Importing the required packages

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, precision_recal
import joblib
import pickle
```

Loading the dataset

In [2]:
```
df = pd.read_csv('flightdata.csv')
df.head()
```

Out[2]:

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	UNIQUE_CARRIER	TAIL_NUM	FL_
0	2016	1	1	1	5	DL	N836DN	
1	2016	1	1	1	5	DL	N964DN	
2	2016	1	1	1	5	DL	N813DN	
3	2016	1	1	1	5	DL	N587NW	
4	2016	1	1	1	5	DL	N836DN	

5 rows × 26 columns

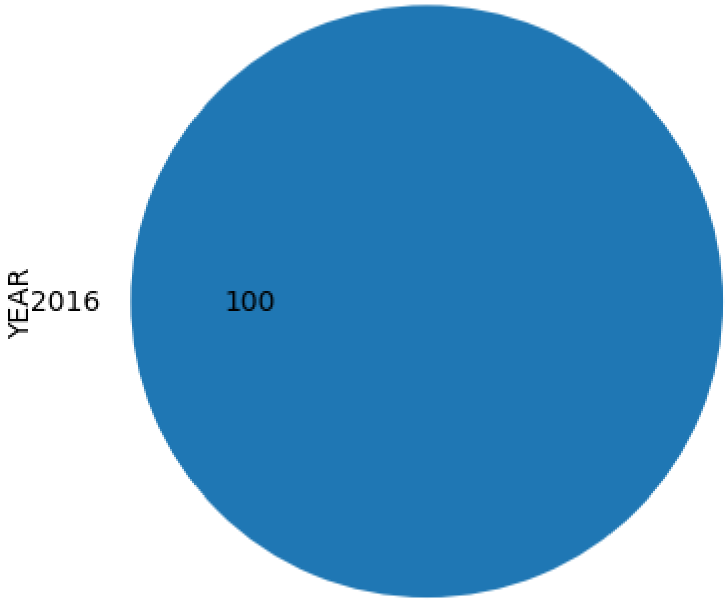
