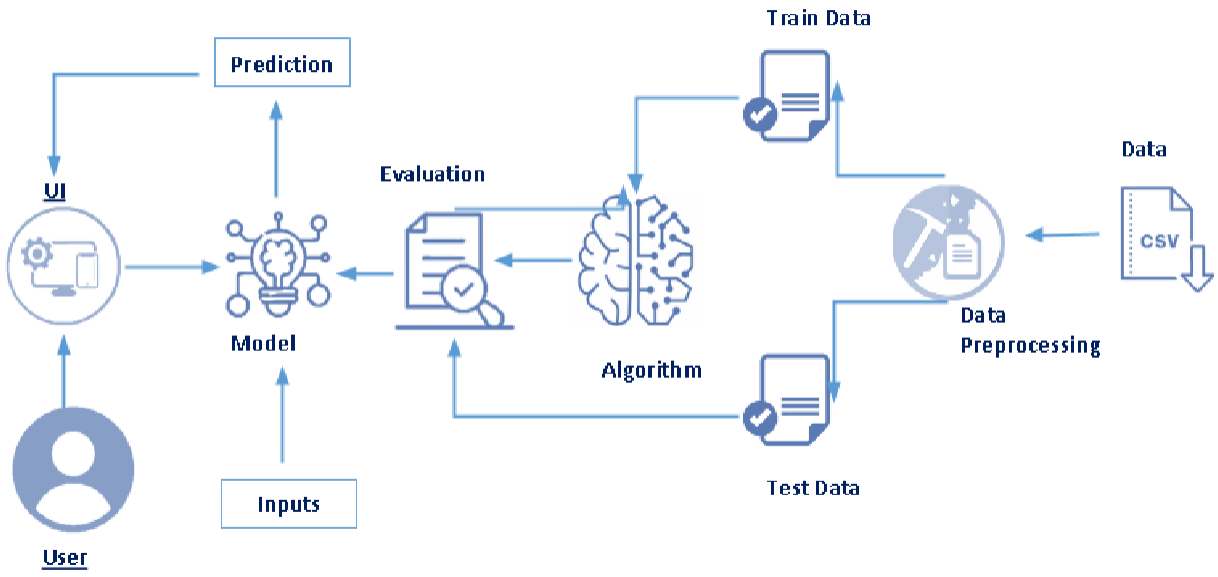
Intelligent Admissions: Flight Delay Prediction for aviation Industry using Machine Learning

1.1 OVERVIEW:

The input to our algorithm is rows of feature vector like departure date, departure delay, distance between the two airports, scheduled arrival time etc. We then use decision tree classifier to predict if the flight arrival will be delayed or not. A flight is delayed when difference between scheduled and actual arrival times is greater than 15 minutes. Furthermore, we compare decision tree classifier with logistic regression and a simple neural network for various figures of merit. Finally, it will be integrated to web based application.

AI Expand into these core university practices, new concerns are also being raised about the tool's threats to personal privacy and it ability to Systematic bias.

Technical Architecture:



A PROJECT DESCRIPTION:

Flight delay is inevitable and it plays an important role in both profits and loss of the airlines. An accurate estimation of flight delay is critical for airlines because the results can be applied to increase customer satisfaction and incomes of airline agencies. There have been many researches on modeling and predicting flight delays, where most of them have been trying to predict the delay through extracting important characteristics and most related features. However, most of the proposed methods are not accurate enough because of massive volume data, dependencies and extreme number of parameters.

This paper proposes a model for predicting flight delay based on Deep Learning (DL). DL is one of the newest methods employed in solving problems with high level of complexity and massive amount of data. Moreover, DL is capable to automatically extract the important features from data. Furthermore, due to the fact that most of flight delay data are noisy, a technique based on stack noising auto encoder is designed and added to the proposed model.

Also, algorithm is applied to find weight and bias proper values, and finally the output has been optimized to produce high accurate results. In order to study effect of stack noising auto encoder and LM algorithm on the model structure, two other structures are also designed.

Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analysis by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

to accomplish this, we have to complete all the activities listed below.

- Define Problem / Problem Understanding
  - Specify the business problem
  - Business requirements
  - Literature Survey
  - Social (or) Business Impact
- Data Collection & Preparation
  - Collect the dataset
  - Data Preparation
- Exploratory Data Analysis
  - Descriptive statistical
  - Visual Analysis
- Model Building
  - Training the model in multiple algorithms

- Testing the model

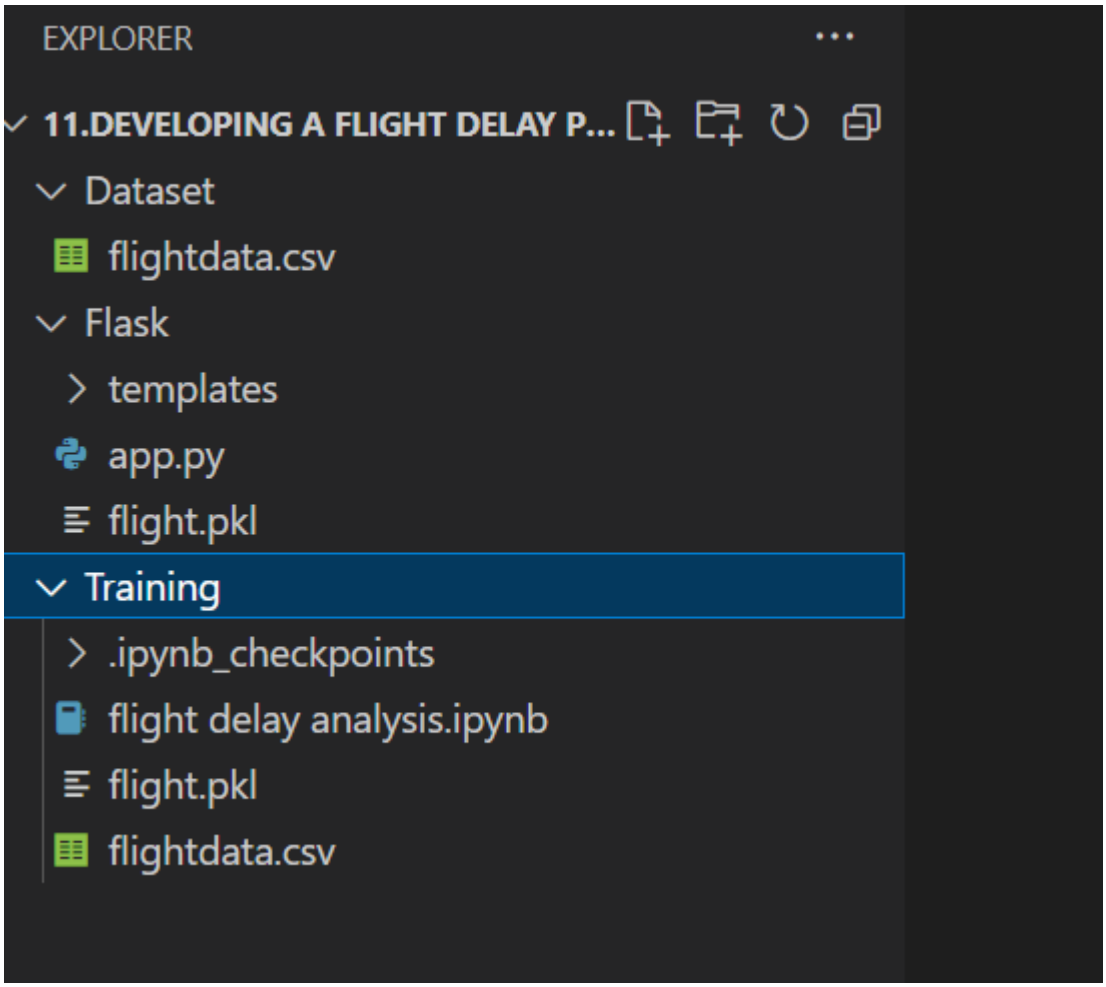
- Performance Testing & Hyper-parameter Tuning
  - Testing model with multiple evaluation metrics
  - Comparing model accuracy before & after applying hyper-parametertuning

