SPRINT – 4 PROJECT DOCUMENT

Date	18 October 2022
Team ID	PNT2022TMID15779
Project Name	Flight Delay Prediction Using Machine Learning

DEVELOPMENT PHASE:

Outline:

1. Data Pre-processing
2. EDA/Data Analysis
3. Feature Engineering
4. Model Building
5. Saving Best Model

Required Libraries:

- Pandas - Data Pre-processing
- Numpy - Data Pre-processing, Analysis
- Matplotlib - Visualization
- Seaborn - Visualization
- Sklearn - Model Building
- Pickle - Model saving

Software/Tool:

- Anaconda- Jupyter Notebook
- Used Language Python

Data Pre-processing:

Data Collection:

Dataset is collected from the IBM career smartinternz portal in Guided Project.

Dataset description:

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)

Columns Description:

Dest means Destination Airport.

Crs_dep_time and crs_arr_time is planned departure and arrival time.

Crs_elapsed _time is estimated travel time as per plan.

Arr_time and dep_time are actual arrival and departure time.

Actual_elapsed_time is actual travelled time

To pre-process our dataset, we need to import above mentioned required libraries, then import data using pandas.

This data does not contain any duplicated values and null values except in arrival , departure time columns, because these left empty when flights are cancelled.

Descriptive Analytics:

In [19]:
```
data1.describe()
```

Out[19]:

	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	FL_NUM	CRS_DEP_TIME.1	DEP_DELAY	DEP_DEL15	CRS_ARR_TIME.1	ARR_DEL
count	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11124.000000	11124.000000	11231.000000	11043.000
mean	2.544475	6.628973	15.790758	3.960199	1334.325617	1320.798326	8.460266	0.142844	1537.312795	-2.573
std	1.090701	3.354678	8.782056	1.995257	811.875227	490.737845	36.762969	0.349930	502.512494	39.232
min	1.000000	1.000000	1.000000	1.000000	7.000000	10.000000	-16.000000	0.000000	2.000000	-67.000
25%	2.000000	4.000000	8.000000	2.000000	624.000000	905.000000	-3.000000	0.000000	1130.000000	-19.000
50%	3.000000	7.000000	16.000000	4.000000	1267.000000	1320.000000	-1.000000	0.000000	1559.000000	-10.000
75%	3.000000	9.000000	23.000000	6.000000	2032.000000	1735.000000	4.000000	0.000000	1952.000000	1.000
max	4.000000	12.000000	31.000000	7.000000	2853.000000	2359.000000	645.000000	1.000000	2359.000000	615.000

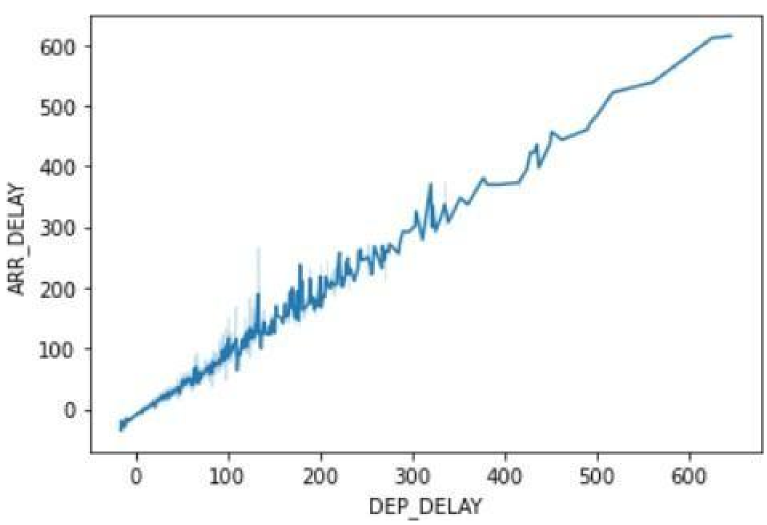
In [19]:
```
data1.describe()
```

Out[19]:

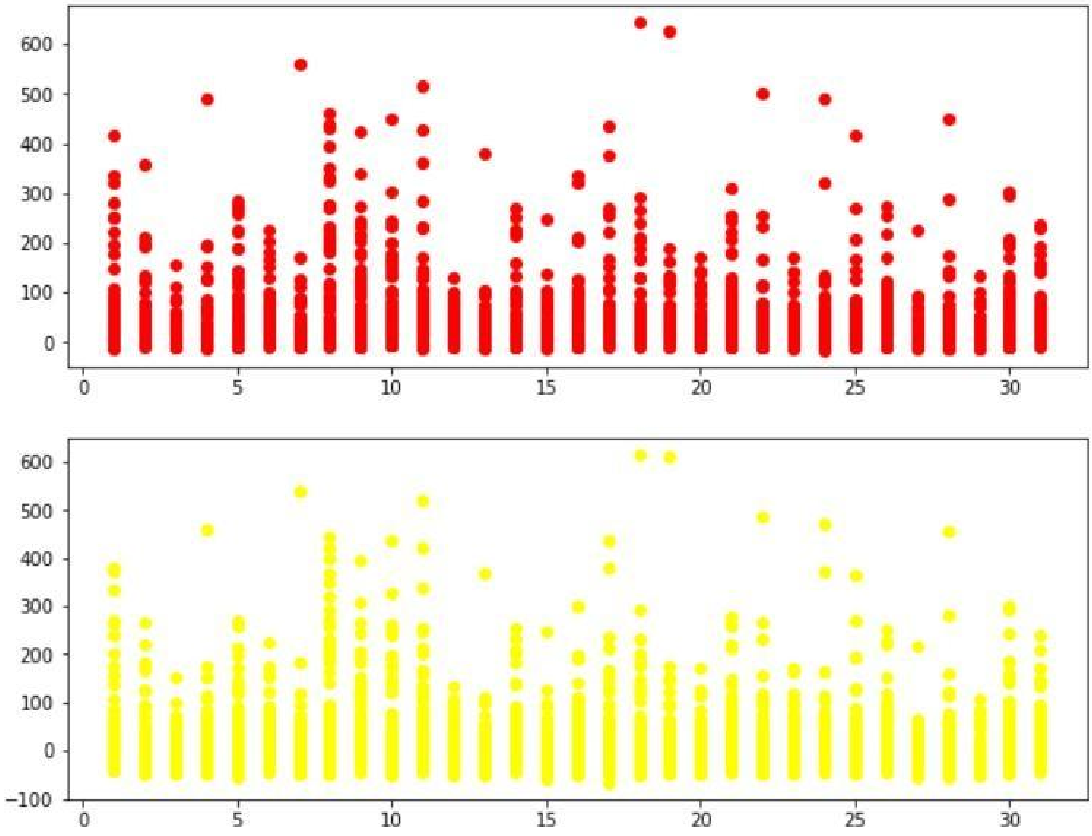
M	CRS_DEP_TIME.1	DEP_DELAY	DEP_DEL15	CRS_ARR_TIME.1	ARR_DELAY	ARR_DEL15	CANCELLED	DIVERTED	CRS_ELAPSED_TIME	DISTANCE
00	11231.000000	11124.000000	11124.000000	11231.000000	11043.000000	11043.000000	11231.000000	11231.000000	11231.000000	11231.000000
17	1320.798326	8.460266	0.142844	1537.312795	-2.573123	0.124513	0.010150	0.006589	190.652124	1161.031965
27	490.737845	36.762969	0.349930	502.512494	39.232521	0.330181	0.100241	0.080908	78.386317	643.683379
00	10.000000	-16.000000	0.000000	2.000000	-67.000000	0.000000	0.000000	0.000000	93.000000	509.000000
00	905.000000	-3.000000	0.000000	1130.000000	-19.000000	0.000000	0.000000	0.000000	127.000000	594.000000
00	1320.000000	-1.000000	0.000000	1559.000000	-10.000000	0.000000	0.000000	0.000000	159.000000	907.000000
00	1735.000000	4.000000	0.000000	1952.000000	1.000000	0.000000	0.000000	0.000000	255.000000	1927.000000
00	2359.000000	645.000000	1.000000	2359.000000	615.000000	1.000000	1.000000	1.000000	397.000000	2422.000000

Data Analysis And Visualization:

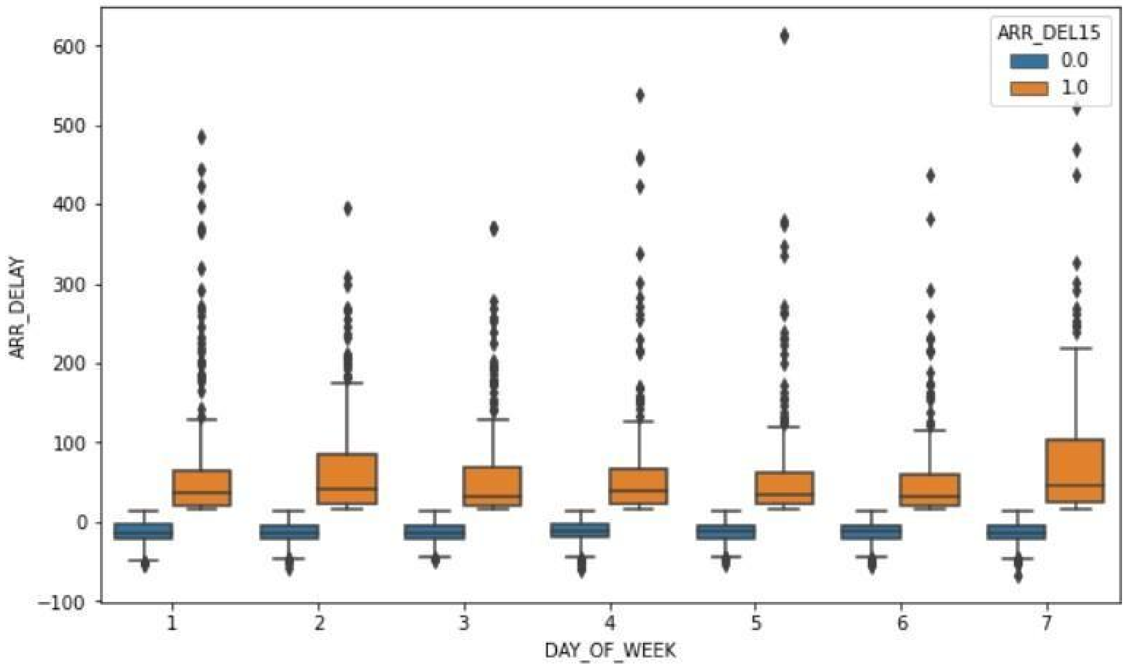
This graph shows the positive trend and strong binding between arrival and departure delay.



