SPRINT – 1 PROJECT DOCUMENT

Date	28 October 2022
Team ID	PNT2022TMID26438
Project Name	Flight Delay Prediction Using Machine Learning

DEVELOPMENT PHASE:

SPRINT-1:

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
- Imblearn - Balancing Data
- 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:

The dataset contains 31 variables with various data types such as string, object, time, integer, float.

```
Data columns (total 31 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   object
 13   CRS_DEP_TIME.1         11231 non-null   int64
 14   DEP_TIME               11124 non-null   object
 15   DEP_TIME.1             11124 non-null   float64
 16   DEP_DELAY              11124 non-null   float64
 17   DEP_DEL15              11124 non-null   float64
 18   CRS_ARR_TIME           11231 non-null   object
 19   CRS_ARR_TIME.1         11231 non-null   int64
 20   ARR_TIME               11116 non-null   object
 21   ARR_TIME.1             11116 non-null   float64
 22   ARR_DELAY              11043 non-null   float64
 23   ARR_DEL15              11043 non-null   float64
 24   CANCELLED              11231 non-null   int64
 25   DIVERTED               11231 non-null   int64
 26   CRS_ELAPSED_TIME1      11231 non-null   object
 27   ACTUAL_ELAPSED_TIME1  11231 non-null   object
 28   CRS_ELAPSED_TIME       11231 non-null   int64
 29   ACTUAL_ELAPSED_TIME    11043 non-null   float64
 30   DISTANCE               11231 non-null int64
dtypes: float64(7), int64(14), object(
```

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	11043.0000
mean	2.544475	6.628973	15.790758	3.960199	1334.325617	1320.798326	8.460266	0.142844	1537.312795	-2.5731
std	1.090701	3.354678	8.782056	1.995257	811.875227	490.737845	36.762969	0.349930	502.512494	39.2325
min	1.000000	1.000000	1.000000	1.000000	7.000000	10.000000	-16.000000	0.000000	2.000000	-67.0000
25%	2.000000	4.000000	8.000000	2.000000	624.000000	905.000000	-3.000000	0.000000	1130.000000	-19.0000
50%	3.000000	7.000000	16.000000	4.000000	1267.000000	1320.000000	-1.000000	0.000000	1559.000000	-10.0000
75%	3.000000	9.000000	23.000000	6.000000	2032.000000	1735.000000	4.000000	0.000000	1952.000000	1.0000
max	4.000000	12.000000	31.000000	7.000000	2853.000000	2359.000000	645.000000	1.000000	2359.000000	615.0000

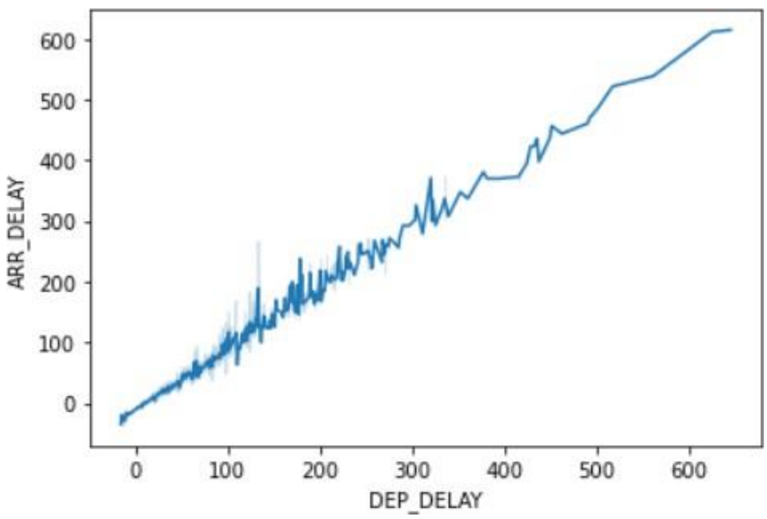
In [19]:
```
data1.describe()
```

Out[19]:

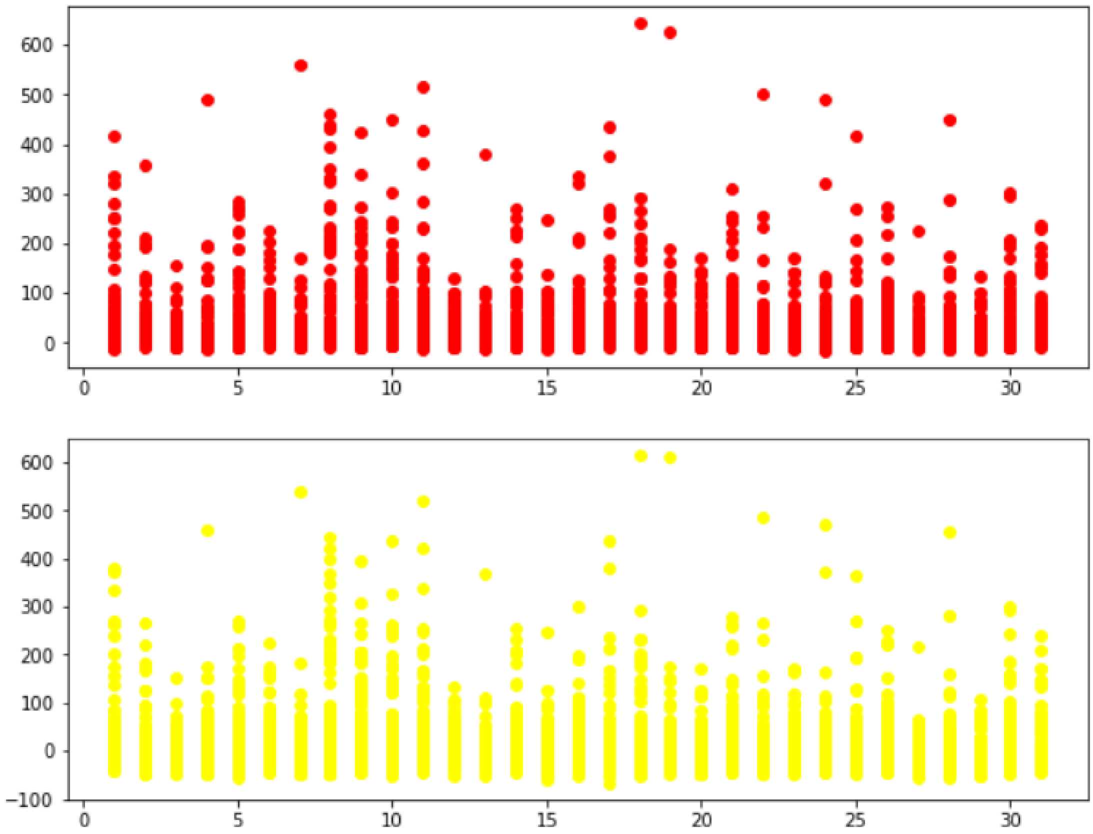
M	CRS_DEP_TIME.1	DEP_DELAY	DEP_DEL15	CRS_ARR_TIME.1	ARR_DELAY	ARR_DEL15	CANCELLED	DIVERTED	CRS_ELAPSED_TIME	DISTANCE
00	11231.000000	11124.000000	11124.000000	11231.000000	11043.000000	11043.000000	11231.000000	11231.000000	11231.000000	11231.000000
17	1320.798326	8.460266	0.142844	1537.312795	-2.573123	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:

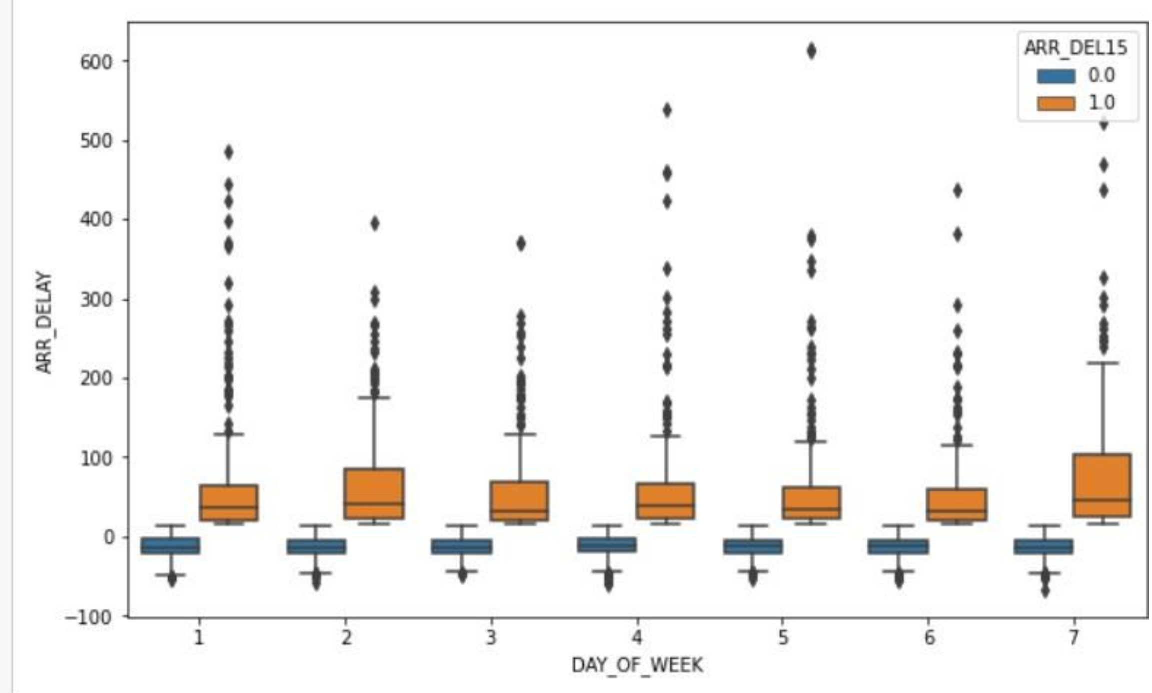
This graph shows the positive trend and strong binding between arrival and departure delay.



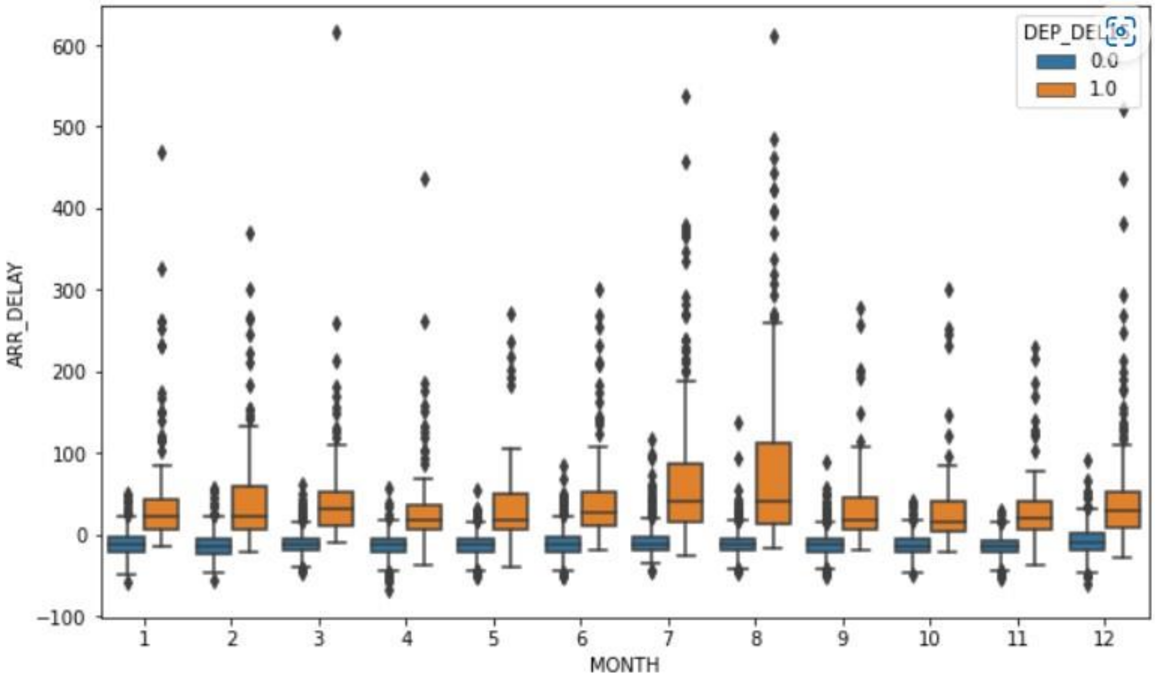
```
plt.scatter(data1["DAY_OF_MONTH"],data1["DEP_DELAY"],color="red")
plt.subplot(2,1,2)
plt.scatter(data1["DAY_OF_MONTH"],data1["ARR_DELAY"],color="yellow")
plt.show()
```