In [3]:
```
df.info()
```

```
<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
```

Performing Univariate Analysis

Using Pie Chart

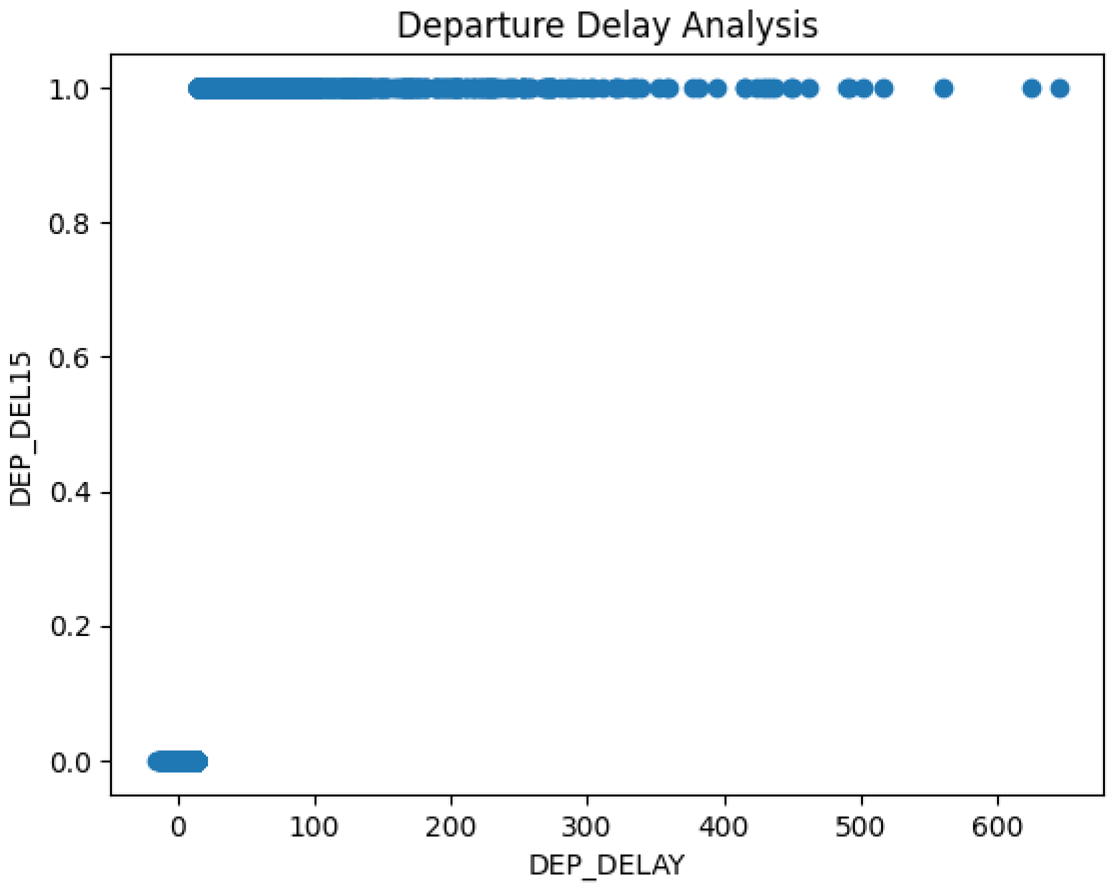
In [4]:
```
df['YEAR'].value_counts().plot(kind='pie', autopct='%.0f')
plt.show()
```



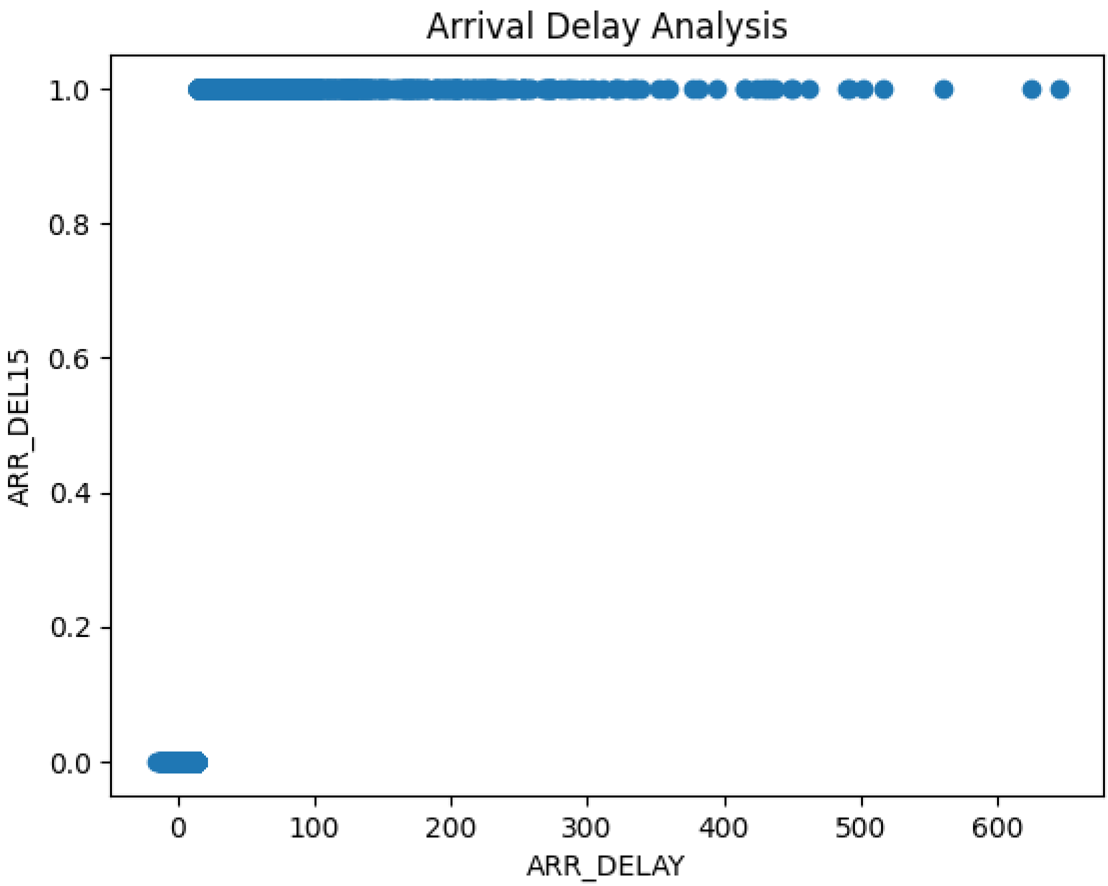
Performing Bivariate Analysis

Using scatterplot

In [5]:
```
plt.scatter(df.DEP_DELAY, df.DEP_DEL15)
plt.title('Departure Delay Analysis')
plt.xlabel('DEP_DELAY')