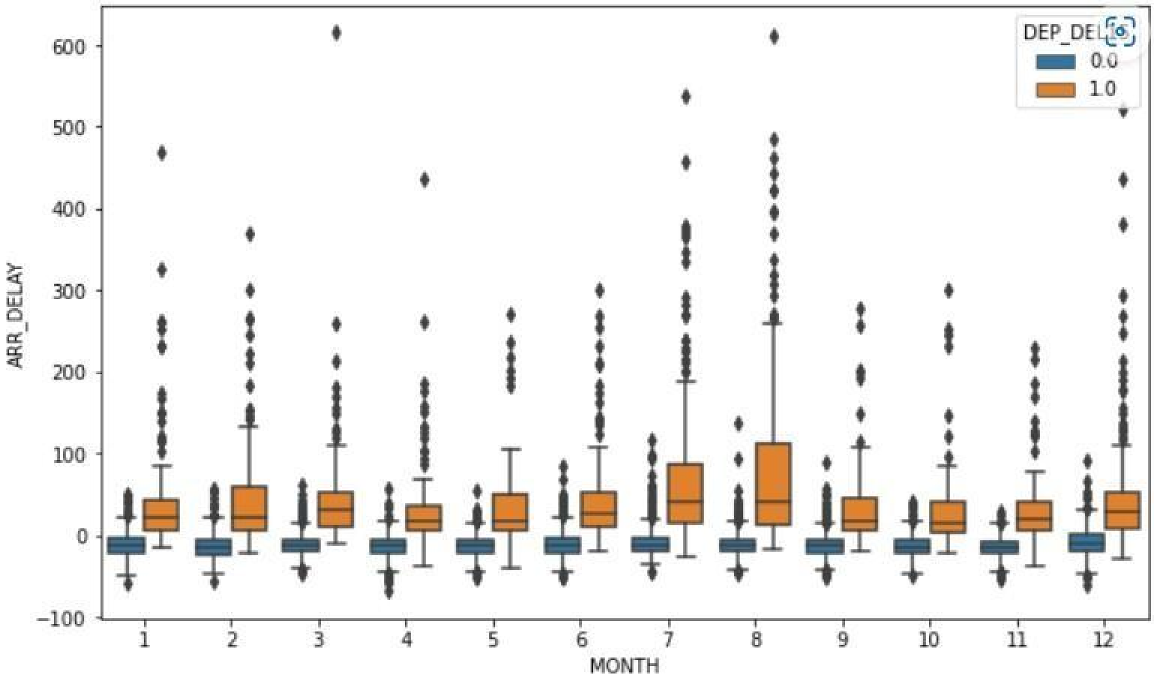
```
plt.scatter(data1["DAY_OF_MONTH"],data1["DEP_DELAY"],color="red")
plt.subplot(2,1,2)
plt.scatter(data1["DAY_OF_MONTH"],data1["ARR_DELAY"],color="yellow")
plt.show()
```



This above picture shows the relationship between day of month and delays.

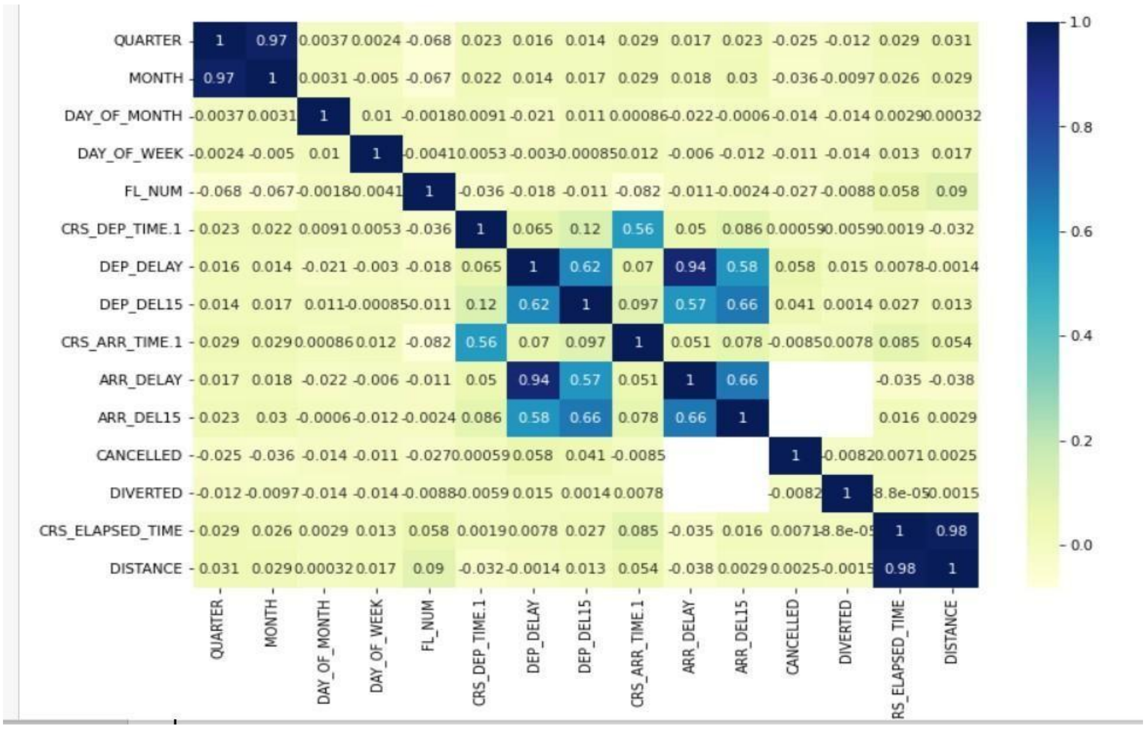


This above boxplot shows the trends of days of the week and delay, Monday and Saturday had high delays.



This above boxplot shows the seasonal relationship between months and delays. August had highest no of delays.

Correlation between columns:



Feature Engineering:

We engineered Season from the month according to the analysis

```
In [25]: data1.groupby(by="DAY_OF_WEEK")["DEP_DEL15"].sum()
Out[25]: DAY_OF_WEEK
         1     253.0
         2     213.0
         3     204.0
         4     245.0
         5     250.0
         6     198.0
         7     226.0
         Name: DEP_DEL15, dtype: float64

In [26]: data1.groupby(by="MONTH")["DEP_DEL15"].sum()
Out[26]: MONTH
         1      113.0
         2      115.0
         3      104.0
         4       96.0
         5       86.0
         6      168.0
         7      219.0
         8      246.0
         9       88.0
         10      86.0
         11      66.0
         12     202.0
         Name: DEP_DEL15, dtype: float64
```

Then Engineered NDELAY column from the summary of ARR_DEL15, DEP_DEL15, CANCELLED, DIVERTED columns.

Splitted NDELAY as dependenr column and others independent columns after removing unnecessary columns.

Data Balancing:

We balanced our using SMOTE technique which works based on KNN principle.

Balancing Dataset Using SMOTE Technique

```
In [48]: from imblearn.combine import SMOTETomek
         smote=SMOTETomek(sampling_strategy={1:2000,2:2000,3:400,4:700},random_state=42)
         x1,y2=smote.fit_resample(x,y)
         y2.value_counts()

Out[48]: 0.0    8316
         1.0    1537
         2.0    1493
         4.0     634
         3.0     340
         Name: NDELAY, dtype: int64
```

Encoding Categorical columns into numerical columns:

We encoded ORGIN ,DEST into numerical columns.