- Model Deployment
  - Save the best model
  - Integrate with Web Framework

- Project Demonstration & Documentation
  - Record explanation Video for project end to end solution
  - Project Documentation-Step by step project development procedure

Project Structure:

Create the Project folder which contains files as shown below



● We are building a flask application which needs HTML pages stored in the templates
Folder and a python script app.py for scripting.
Flight.pkl is our saved model. Further we will use this model for flask integration.
● Training folder contains a model training file.

1.2 PURPOSE:

As most frequent flyers already know, extreme weather events can cause flight delays, and even flight cancellations in some cases. Although planes are equipped to take off, fly, and land in all types of weather, sometimes pilots must be far more cautious in certain scenarios.

A weather condition in the west can affect the flights in the east, and vice versa. High winds and sleet are two additional factors in weather delays.

Just like there's traffic during rush hour on land, air traffic can get heavy during rush hour, as well. A crowded or exceptionally busy airport can have planes lined up on the runway waiting to take off. This leaves many planes circling the airport, waiting on their clearance to land.

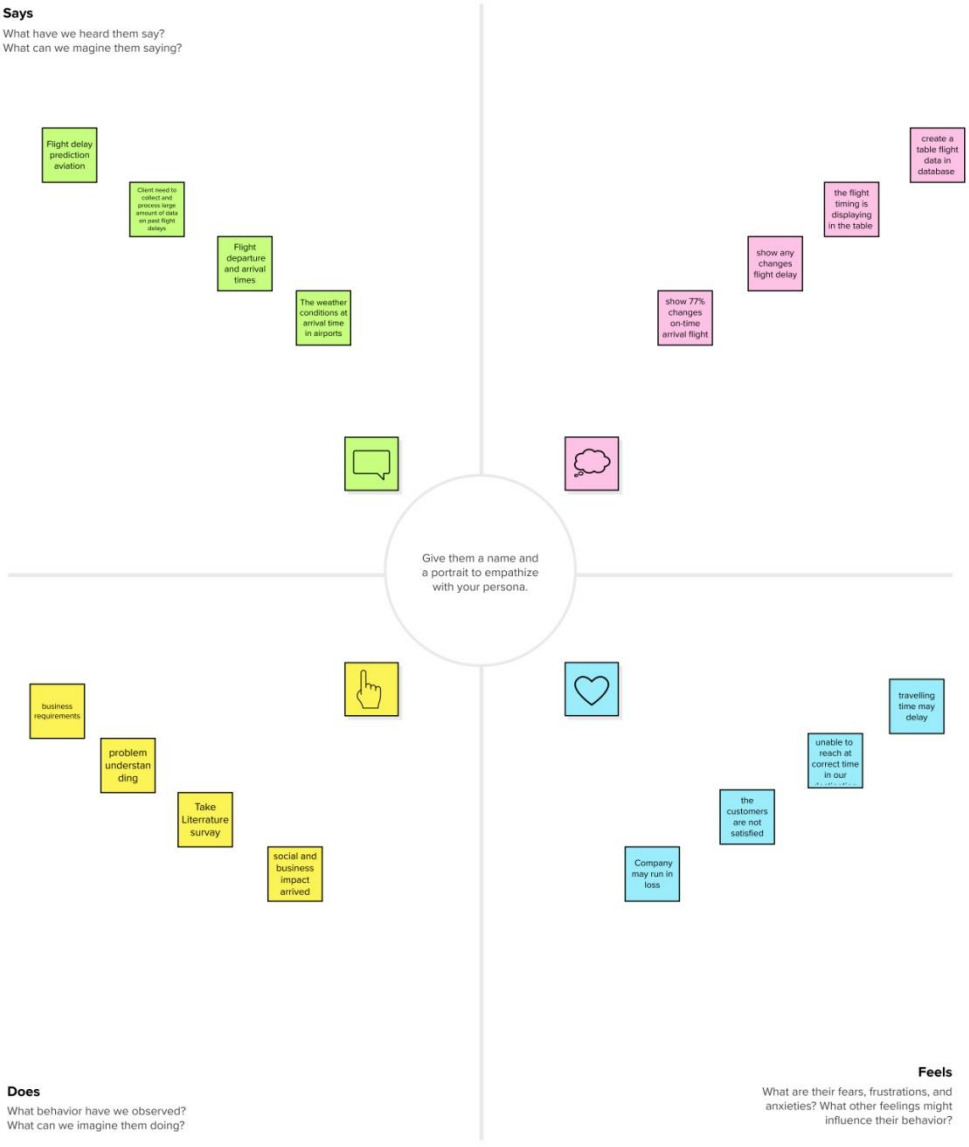
2. PROBLEM DEFINITION AND DESIGN THINKING

When you're in the airport, two times hover at the front of your brain, the part probably reserved in pre-civilized times for "home" and "water source" — the **boarding time** and the **departure time**. As passengers, we assume these times are fixed, especially the departure time. There's a word for what happens when the departure time changes: *delay*. But what happens when airlines don't see that departure time as written in stone.