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

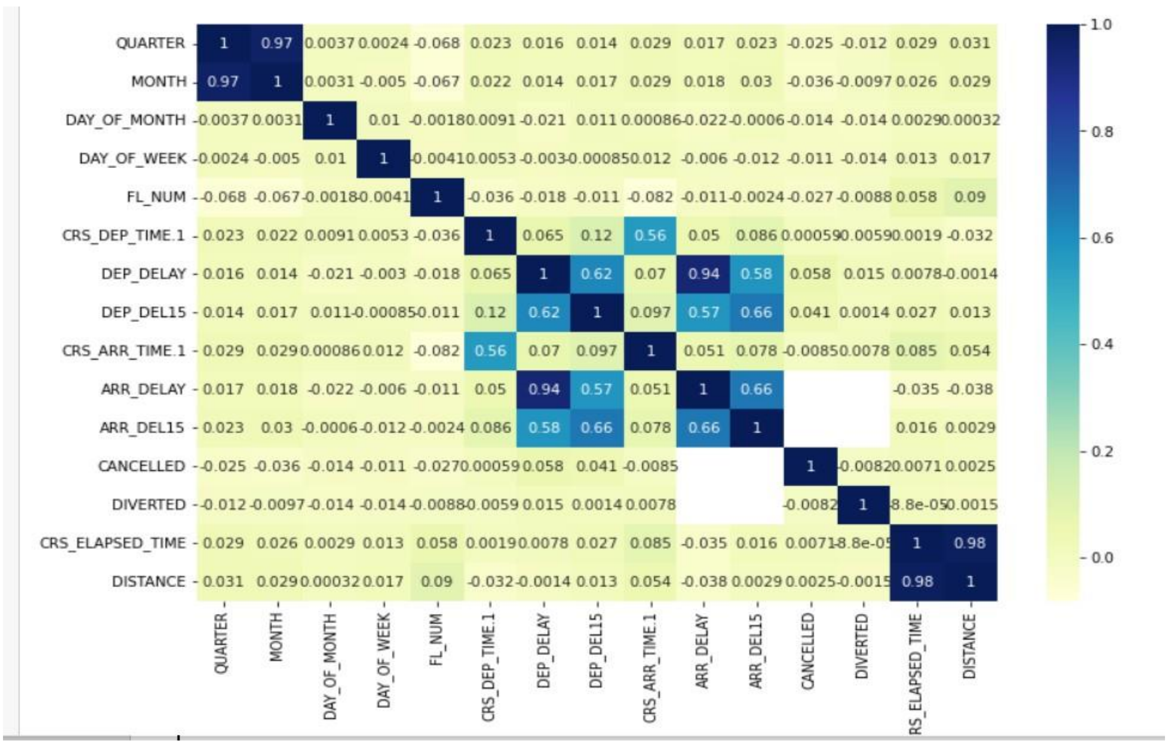


This above boxplot shows the trends of days of the week and delay, Monday and Saturday had high delays.