Model Buliding:

We builded

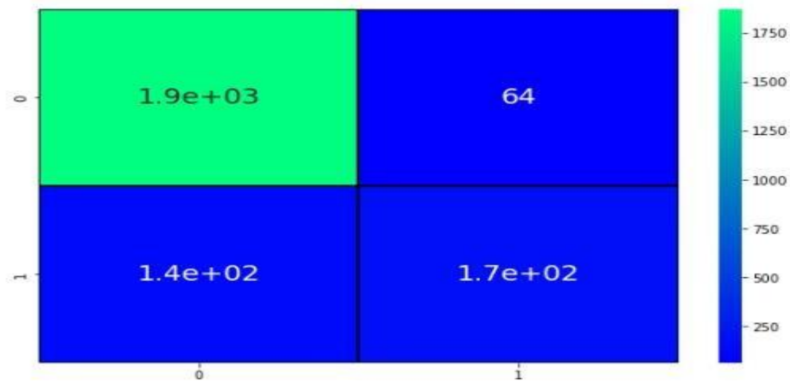
```
Decision Tree with 0.7536525974025974
Random Forest with 0.8368506493506493
SVM with 0.6128246753246753
KNN with 0.7280844155844156
Logistic Regession with 0.6830357142857143
```

We will explore only Random Forest and Decision Tree which have high accuracy

Random Forest:

```
#print(round(accuracy_score(prediction3,y_test)*100,2))
#print('Testing Accuracy for knn',(TP+TN)/(TP+TN+FN+FP))
print('Testing Sensitivity for Random Forest',(TP/(TP+FN)))
print('Testing Specificity for Random Forest',(TN/(TN+FP)))
print('Testing Precision for Random Forest',(TP/(TP+FP)))
print('Testing accuracy for Random Forest',accuracy_score(y_test, pred))
```

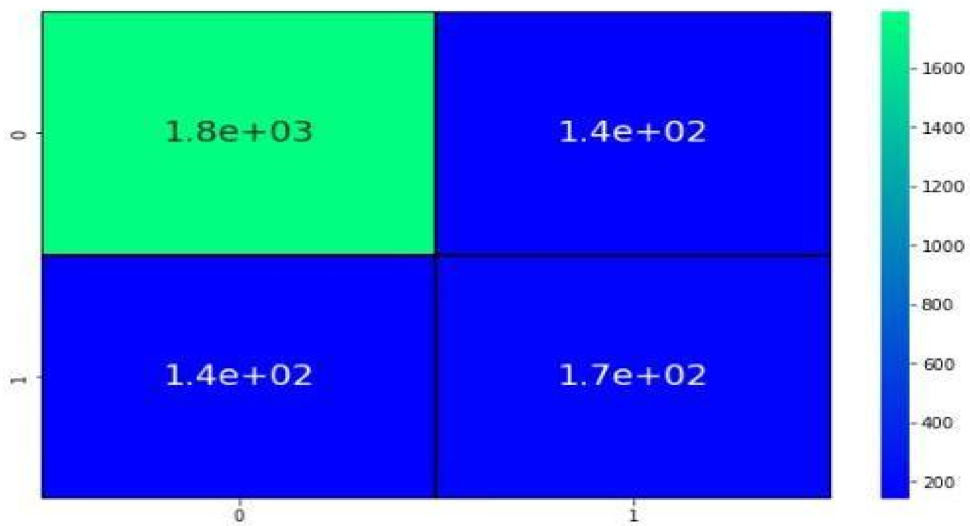
```
Testing Sensitivity for Random Forest 0.9313432835820895
Testing Specificity for Random Forest 0.729957805907173
Testing Precision for Random Forest 0.9669421487603306
Testing accuracy for Random Forest 0.9101023587004895
```



Decision Tree:

```
print('Testing Precision for Decision Tree',(TP/(TP+FP)))
print('Testing accuracy for Decision Tree',accuracy_score(y_test, pred1))
```

```
Testing Accuracy for Decision Tree 0.872719181130396
Testing Sensitivity for Decision Tree 0.9265770423991727
Testing Specificity for Decision Tree 0.5399361022364217
Testing Precision for Decision Tree 0.9256198347107438
Testing accuracy for Decision Tree 0.872719181130396
```



Model Saving:

Random Forest gives the best accuracy then others , so we save random forest model using pickle.

```
In [71]: import pickle

In [72]: pickle.dump(rf,open("rfmodel.pkl",'wb'))
```

Conclusion:

In this sprint , we builded our model , evaluated and saved. In next sprint, we deploy our model IBM cloud using IBM Watson and building Dashboard.

DEVELOPMENT PHASE:

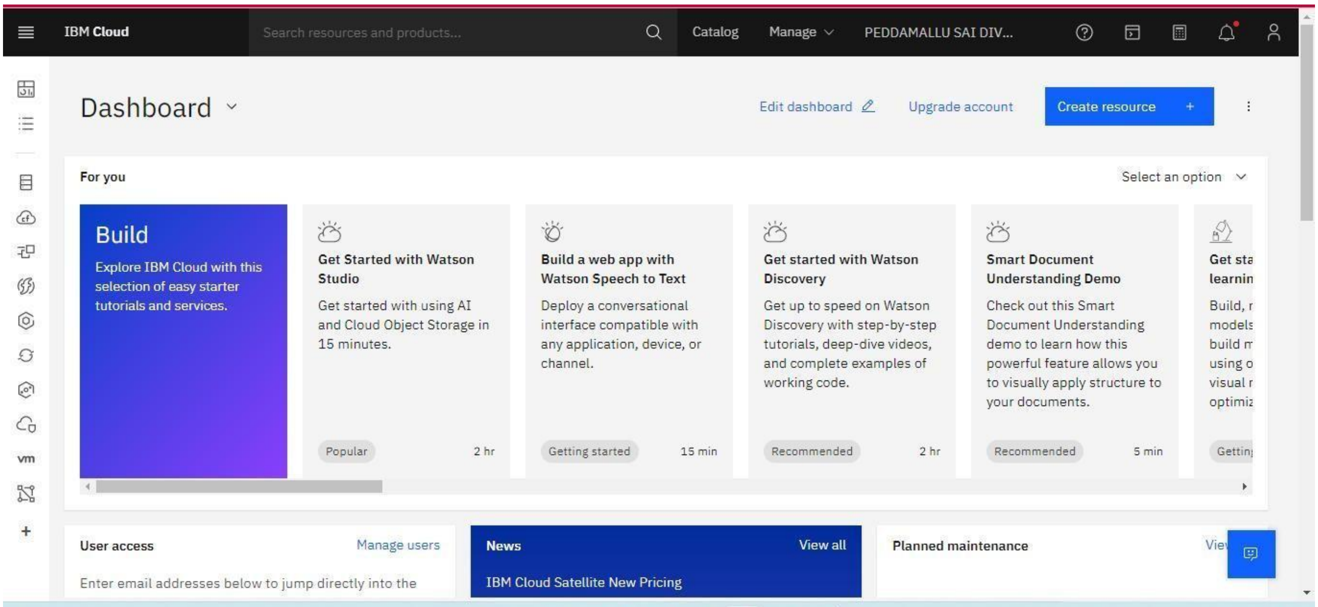
- Creating IBM cloud account & Required Resources
- Importing jupyter notebook file in ibm watson
- Deploy our model in IBM Watson
- Predict the result

Creating IBM cloud account & Required Resources:

Creating IBM cloud account:

Frist, need to create IBM Cloud account by using SI Mail Id and SI Password which is provided by IBM in profile.

Below dashboard of an account after created,



Creating IBM Cloud Required Resources:

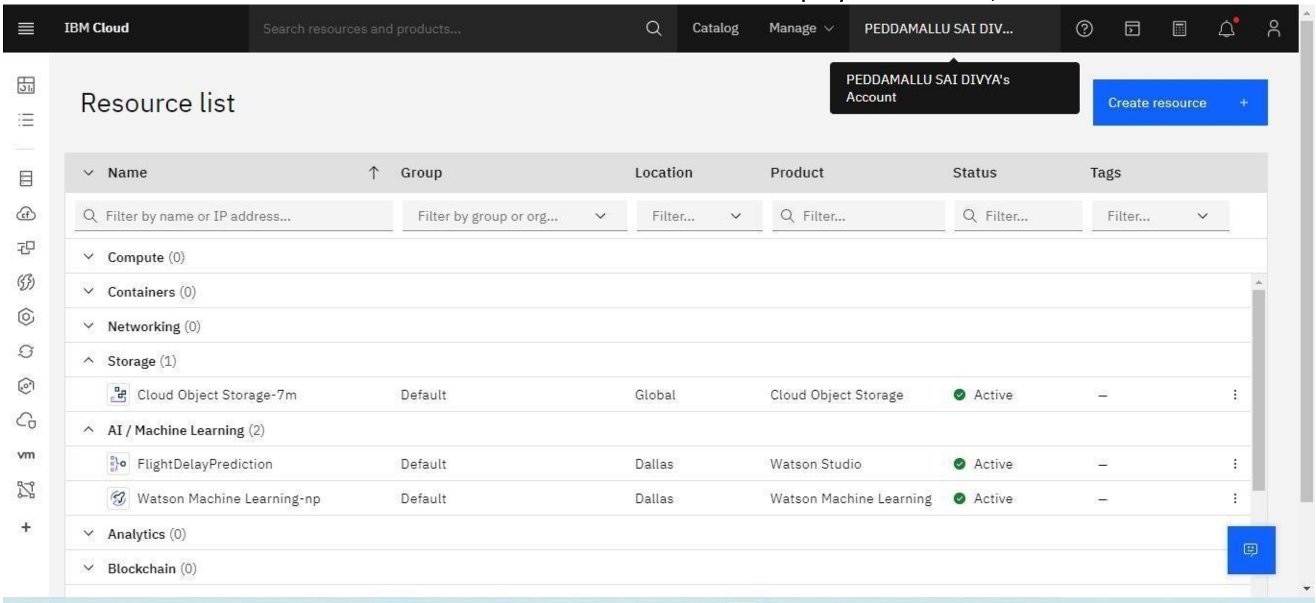
After creating IBM cloud account, to deploy ML model, need to create following resources such as,