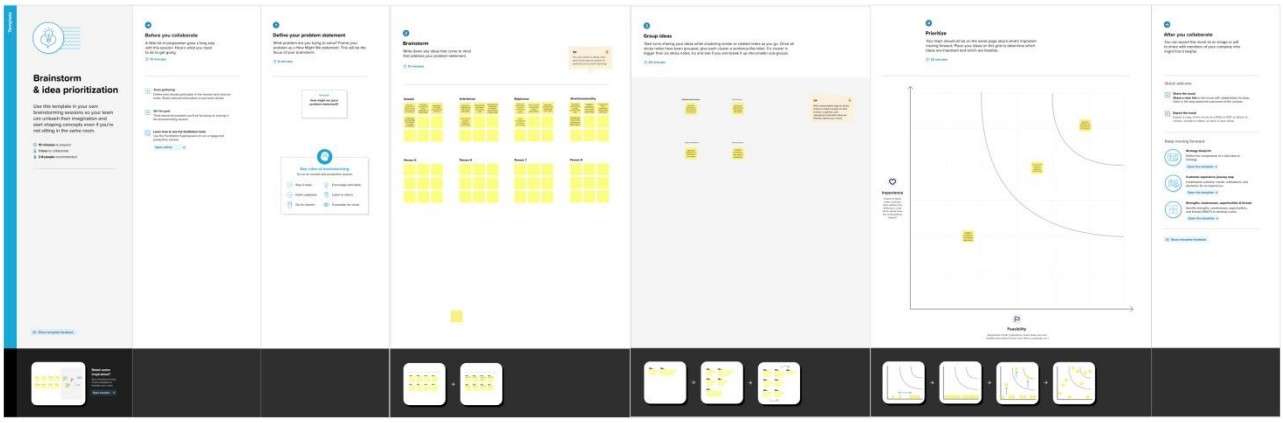
2.1 EMPATHY MAP:

Build empathy

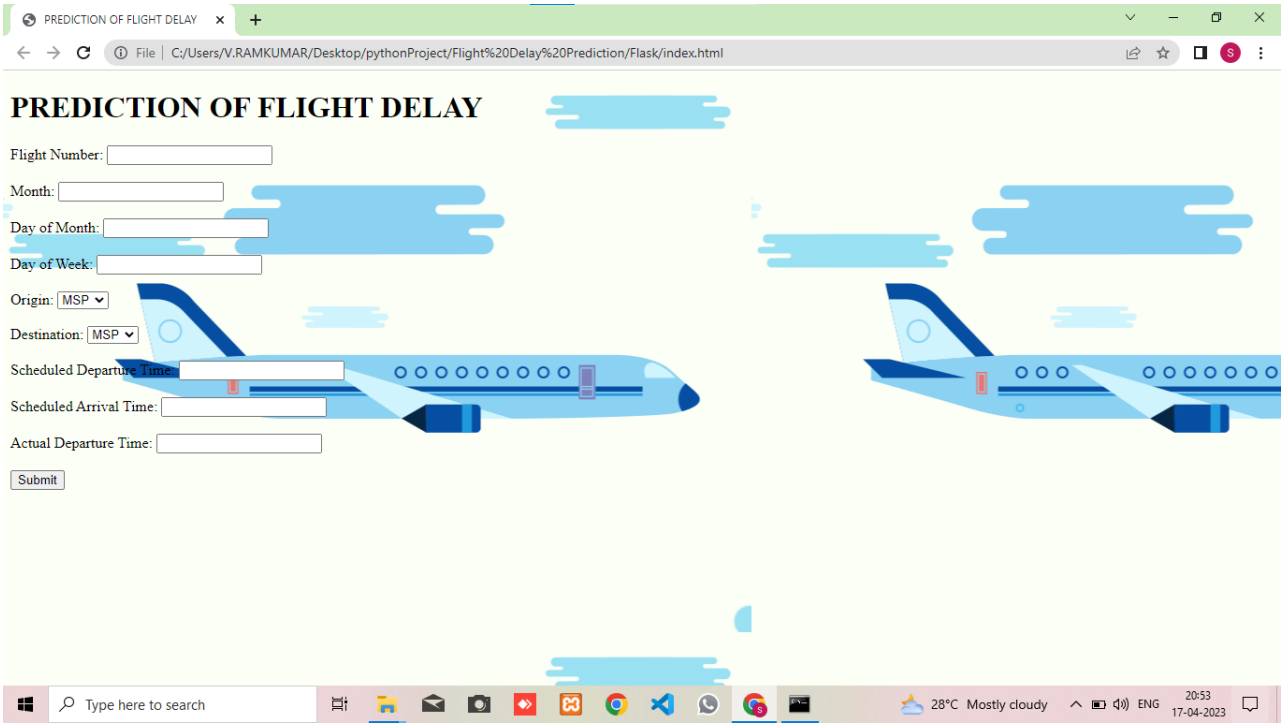
The information you add here should be representative of the observations and research you've done about your users.



2.2Ideation and Brain storming Map:



3. RESULT:



4. ADVANTAGE AND DISADVANTAGE:

Advantage:

1. You can enjoy your holiday destinations for longer

Even if it is at the airport, you can enjoy your holiday destination for a little longer. Enjoy the local food, write and send a holiday card from the airport or sit back and relax and enjoy the vibrant life at the airport.

2. You have a right to care

The wait that comes with a flight delay can last long… Luckily, in case of a delay of two hours or more, or with a delay due to exceptional circumstances, you are entitled to care. Food and drinks should be provided by the airline, usually by vouchers. If not, make sure to save your receipts to claim the extra costs from the airline. In case of a delay that forces you to take an overnight stay, you are also entitled to a hotel, including the transport to the accommodation.