plt.ylabel('DEP_DEL15')
plt.show()
```

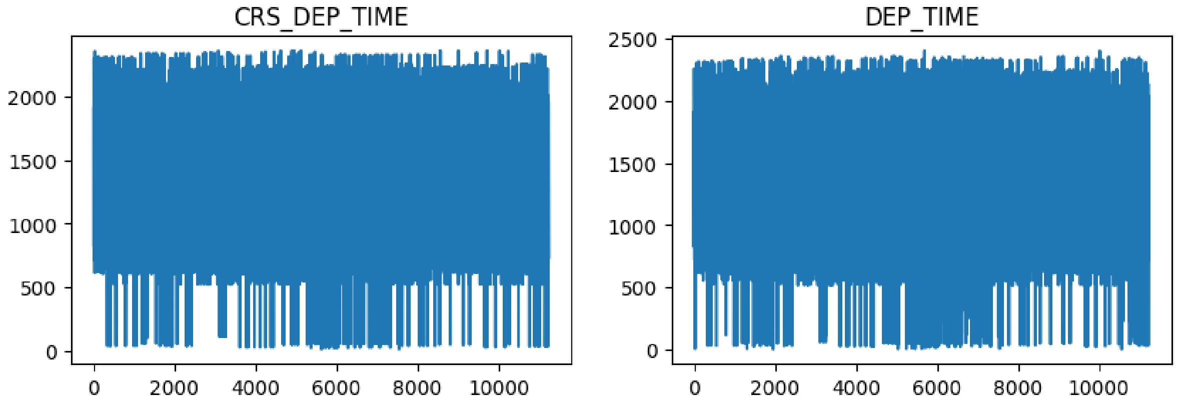


In [6]:
```
plt.scatter(df.DEP_DELAY, df.DEP_DEL15)
plt.title('Arrival Delay Analysis')
plt.xlabel('ARR_DELAY')
plt.ylabel('ARR_DEL15')
plt.show()
```

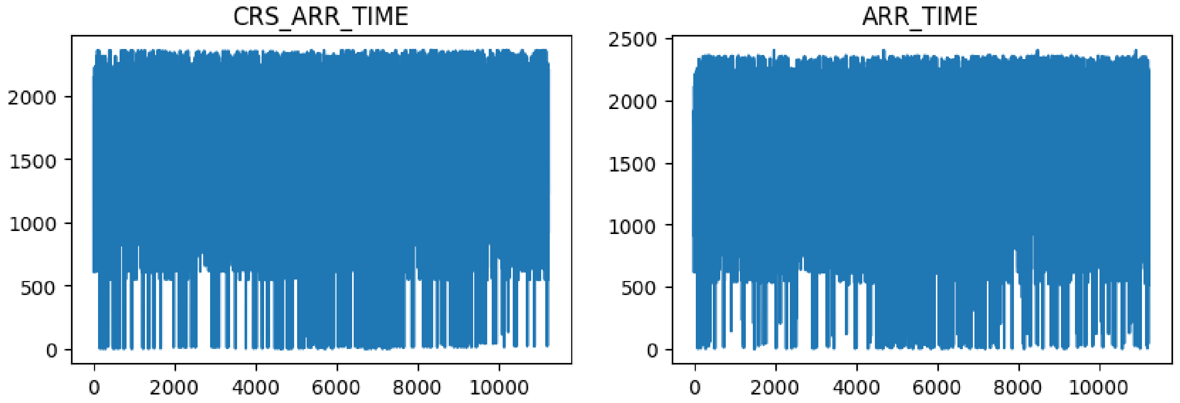


Using lineplots

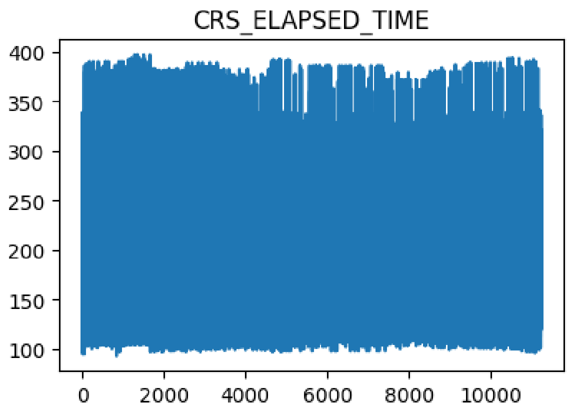
In [7]:
```
fig, ax = plt.subplots(figsize=(10, 3))
plt.subplot(1, 2, 1)
plt.title('CRS_DEP_TIME')
plt.plot(df.CRS_DEP_TIME)
plt.subplot(1, 2, 2)
plt.title('DEP_TIME')