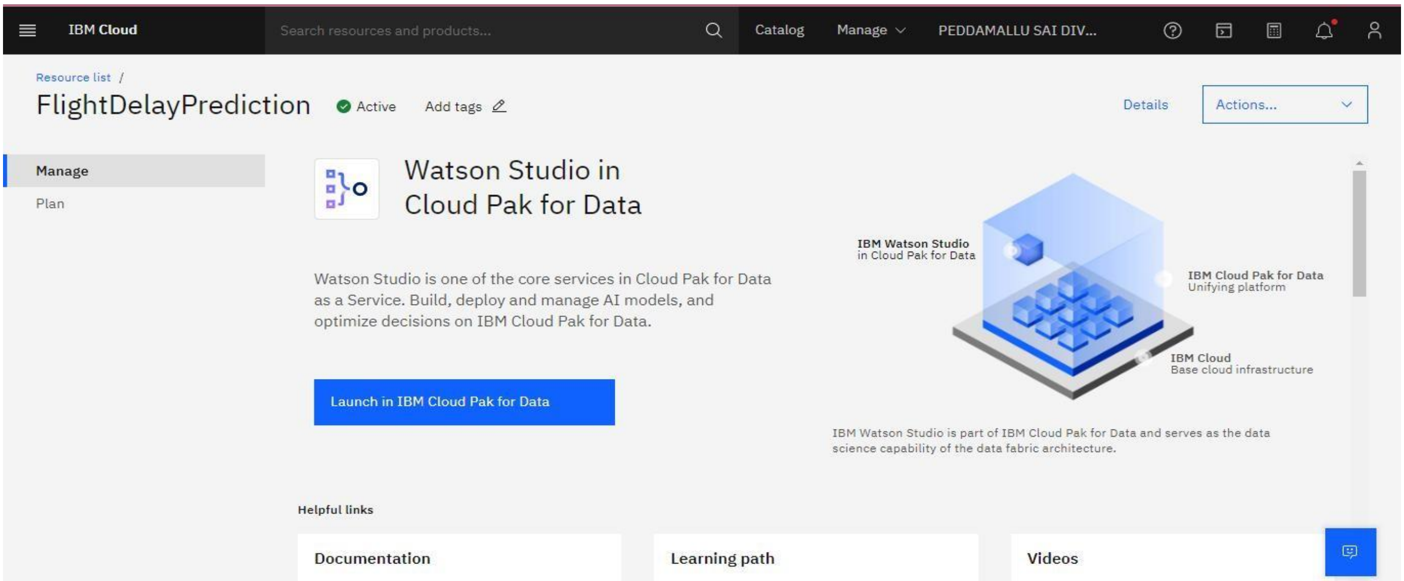
Cloud Object Storage

Watson Machine Learning

Watson Studio

After created above resources Resource List of an account is displayed as follow,

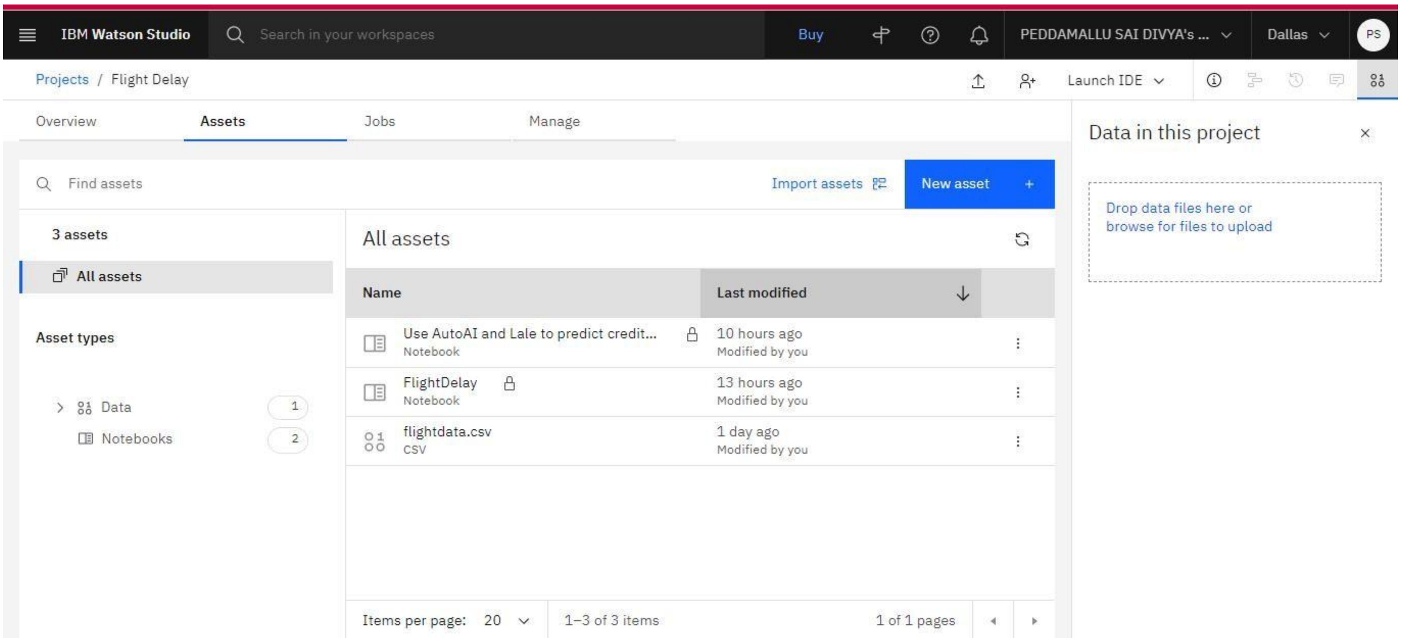




All the resource are in active state.

All the required cloud resources are created successfully.

Import .ipynb file of sprint-1 which ML models are build in Jupyter notebook.



Import Required Libraries

The first step is usually importing the libraries that will be needed in the program.

IBM Watson Studio

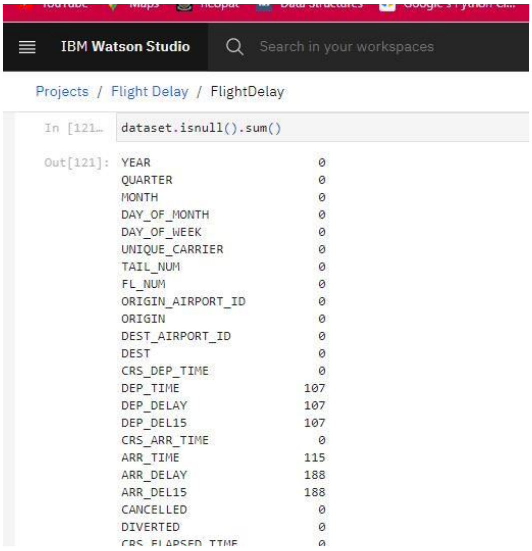
Projects / Flight Delay / FlightDelay

```
In [84]: #!pip install imbalanced-learn
         !pip install imblearn

Requirement already satisfied: imblearn in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (0.0)
Requirement already satisfied: imbalanced-learn in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from imblearn) (0.9.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from imbalanced-learn->imblearn) (2.2.0)
Requirement already satisfied: joblib>=1.0.0 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from imbalanced-learn->imblearn) (1.2.0)
Requirement already satisfied: scipy>=1.3.2 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from imbalanced-learn->imblearn) (1.7.3)
Requirement already satisfied: numpy>=1.17.3 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from imbalanced-learn->imblearn) (1.20.3)
Requirement already satisfied: scikit-learn>=1.1.0 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from imbalanced-learn->imblearn) (1.1.3)

In [116…
import sys
import numpy as np
import pandas as pd
import seaborn as sns
import pickle
import sklearn
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import sklearn.metrics as metrics
from matplotlib import pyplot as plt
import imblearn
```

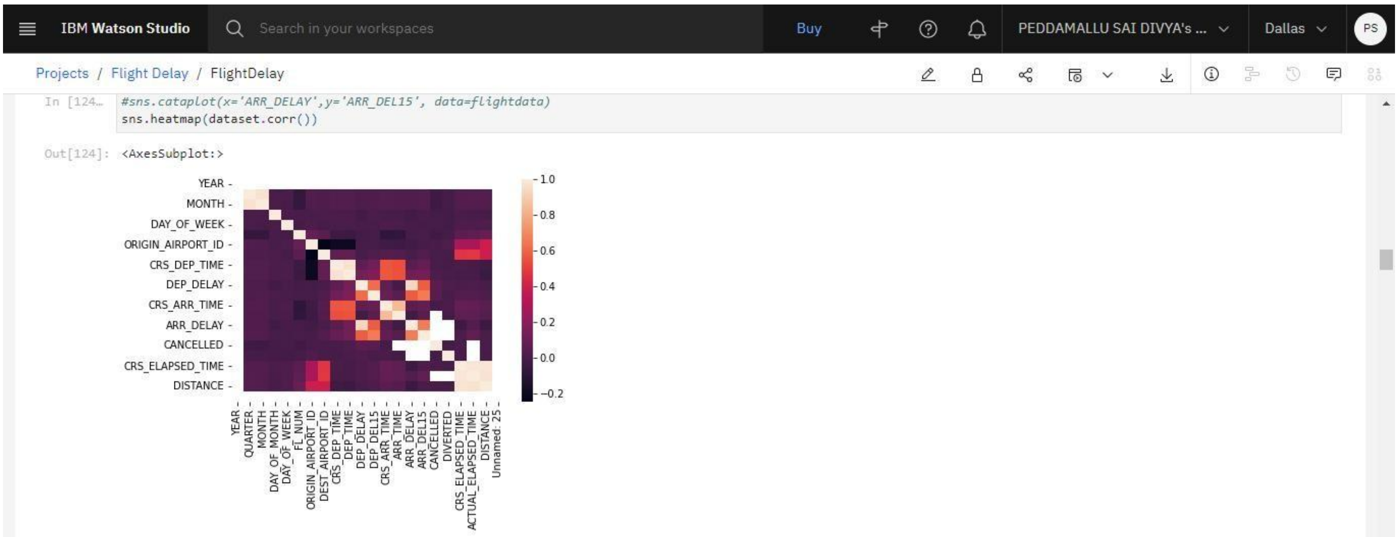
Importing The Dataset and Analyze The Data set

```
In [119… 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
```