3. You have time to sort out your holiday photos

Reality teaches us that once you are home, you often do not take time to sort out your holiday photos... let alone make a photo album. Are you delayed? Then sort them out to create that photo album back home that you are so looking forward to!

Disadvantage:

Quantifying delay statistics

Starting from a histogram of the flight delays, we derive three indices/measures to quantify flight delay distributions: Mean delay, exponent of left exponential and power-law exponent of right q-exponential, as explained below in detail. We will use the LHR data previous to any COVID-19 influence as our main example.

Methods

As a first step, we split the full histogram at its peak value into two histograms, a left flank of predominantly negative delays and a right flank of predominantly positive delays, see Fig. 2. Based on the shape of the empirical distributions, we use exponentials and q-exponentials as fitting functions, see also "Methods" for details. Splitting the histogram has two advantages: Firstly, the analysis of each flank is much simpler than the analysis of the full aggregated data. Secondly, a given stakeholder might be particularly interested in positive rather than negative delays, or vice versa.

Source code

```
[ ] import pandas as pd
    import numpy as np
    import pickle
    import matplotlib.pyplot as plt
    %matplotlib inline
    import seaborn as sns
    import sklearn
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.ensemble import GradientBoostingClassifier,RandomForestClassifier
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.model_selection import RandomizedSearchCV
    import imblearn
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
    from sklearn.metrics import accuracy_score,classification_report, confusion_matrix,f1_score
```

```
[ ]
```

```
dataset=pd.read_csv("/content/flightdata.csv")
dataset.head()
```

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	UNIQUE_CARRIER	TAIL_NUM	FL_NUM	ORIGIN_AIRPORT_ID	ORIGIN	...	CRS_ARR_TIME	ARR_TIME	ARR_DELAY	ARR_DEL15	CANCELLED	DIVERTED	CRS_ELAPSED_TIME	ACTUAL
0	2016	1	1	1	5	DL	N836DN	1399	10397	ATL	...	2143	2102.0	-41.0	0.0	0.0	0.0	338.0	
1	2016	1	1	1	5	DL	N964DN	1476	11433	DTW	...	1435	1439.0	4.0	0.0	0.0	0.0	110.0	
2	2016	1	1	1	5	DL	N813DN	1597	10397	ATL	...	1215	1142.0	-33.0	0.0	0.0	0.0	335.0	
3	2016	1	1	1	5	DL	N587NW	1768	14747	SEA	...	1335	1345.0	10.0	0.0	0.0	0.0	196.0	
4	2016	1	1	1	5	DL	N836DN	1823	14747	SEA	...	607	615.0	8.0	0.0	0.0	0.0	247.0	

5 rows × 26 columns

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11231 entries, 0 to 11230
Data columns (total 26 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   YEAR                 11231 non-null  int64
 1   QUARTER              11231 non-null  int64
 2   MONTH                11231 non-null  int64
 3   DAY_OF_MONTH         11231 non-null  int64
 4   DAY_OF_WEEK          11231 non-null  int64
 5   UNIQUE_CARRIER       11231 non-null  object
 6   TAIL_NUM             11231 non-null  object
 7   FL_NUM               11231 non-null  int64
 8   ORIGIN_AIRPORT_ID    11231 non-null  int64
 9   ORIGIN               11231 non-null  object
 10  DEST_AIRPORT_ID      11231 non-null  int64
 11  DEST                 11231 non-null  object
 12  CRS_DEP_TIME         11231 non-null  int64
 13  DEP_TIME             11124 non-null  float64
 14  DEP_DELAY            11124 non-null  float64
 15  DEP_DEL15            11124 non-null  float64
 16  CRS_ARR_TIME         11231 non-null  int64
 17  ARR_TIME             11116 non-null  float64
 18  ARR_DELAY            11043 non-null  float64
 19  ARR_DEL15            11043 non-null  float64
 20  CANCELLED            11231 non-null  float64
 21  DIVERTED             11231 non-null  float64
 22  CRS_ELAPSED_TIME     11231 non-null  float64
 23  ACTUAL_ELAPSED_TIME  11043 non-null  float64
 24  DISTANCE             11231 non-null  float64
 25  Unnamed: 25          0 non-null      float64
dtypes: float64(12), int64(10), object(4)
memory usage: 2.2+ MB
```

```
[ ] dataset=dataset.drop ( 'Unnamed: 25' ,  axis= 1 )
    dataset.isnull () . sum()
```

```
[ ]
```

```
YEAR                   0
QUARTER                0
MONTH                  0
DAY_OF_MONTH           0
DAY_OF_WEEK            0
UNIQUE_CARRIER         0
TAIL_NUM               0
FL_NUM                 0
ORIGIN_AIRPORT_ID      0
ORIGIN                 0
DEST_AIRPORT_ID        0
DEST                   0
CRS_DEP_TIME           0
DEP_TIME             107
DEP_DELAY            107
DEP_DEL15            107
CRS_ARR_TIME           0
ARR_TIME             115
ARR_DELAY            188
ARR_DEL15            188
CANCELLED              0
DIVERTED               0
CRS_ELAPSED_TIME       0
ACTUAL_ELAPSED_TIME  188
DISTANCE               0
dtype: int64
```

```
[ ] dataset['ARR_DELAY'].fillna(dataset['ARR_DELAY'].median(),inplace=True)
```

```
[ ] dataset = dataset[["FL_NUM","MONTH","DAY_OF_MONTH","DAY_OF_WEEK","ORIGIN","DEST","CRS_ARR_TIME","DEP_DEL15","ARR_DEL15"]]
```

```
dataset.isnull().sum()
```

```
FL_NUM          0
MONTH           0
DAY_OF_MONTH    0
DAY_OF_WEEK     0
ORIGIN          0
DEST            0
CRS_ARR_TIME    0
DEP_DEL15     107
ARR_DEL15     188
dtype: int64
```