plt.plot(df.DEP_TIME)
plt.show()
```

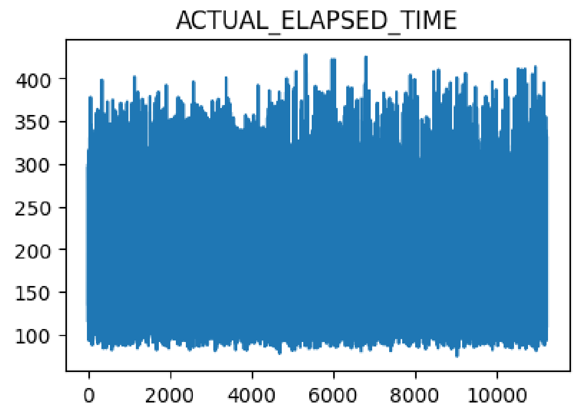


In [8]:
```
fig, ax = plt.subplots(figsize=(10, 3))
plt.subplot(1, 2, 1)
plt.title('CRS_ARR_TIME')
plt.plot(df.CRS_ARR_TIME)
plt.subplot(1, 2, 2)
plt.title('ARR_TIME')
plt.plot(df.ARR_TIME)
plt.show()
```



In [9]:
```
fig, ax = plt.subplots(figsize=(10, 3))
plt.subplot(1, 2, 1)
plt.title('CRS_ELAPSED_TIME')
plt.plot(df.CRS_ELAPSED_TIME)
plt.subplot(1, 2, 2)
plt.title('ACTUAL_ELAPSED_TIME')
plt.plot(df.ACTUAL_ELAPSED_TIME)