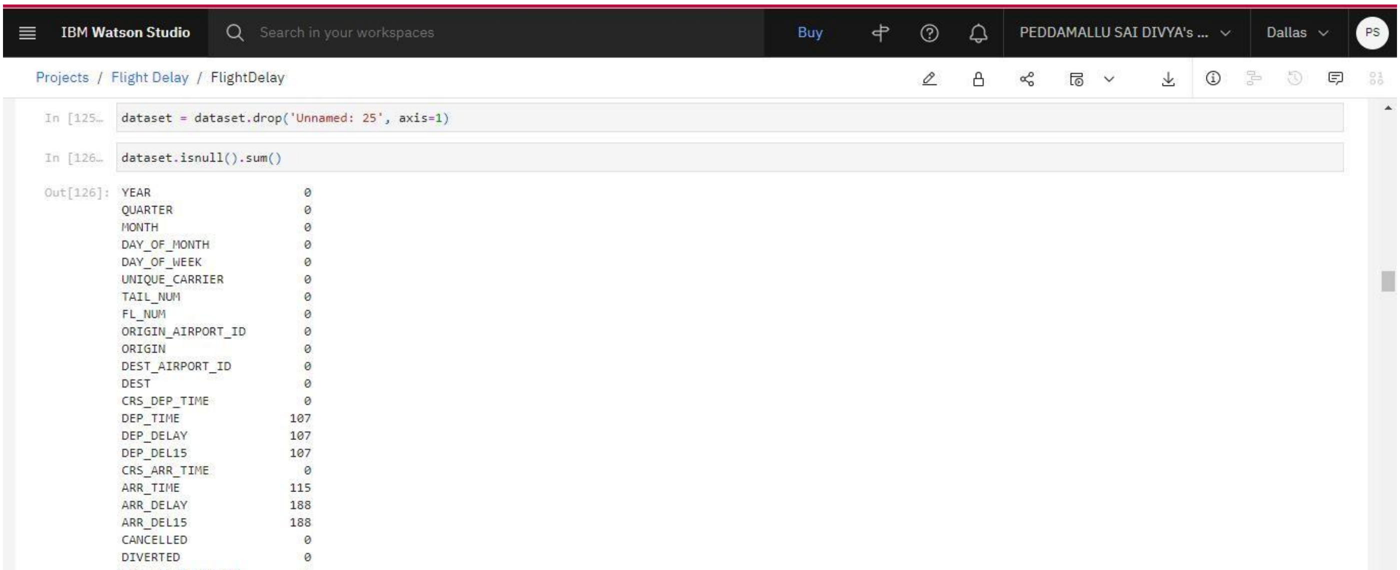
Handling Missing Values



Data Visualization



Dropping Un-Necessary Columns



Label Encoding & One Hot Encoding

IBM Watson Studio | Search in your workspaces | Buy | PEDDAMALLU SAI DIVYA's ... | Dallas | PS

Projects / Flight Delay / FlightDelay

```
dtype: int64
```

In [128]:
```
dataset = dataset.fillna({'ARR_DEL15':1})
dataset = dataset.fillna({'DEP_DEL15':0})
dataset.iloc[177:185]
```

Out[128]:

	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

In [129]:
```
import math

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

Splitting The Dataset Into Dependent And Independent Variables

IBM Watson Studio | Search in your workspaces | Buy | PEDDAMALLU SAI DIVYA's ...

Projects / Flight Delay / FlightDelay

In [174]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [175]:
```
!pip install imblearn
```