```
dataset[dataset.isnull().any(axis=1)].head(10)
```

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15
177	2834	1	9	6	MSP	SEA	852	0.0	NaN
179	86	1	10	7	MSP	DTW	1632	NaN	NaN
184	557	1	10	7	MSP	DTW	912	0.0	NaN
210	1096	1	10	7	DTW	MSP	1303	NaN	NaN
478	1542	1	22	5	SEA	JFK	723	NaN	NaN
481	1795	1	22	5	ATL	JFK	2014	NaN	NaN
491	2312	1	22	5	MSP	JFK	2149	NaN	NaN
499	423	1	23	6	JFK	ATL	1600	NaN	NaN
600	425	1	23	6	JFK	ATL	1827	NaN	NaN
601	427	1	23	6	JFK	SEA	1053	NaN	NaN

```
dataset['DEP_DEL15'].mode()
```

```
0    0.0
Name: DEP_DEL15, dtype: float64
```

```
dataset=dataset.fillna({'ARR_DEL15':1})
dataset=dataset.fillna({'DEP_DEL15':0})
dataset.iloc[177:185]
```

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15
177	2834	1	9	6	MSP	SEA	852	0.0	1.0
178	2839	1	9	6	DTW	JFK	1724	0.0	0.0
179	86	1	10	7	MSP	DTW	1632	0.0	1.0
180	87	1	10	7	DTW	MSP	1649	1.0	0.0
181	423	1	10	7	JFK	ATL	1600	0.0	0.0
182	440	1	10	7	JFK	ATL	849	0.0	0.0
183	485	1	10	7	JFK	SEA	1945	1.0	0.0
184	557	1	10	7	MSP	DTW	912	0.0	1.0

```
import math
```

```
for index,row in dataset.iterrows():
  dataset.loc[index, 'CRS_ARR_TIME']=math.floor(row['CRS_ARR_TIME']/100)
dataset.head()
```

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15
0	1399	1	1	5	ATL	SEA	21	0.0	0.0
1	1476	1	1	5	DTW	MSP	14	0.0	0.0
2	1597	1	1	5	ATL	SEA	12	0.0	0.0
3	1768	1	1	5	SEA	MSP	13	0.0	0.0
4	1823	1	1	5	SEA	DTW	6	0.0	0.0

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
dataset['DEST']=le.fit_transform (dataset['DEST'])
dataset['ORIGIN']=le.fit_transform (dataset['ORIGIN'])
```

```
dataset.head(5)
```

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15
0	1399	1	1	5	0	4	21	0.0	0.0
1	1476	1	1	5	1	3	14	0.0	0.0
2	1597	1	1	5	0	4	12	0.0	0.0
3	1768	1	1	5	4	3	13	0.0	0.0
4	1823	1	1	5	4	1	6	0.0	0.0

```
dataset['ORIGIN'].unique()
```

```
array([0, 1, 4, 3, 2])
```

```
dataset = pd.get_dummies(dataset, columns=['ORIGIN','DEST'])
dataset.head()
```

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15	ORIGIN_0	ORIGIN_1	ORIGIN_2	ORIGIN_3	ORIGIN_4	DEST_0	DEST_1	DEST_2	DEST_3	DEST_4
0	1399	1	1	5	21	0.0	0.0	1	0	0	0	0	0	0	0	0	1
1	1476	1	1	5	14	0.0	0.0	0	1	0	0	0	0	0	0	1	0
2	1597	1	1	5	12	0.0	0.0	1	0	0	0	0	0	0	0	0	1
3	1768	1	1	5	13	0.0	0.0	0	0	0	0	1	0	0	0	1	0
4	1823	1	1	5	6	0.0	0.0	0	0	0	0	1	0	1	0	0	0

```
x=dataset.iloc[:,0:8].values

y=dataset.iloc[:,8:9].values
```

```
x
```

```
array([[1.399e+03, 1.000e+00, 1.000e+00, ..., 0.000e+00, 0.000e+00,
        1.000e+00],
       [1.476e+03, 1.000e+00, 1.000e+00, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [1.597e+03, 1.000e+00, 1.000e+00, ..., 0.000e+00, 0.000e+00,
        1.000e+00],
       ...,
       [1.823e+03, 1.200e+01, 3.000e+01, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [1.901e+03, 1.200e+01, 3.000e+01, ..., 0.000e+00, 0.000e+00,
        1.000e+00],
       [2.005e+03, 1.200e+01, 3.000e+01, ..., 0.000e+00, 0.000e+00,
        1.000e+00]])
```

```
from sklearn.preprocessing import OneHotEncoder
oh= OneHotEncoder()
z=oh.fit_transform(x[:,4:5]).toarray()
t=oh.fit_transform(x[:,5:6]).toarray()
```

```
z
```

```
array([[0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

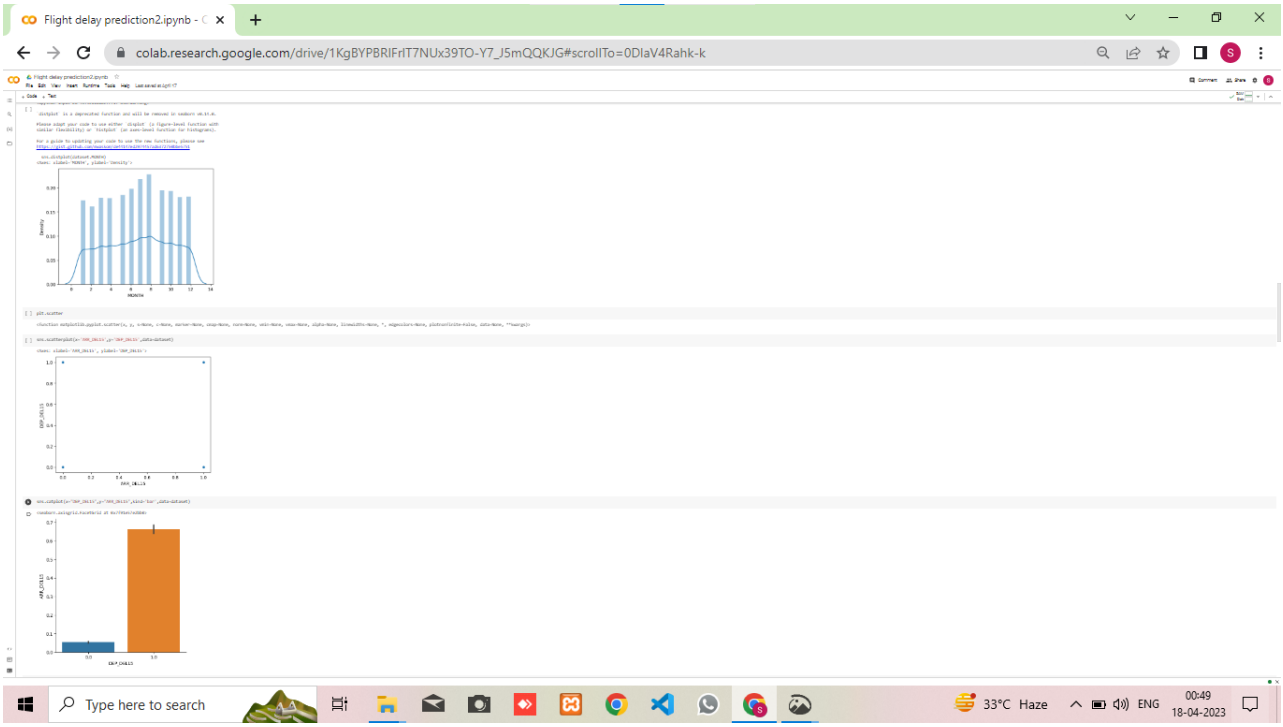
```
t
```

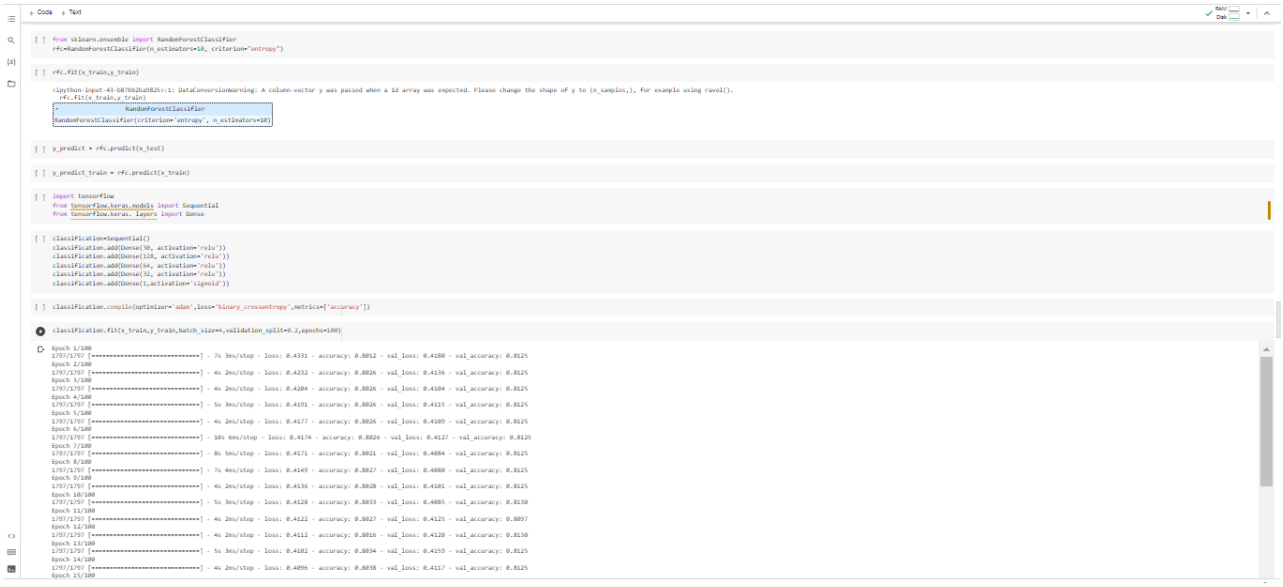
```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [1., 0.],
       [1., 0.],
       [1., 0.]])
```