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:

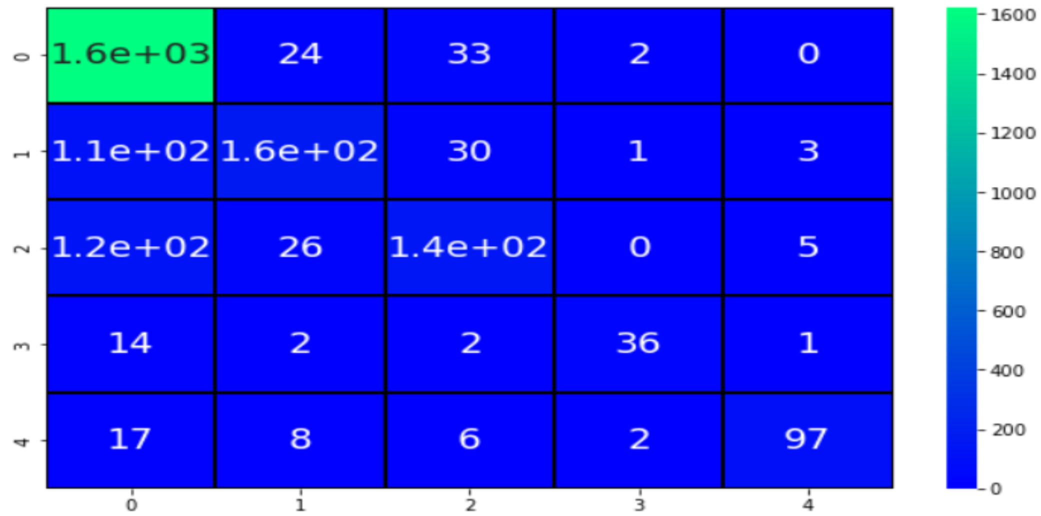
We builded

```
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

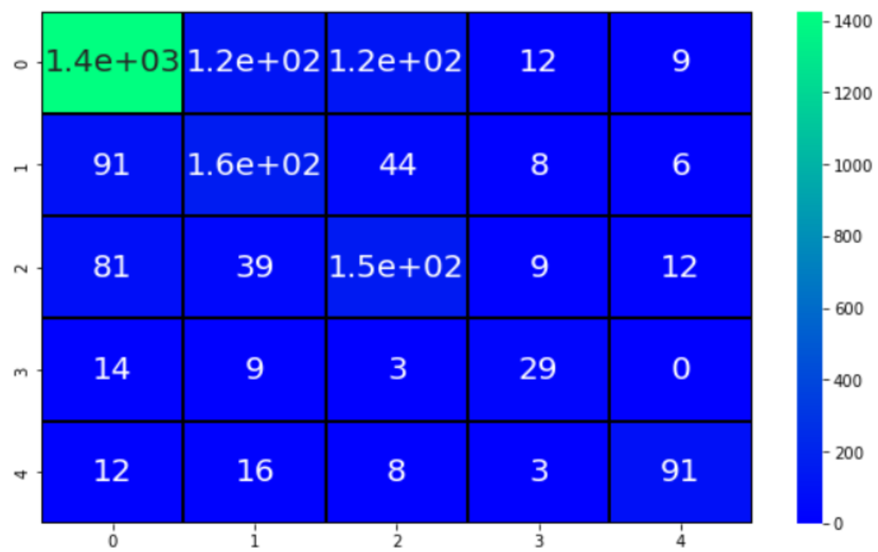
Random Forest:

```
Testing Sensitivity for Random Forest 0.9360230547550432
Testing Specificity for Random Forest 0.8716577540106952
Testing Precision for Random Forest 0.9854368932038835
Testing accuracy for Random Forest 0.8368506493506493
```



Decision Tree:

```
Testing Accuracy for Decision Tree 0.8849804578447794
Testing Sensitivity for Decision Tree 0.9400131839156229
Testing Specificity for Decision Tree 0.5802919708029197
Testing Precision for Decision Tree 0.9253731343283582
Testing accuracy for Decision Tree 0.7516233766233766
```



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.