```
Requirement already satisfied: imblearn in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (0.0)
Requirement already satisfied: imbalanced-learn in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from imblearn) (0.9.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from imbalanced-learn->imblearn) (2.2.0)
Requirement already satisfied: scikit-learn>=1.1.0 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from imbalanced-learn->imblearn) (1.1.3)
Requirement already satisfied: scipy>=1.3.2 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from imbalanced-learn->imblearn) (1.7.3)
Requirement already satisfied: joblib>=1.0.0 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from imbalanced-learn->imblearn) (1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from imbalanced-learn->imblearn) (1.20.3)
```

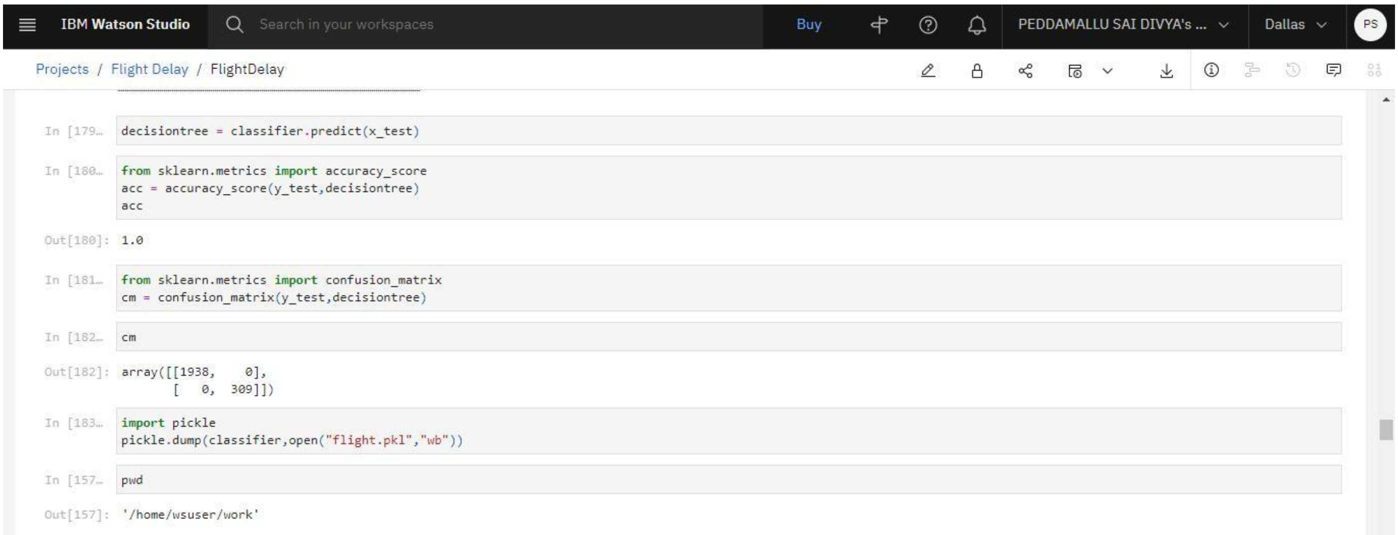
In [176]:
```
from imblearn.over_sampling import SMOTE
smote = SMOTE()
```

In [177]:
```
x_train_smote, y_train_smote = smote.fit_resample(x_train, y_train)
```

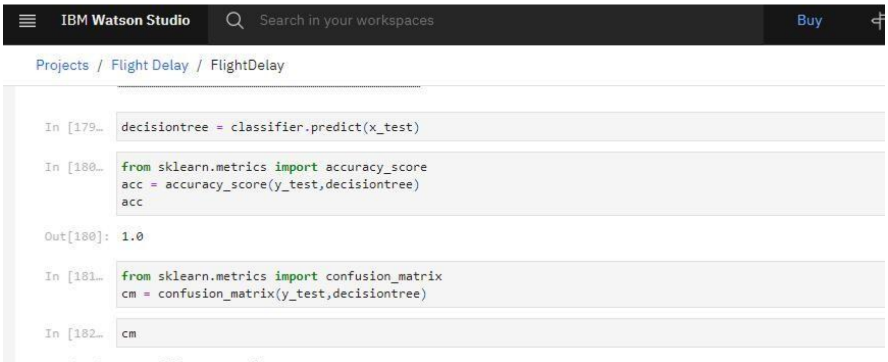
In [178]:
```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(random_state = 0)
classifier.fit(x_train_smote,y_train_smote)
```

Out[178]:
```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=0)
```

Splitting The Dataset Into Dependent And Independent Variables



Train And Test The Model Using Decision Tree Classifier



Model Evaluation

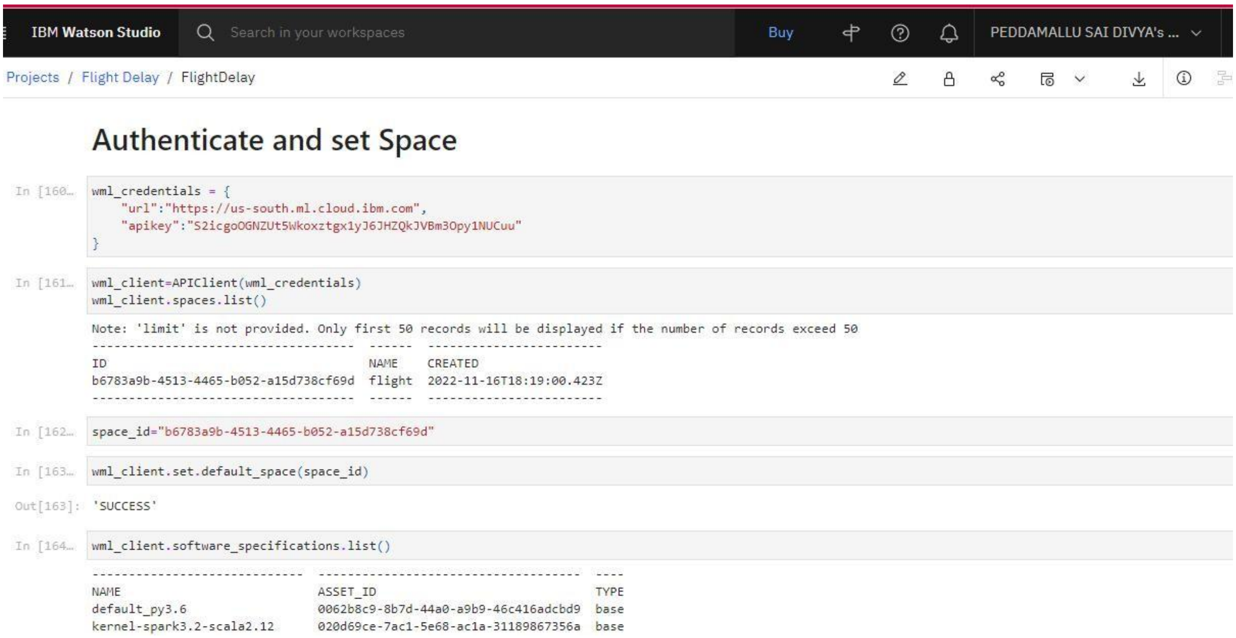
Deploy our model in IBM Watson:

To deploy ML model in IBM cloud, need to create project in IBM Watson. After successful creation of project import .ipynb file of sprint-1 which ML models are build in Jupyter notebook.

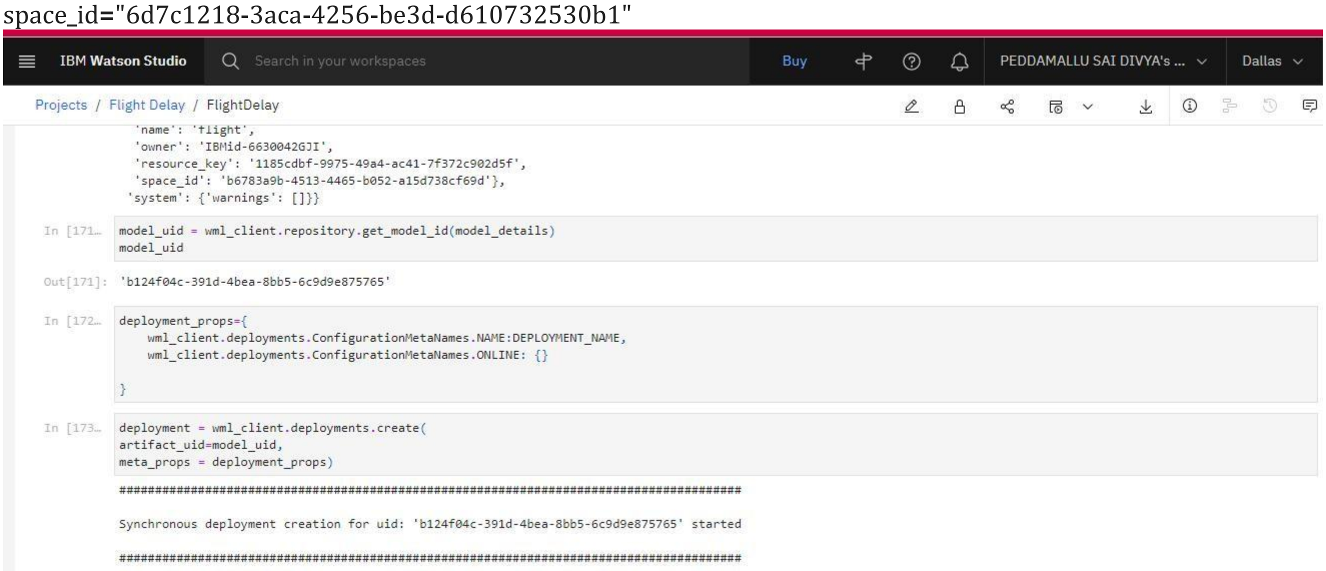
Upload required datasets and import it.

Deploy model using following code,

!pip install -U ibm-watson-machine-learning



from ibm_watson_machine_learning **import** APIClient
import json
import numpy **as** np
wml_cred={
"apikey":"okbr7ARnOQjyplTOyvNFC2QVkCF6q7afpci065Hucby8",
"url":"https://us-south.ml.cloud.ibm.com"
}
wml_clients=APIClient(wml_cred)
wml_clients.spaces.list()
space_id="6d7c1218-3aca-4256-be3d-d610732530b1"



```
wml_clients.set.default_space(space_id)
wml_clients.software_specifications.list(500)
MODEL_NAME="randomforest"
DEPLOYMENT_NAME="rf_deployment"
DEMO_MODEL=rf
soft_sepc_id=wml_clients.software_specifications.get_id_by_name("runtime-22.1-py3.9")
```

In [115]:

```
model_props={ wml_clients.repository.ModelMetaNames.NAME:MODEL_NAME,
  wml_clients.repository.ModelMetaNames.TYPE:"scikit-learn_1.0",
  wml_clients.repository.ModelMetaNames.SOFTWARE_SPEC_UID: soft_sepc_id

}
```

In [116]:

```
model_details=wml_clients.repository.store_model(model=DEMO_MODEL,meta_props=model_props,trai
ning_data=x_train,
                    training_target=y_train.values.ravel())
```

In [117]:

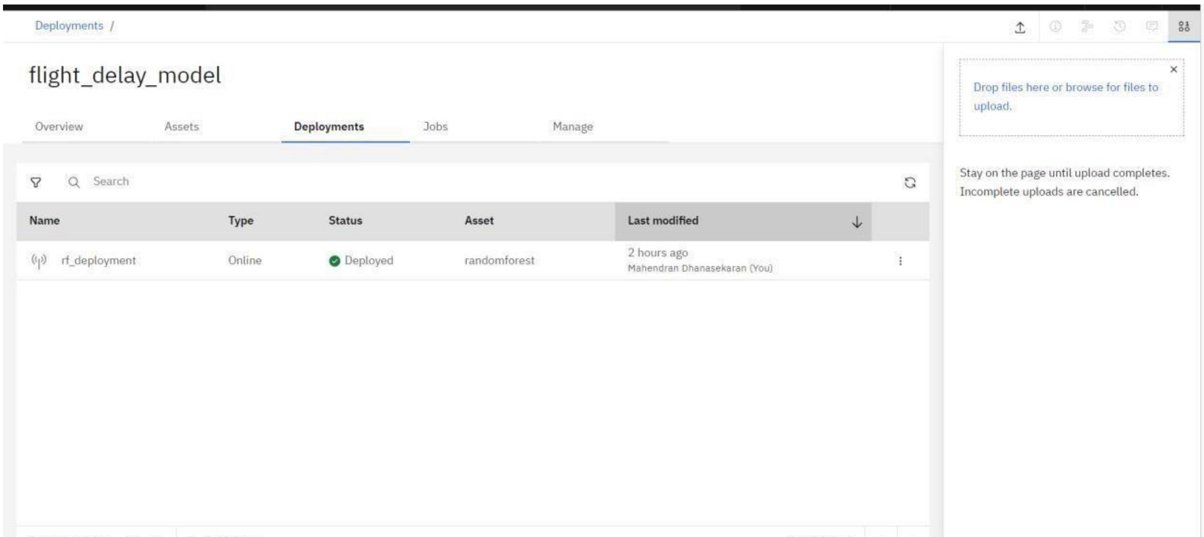
```
model_details
model_id=wml_clients.repository.get_model_id(model_details)
dep_props={
  wml_clients.deployments.ConfigurationMetaNames.NAME:DEPLOYMENT_NAME,
  wml_clients.deployments.ConfigurationMetaNames.ONLINE:{}
}
```

In [125]:

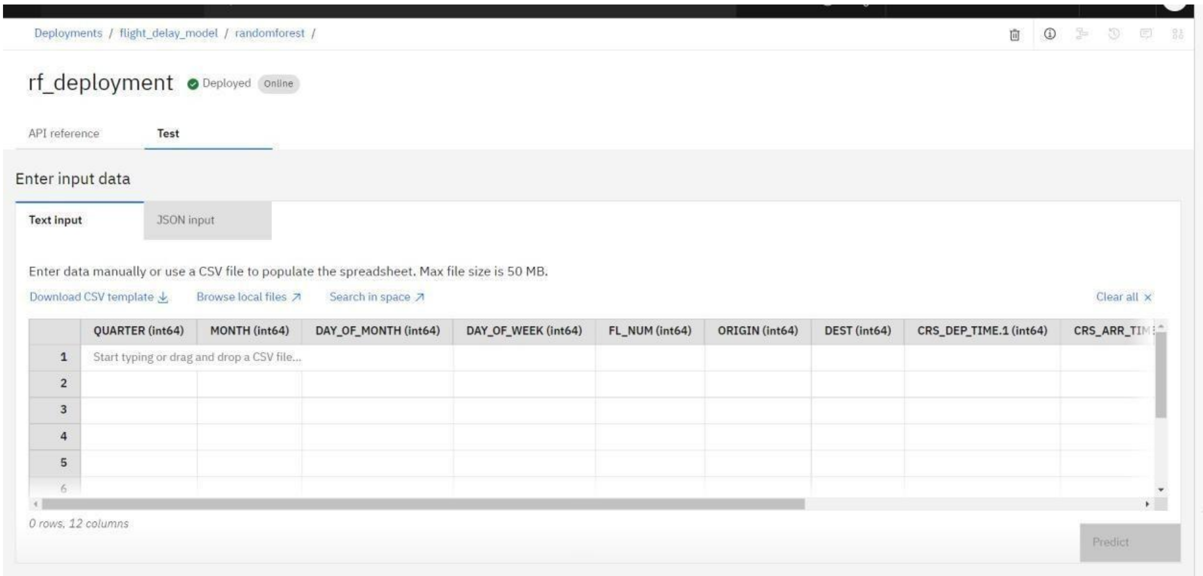
```
deployment=wml_clients.deployments.create(artifact_uid=model_id,meta_props=dep_props)
```

NOTE: APIKey must need to create to deploy and connect API

After successful of deployment, deployed is appeared in Deployment section as follow,



Testing of deployed model as follow, by giving values of all the features and it gives prediction.



Output is predicted by ML model successfully.

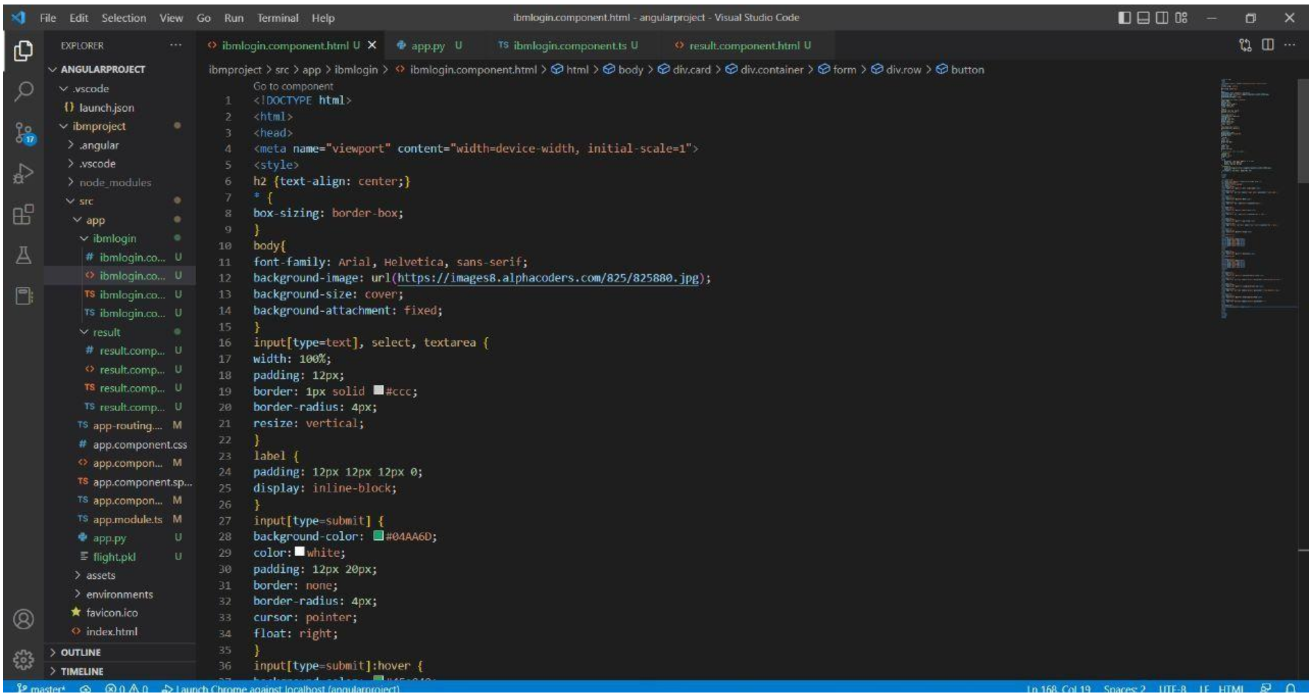
DEVELOPMENT PHASE:

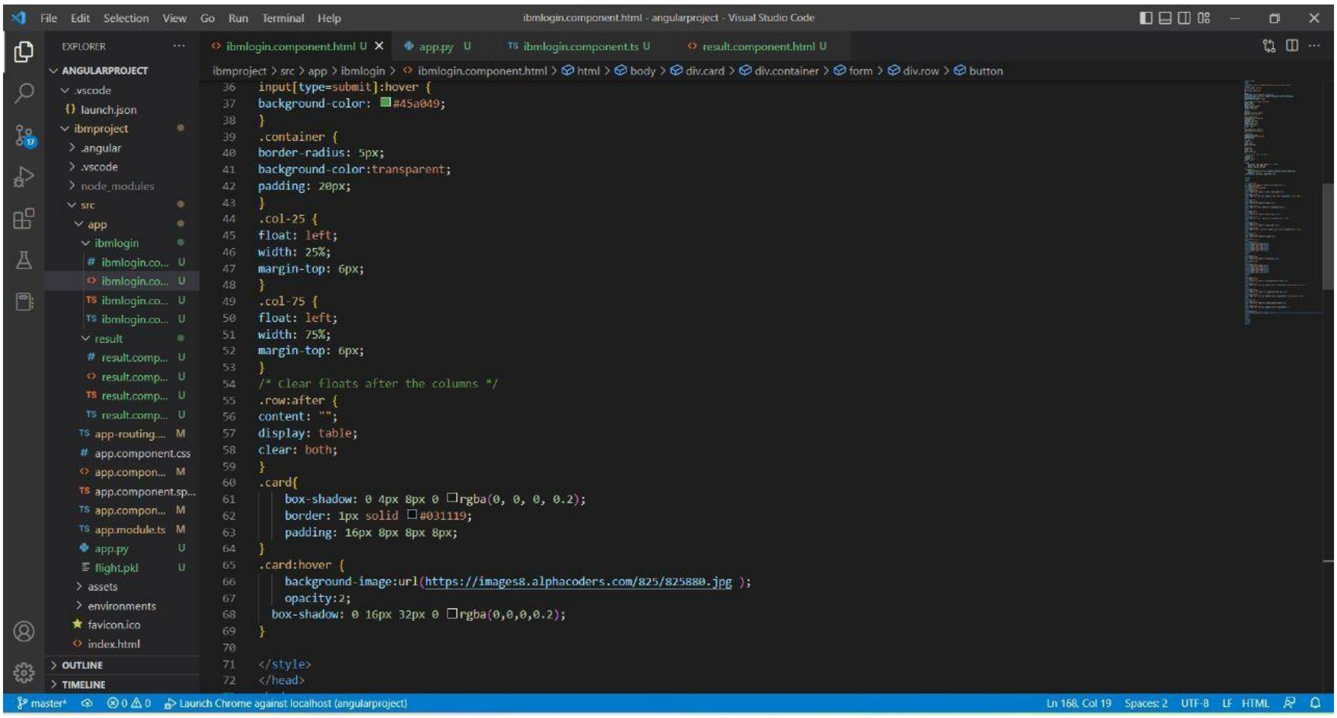
- Importing source code from IBM Watson