```
x=np.delete(x,[4,5],axis=1)
```

```
dataset.describe()
```

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15	ORIGIN_0	ORIGIN_1	ORIGIN_2	ORIGIN_3	ORIGIN_4	DEST_0	DEST_1	DEST_2	DEST_3	DEST_4
count	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000
mean	1334.325617	6.628973	15.790758	3.950199	15.067314	0.141483	0.139168	0.276022	0.195975	0.122340	0.225982	0.179681	0.286795	0.196866	0.116820	0.221975	0.177544
std	811.875227	3.354678	8.782056	1.995257	5.023534	0.348535	0.346138	0.447048	0.396967	0.327693	0.418246	0.383939	0.452285	0.397648	0.321219	0.415593	0.382146
min	7.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	624.000000	4.000000	8.000000	2.000000	11.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	1267.000000	7.000000	16.000000	4.000000	15.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	2032.000000	9.000000	23.000000	6.000000	19.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000
max	2853.000000	12.000000	31.000000	7.000000	23.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000





```
[ ] from sklearn.ensemble import RandomForestClassifier
    rfc=RandomForestClassifier(n_estimators=10, criterion="entropy")

[ ] rfc.fit(x_train,y_train)

    <ipython-input-43-b87bb2ba2825>:1: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
      rfc.fit(x_train,y_train)
    RandomForestClassifier(criterion='entropy', n_estimators=10)

[ ] y_predict = rfc.predict(x_test)

[ ] y_predict_train = rfc.predict(x_train)

[ ] import tensorflow
    from tensorflow.keras.models import Sequential
    from tensorflow.keras. layers import Dense

[ ] classification=Sequential()
    classification.add(Dense(30, activation='relu'))
    classification.add(Dense(128, activation='relu'))
    classification.add(Dense(64, activation='relu'))
    classification.add(Dense(32, activation='relu'))
    classification.add(Dense(1,activation='sigmoid'))

[ ] classification.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

    classification.fit(x_train,y_train,batch_size=4,validation_split=0.2,epochs=100)

    Epoch 1/100
    1797/1797 [==============================] - 7s 3ms/step - loss: 0.4331 - accuracy: 0.8012 - val_loss: 0.4180 - val_accuracy: 0.8125
    Epoch 2/100
    1797/1797 [==============================] - 4s 2ms/step - loss: 0.4232 - accuracy: 0.8026 - val_loss: 0.4136 - val_accuracy: 0.8125
    Epoch 3/100
    1797/1797 [==============================] - 4s 2ms/step - loss: 0.4204 - accuracy: 0.8026 - val_loss: 0.4104 - val_accuracy: 0.8125
    Epoch 4/100
    1797/1797 [==============================] - 5s 3ms/step - loss: 0.4191 - accuracy: 0.8026 - val_loss: 0.4115 - val_accuracy: 0.8125
    Epoch 5/100
    1797/1797 [==============================] - 4s 2ms/step - loss: 0.4177 - accuracy: 0.8026 - val_loss: 0.4109 - val_accuracy: 0.8125
    Epoch 6/100
    1797/1797 [==============================] - 10s 6ms/step - loss: 0.4174 - accuracy: 0.8026 - val_loss: 0.4127 - val_accuracy: 0.8125
    Epoch 7/100
    1797/1797 [==============================] - 8s 5ms/step - loss: 0.4171 - accuracy: 0.8021 - val_loss: 0.4084 - val_accuracy: 0.8125
    Epoch 8/100
    1797/1797 [==============================] - 7s 4ms/step - loss: 0.4149 - accuracy: 0.8027 - val_loss: 0.4080 - val_accuracy: 0.8125
    Epoch 9/100
    1797/1797 [==============================] - 4s 2ms/step - loss: 0.4136 - accuracy: 0.8028 - val_loss: 0.4101 - val_accuracy: 0.8125
    Epoch 10/100
    1797/1797 [==============================] - 5s 3ms/step - loss: 0.4128 - accuracy: 0.8033 - val_loss: 0.4085 - val_accuracy: 0.8130
    Epoch 11/100
    1797/1797 [==============================] - 4s 2ms/step - loss: 0.4122 - accuracy: 0.8027 - val_loss: 0.4125 - val_accuracy: 0.8097
    Epoch 12/100
    1797/1797 [==============================] - 4s 2ms/step - loss: 0.4112 - accuracy: 0.8016 - val_loss: 0.4120 - val_accuracy: 0.8130
    Epoch 13/100
    1797/1797 [==============================] - 5s 3ms/step - loss: 0.4102 - accuracy: 0.8034 - val_loss: 0.4159 - val_accuracy: 0.8125
    Epoch 14/100
    1797/1797 [==============================] - 4s 2ms/step - loss: 0.4096 - accuracy: 0.8038 - val_loss: 0.4117 - val_accuracy: 0.8125
    Epoch 15/100
```

File Edit View Insert Runtime Tools Help Last saved at April 17

```
y_pred = classifier.predict([[129,99,1,0,0,1]])
```

```
print(y_pred)
(y_pred)
```

```
[0]
array([0], dtype=uint8)
```

```
y_pred = rfc.predict([[129,99,1,0,0,1]])
```

```
print(y_pred)
(y_pred)
```

```
[0]
array([0], dtype=uint8)
```

```
classification.save("flight.h5")
```

```
y_pred=classification.predict(x_test)
```

```
71/71 [==============================] - 0s 1ms/step
```

```
y_pred
```

```
array([[0.9451136 ],
       [0.17883752],
       [0.        ],
       ...,
       [0.35195875],
       [0.        ],
       [0.27439657]], dtype=float32)
```

```
y_pred = (y_pred > 0.5)
y_pred
```

```
array([[ True],
       [False],
       [False],
       ...,
       [False],
       [False],
       [False]])
```

```
def predict_exit(sample_value):
  sample_value = np.array(sample_value)
  sample_value=sample_value.reshape(1, -1)
  sample_value=sc.transform(sample_value)
  return classifier.predict(sample_value)
```

```
test=classification.predict([[1,1,121.000000,36.0,0,1]])
if test==1:
  print('Prediction: Chance of delay')
else:
  print('Prediction: No chance of delay.')
```

```
    dfs = []
models = [
        ('RF',RandomForestClassifier()),
        ('DecisionTree',DecisionTreeClassifier()),
        ('ANN',MLPClassifier())
        ]
results = []
    names = []
    scoring = ['accuracy','precision_weighted','recall_weighted','f1_weighted', 'roc_auc']
    target_names = ['no delay', 'delay']
    for name, model in models:
      kfold=model_selection.KFold(n_splits=5, shuffle=True, random_state=90210)
      cv_results = model_selection.cross_validate(model, x_train, y_train, cv=kfold, scoring=scoring)
      clf = model.fit(x_train, y_train)
      y_pred=clf.predict(x_test)
      print(name)
      print(classification_report (y_test, y_pred, target_names=target_names))
      results.append(cv_results)
      names.append(name)
      this_df = pd.DataFrame(cv_results)
      this_df['model'] = name
      dfs.append(this_df)
final = pd.concat(dfs, ignore_index=True)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
<ipython-input-61-ff794698e872>:14: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  clf = model.fit(x_train, y_train)
RF
              precision    recall  f1-score   support

    no delay       0.85      0.93      0.89      1802
       delay       0.54      0.34      0.41       445

    accuracy                           0.81      2247
   macro avg       0.69      0.63      0.65      2247
weighted avg       0.79      0.81      0.79      2247

DecisionTree
              precision    recall  f1-score   support

    no delay       0.99      0.99      0.99      1802
       delay       0.96      0.96      0.96       445

    accuracy                           0.98      2247
   macro avg       0.98      0.98      0.98      2247
weighted avg       0.98      0.98      0.98      2247

/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:1098: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:1098: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:1098: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
```

```
print('Training accuracy:',accuracy_score(y_train,y_predict_train))
print('Training accuracy:',accuracy_score(y_test,y_predict))
```

```
Training accuracy: 0.9841941228851291
Training accuracy: 0.8064085447263017
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test, y_predict)
cm
```

```
array([[1682,  120],
       [ 315,  130]])
```

```
from sklearn.metrics import accuracy_score
desacc=accuracy_score(y_predict,decisiontree)
```

```
desacc
```

```
0.8081886960391633
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,decisiontree)
```

```
cm
```

```
array([[1786,   16],
       [  20,  425]])
```

```
from sklearn.metrics import accuracy_score,classification_report
score=accuracy_score(y_pred,y_test)
print('The accuracy for ANN model is:{}%'.format(score*100))
```

```
The accuracy for ANN model is:79.8842901646664%
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm
```

```
array([[1774,   28],
       [ 424,   21]])
```

```
parameters = {
              'n_estimators' : [1,20,30,55,68,74,90,120,115],
               'criterion':['gini','entropy'],
               'max_features' : ["auto", "sqrt", "log2"],
        'max_depth' : [2,5,8,10], 'verbose' : [1,2,3,4,6,8,9,10]