- Creating HTML Pages
- Creating Dashboard using HTML/CSS
- Create web app and Hosting in falsk
- Testing web app

Creating Dashboard using HTML/CSS:

Frontend Dashboard is created using HTML/CSS,

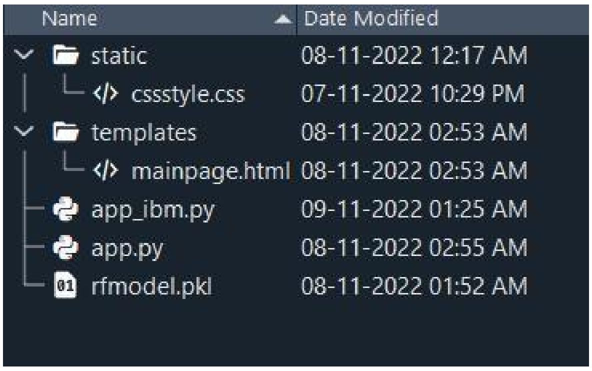
Result as web page like,





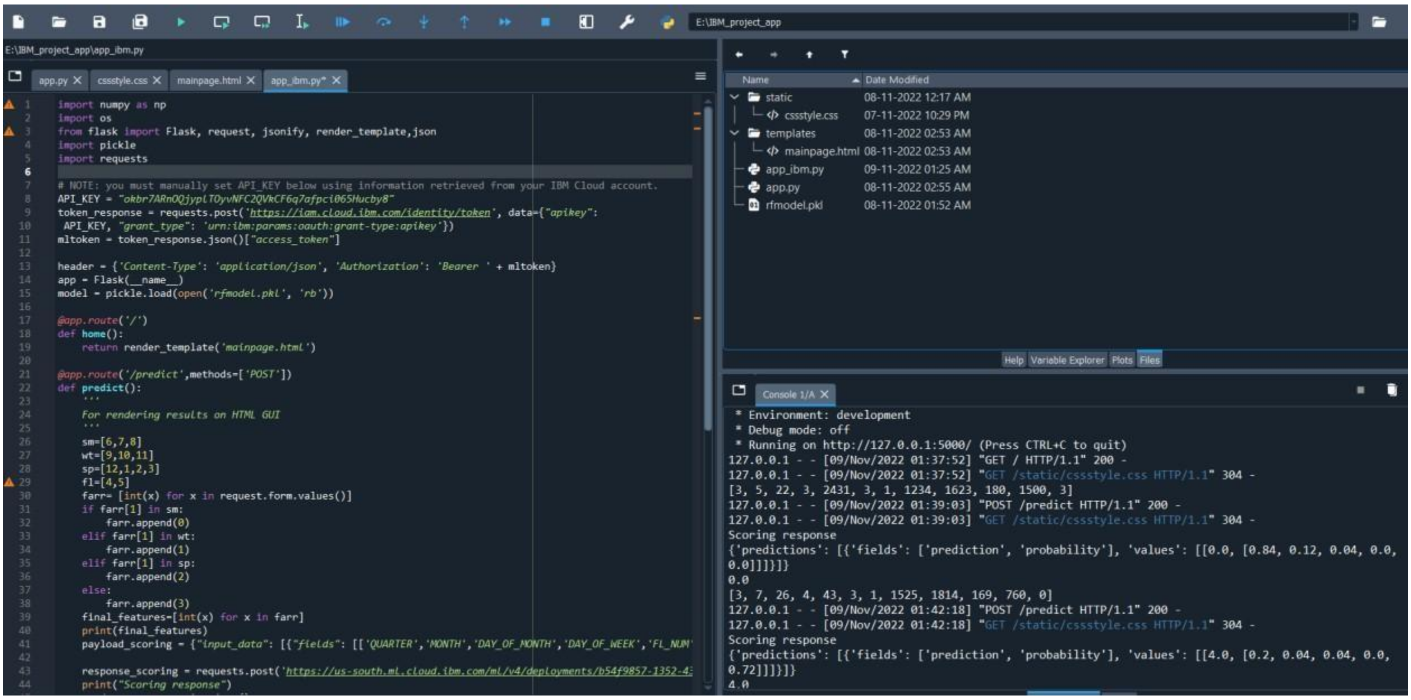
Create web app and Hosting in falsk:

First thing, need to create directory as follow,



Then, code the required logic in app.py file with API connection , request and response code.

Spyder IDE looks like,



Run the app.py file.

Localhost url is displayed in console, copy and paste in browser then search it , frond end HTML?CSS page is displayed. Successfully created and hosted web app in flask.

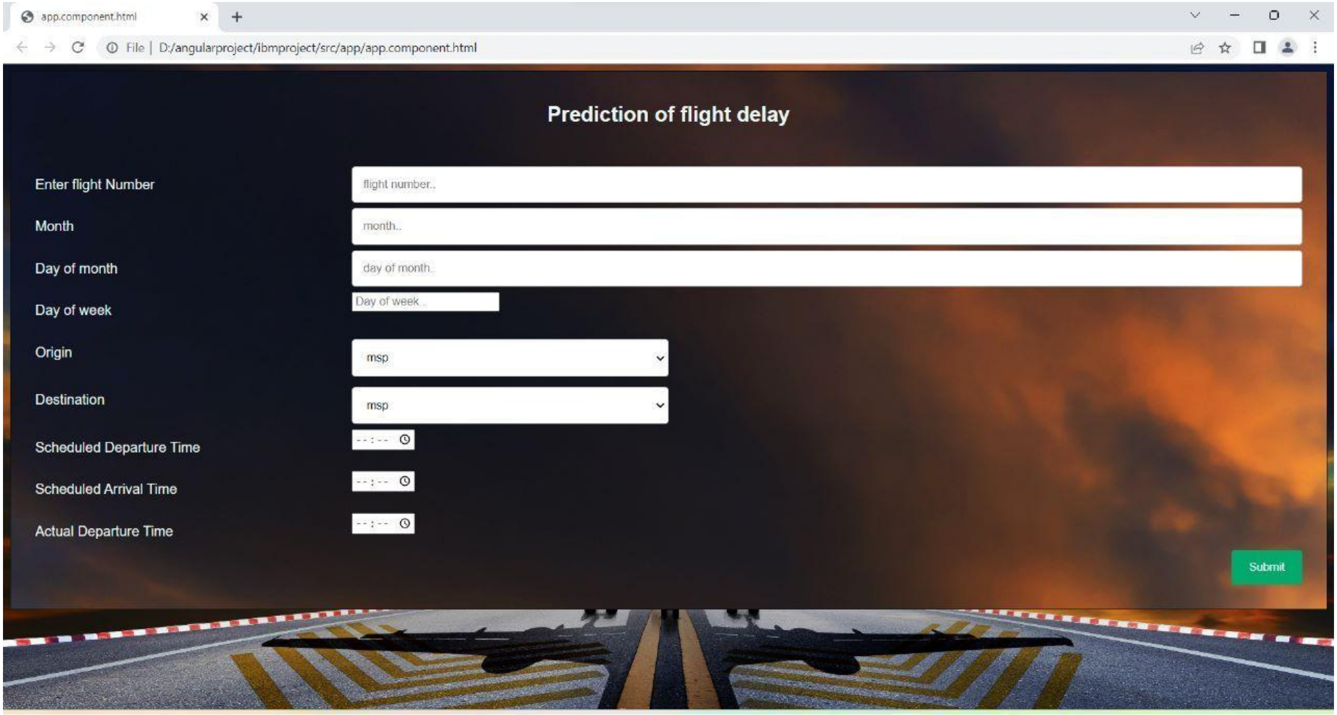
If any error caused as flask in production mode, then

Set FLASK_ENV=Development,

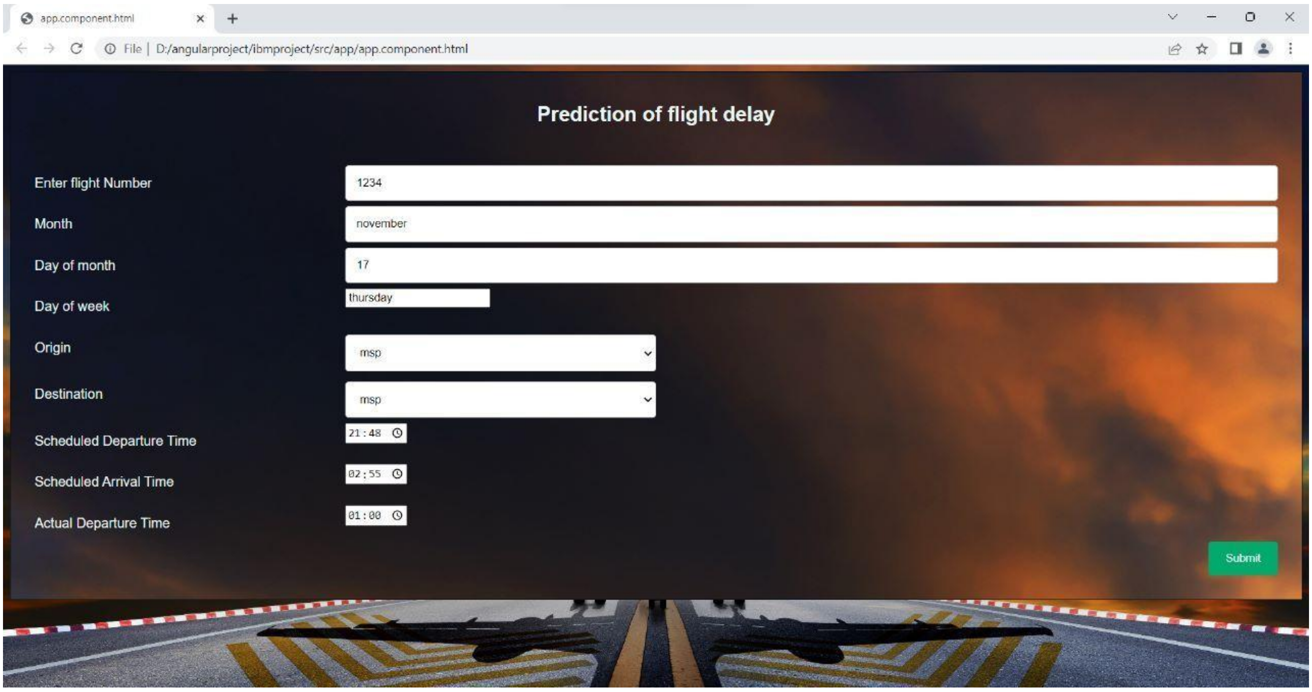
Then run the app

Testing web app:

Enter the data on the required fields,



Testing the web app while entering the values



Output is Predicted By ML Model Successful