}
```

```
rf = RandomForestClassifier()
```

```
RCV = RandomizedSearchCV(estimator=rf,param_distributions=parameters, cv=10,n_iter=4)
```

```
print('Training accuracy:',accuracy_score(y_train,y_predict_train))
print('Training accuracy:',accuracy_score(y_test,y_predict1))
```

```
Training accuracy: 0.9841941228851291
Training accuracy: 0.8864085447263017
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test, y_predict)
cm
```

```
array([[1682,  120],
       [ 315,  130]])
```

```
from sklearn.metrics import accuracy_score
desacc=accuracy_score(y_predict,decisiontree)
```

```
desacc
```

```
0.8881886960391633
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,decisiontree)
```

```
cm
```

```
array([[1786,   16],
       [  20,  425]])
```

```
from sklearn.metrics import accuracy_score,classification_report
score=accuracy_score(y_pred,y_test)
print('The accuracy for ANN model is:{}%'.format(score*100))
```

```
The accuracy for ANN model is:79.88429016466645%
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm
```

```
array([[1774,   28],
       [ 424,   21]])
```

```
parameters = {
              'n_estimators' : [1,20,30,55,68,74,90,120,115],
               'criterion':['gini','entropy'],
               'max_features' : ["auto", "sqrt", "log2"],
        'max_depth' : [2,5,8,10], 'verbose' : [1,2,3,4,6,8,9,10]


        }
```

```
rf = RandomForestClassifier()
```

```
RCV = RandomizedSearchCV(estimator=rf,param_distributions=parameters, cv=10,n_iter=4)
```

```
bt_params=RCV.best_params_
bt_score=RCV.best_score_
```

```
bt_params
```

```
{'verbose': 10,
 'n_estimators': 68,
 'max_features': 'log2',
 'max_depth': 10,
 'criterion': 'entropy'}
```

```
bt_score
```

```
0.8213481448882625
```

```
model = RandomForestClassifier(verbose= 10, n_estimators= 120, max_features= 'log2',max_depth= 10,criterion= 'entropy')
RCV.fit(x_train,y_train)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  estimator.fit(X_train, y_train, **fit_params)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    0.0s remaining:    0.0s
building tree 1 of 120
building tree 2 of 120
building tree 3 of 120
building tree 4 of 120
building tree 5 of 120
building tree 6 of 120
building tree 7 of 120
building tree 8 of 120
building tree 9 of 120
building tree 10 of 120
building tree 11 of 120
building tree 12 of 120
building tree 13 of 120
building tree 14 of 120
building tree 15 of 120
building tree 16 of 120
building tree 17 of 120
building tree 18 of 120
building tree 19 of 120
building tree 20 of 120
building tree 21 of 120
building tree 22 of 120
building tree 23 of 120
building tree 24 of 120
building tree 25 of 120
building tree 26 of 120
building tree 27 of 120
building tree 28 of 120
building tree 29 of 120
building tree 30 of 120
building tree 31 of 120
```

```
building tree 29 of 120
building tree 30 of 120
building tree 31 of 120
building tree 32 of 120
building tree 33 of 120
building tree 34 of 120
building tree 35 of 120
building tree 36 of 120
building tree 37 of 120
building tree 38 of 120
building tree 39 of 120
building tree 40 of 120
building tree 41 of 120
building tree 42 of 120
building tree 43 of 120
```

```
y_predict_rf=RCV.predict(x_test)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  74 out of  74 | elapsed:    0.0s finished
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
RFC=accuracy_score(y_test,y_predict_rf)
RFC
```

```
0.8268802848421
```

```
import pickle
pickle.dump(RCV,open('flight.pkl','wb'))
```

```
# importing the necessary dependencies
from flask import Flask,request,render_template
import numpy as np
import pandas as pd
import pickle
import os
```

```
model = pickle.load(open('flight.pkl','rb'))

app = Flask(__name__)#initializing the app
```

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


@app.route('/prediction',methods =['POST'])
```

```
destination = request.form['destination']
if(destination == "msp"):
    destination1,destination2,destination3,destination4,destination5 = 0,0,0,0,1
if(destination == "dtw"):
    destination1,destination2,destination3,destination4,destination5 = 1,0,0,0,0
if(destination == "jfk"):
    destination1,destination2,destination3,destination4,destination5 = 0,0,1,0,0
if(destination == "sea"):
    destination1,destination2,destination3,destination4,destination5 = 0,1,0,0,0
if(destination == "alt"):
    destination1,destination2,destination3,destination4,destination5 = 0,0,0,1,0
dept = request.form['dept']
arrtime = request.form['arrtime']
actdept = request.form['actdept']
dept15=int(dept)-int(actdept)
total = [[name,month,dayofmonth,dayofweek,origin1,origin2,origin3,origin4,orgin5,destination1,destination2,destination3,destination4,destination5,i
#print(total)
y_pred = model.predict(total)

print(y_pred)

if(y_pred==[0.]):
    ans="The Flight will be on time"
else:
    ans="The Flight will be delayed"
return render_template("index.html",showcase = ans)
```

```
destination = request.form['destination']
if(destination == "msp"):
    destination1,destination2,destination3,destination4,destination5 = 0,0,0,0,1
if(destination == "dtw"):
    destination1,destination2,destination3,destination4,destination5 = 1,0,0,0,0
if(destination == "jfk"):
    destination1,destination2,destination3,destination4,destination5 = 0,0,1,0,0
if(destination == "sea"):
    destination1,destination2,destination3,destination4,destination5 = 0,1,0,0,0
if(destination == "alt"):
    destination1,destination2,destination3,destination4,destination5 = 0,0,0,1,0
dept = request.form['dept']
arrtime = request.form['arrtime']
actdept = request.form['actdept']
dept15=int(dept)-int(actdept)
total = [[name,month,dayofmonth,dayofweek,origin1,origin2,origin3,origin4,orgin5,destination1,destination2,destination3,destination4,destination5
#print(total)
y_pred = model.predict(total)

print(y_pred)

if(y_pred==[0.]):
    ans="The Flight will be on time"
else:
    ans="The Flight will be delayed"
return render_template("index.html",showcase = ans)
```

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

Result