plt.show()
```

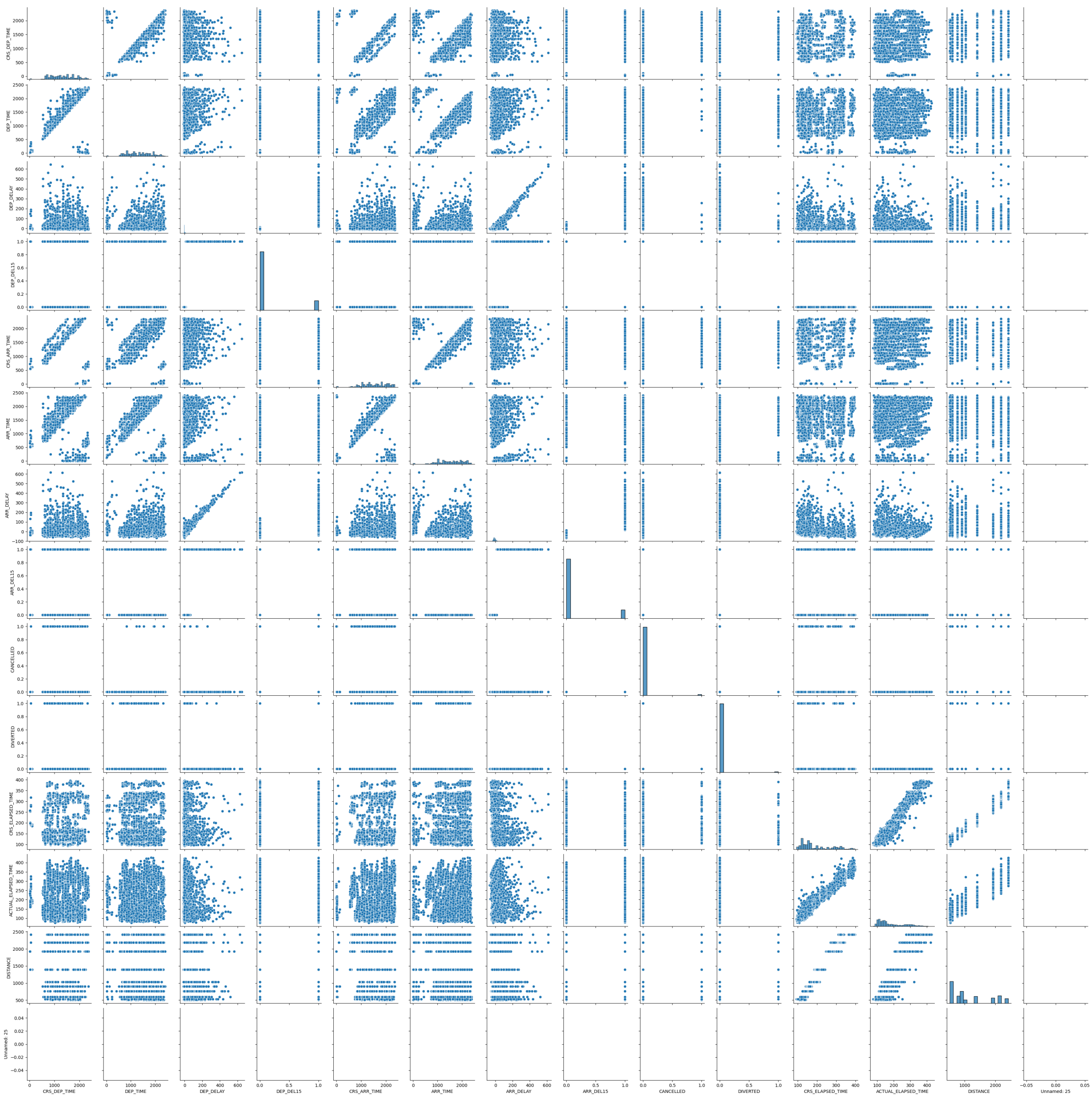




Performing Multivariate Analysis

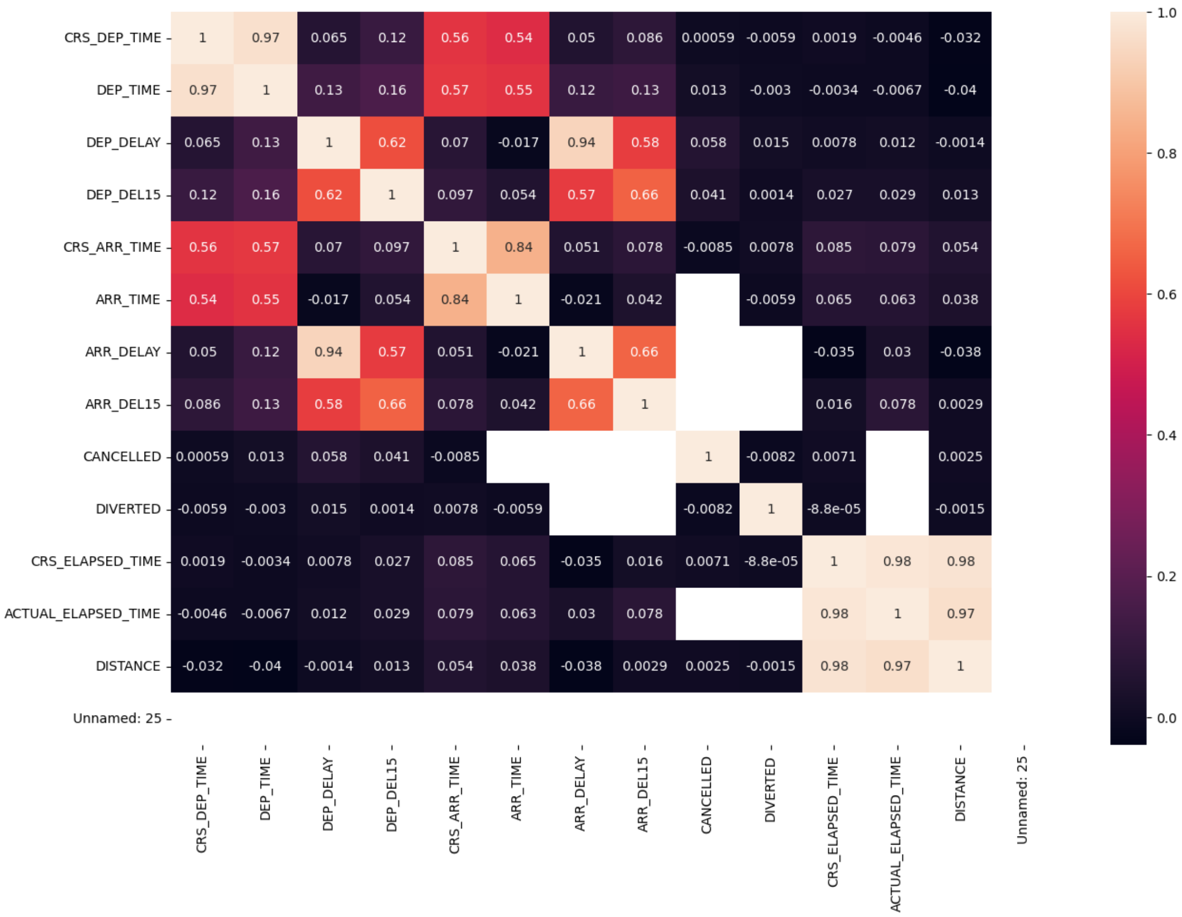
Using pairplot

In [10]:
```
sb.pairplot(df.iloc[:, 12:])
plt.show()
```



Using heatmap

In [11]:
```
fig, ax = plt.subplots(figsize=(15, 10))
sb.heatmap(df.iloc[:, 12:].corr(), annot=True, ax=ax)
plt.show()
```



Performing Descriptive Analysis

In [12]:
```
df.describe()
```

Out[12]:

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	FL_NUM	ORIG
count	11231.0	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	
mean	2016.0	2.544475	6.628973	15.790758	3.960199	1334.325617	
std	0.0	1.090701	3.354678	8.782056	1.995257	811.875227	
min	2016.0	1.000000	1.000000	1.000000	1.000000	7.000000	
25%	2016.0	2.000000	4.000000	8.000000	2.000000	624.000000	
50%	2016.0	3.000000	7.000000	16.000000	4.000000	1267.000000	
75%	2016.0	3.000000	9.000000	23.000000	6.000000	2032.000000	
max	2016.0	4.000000	12.000000	31.000000	7.000000	2853.000000	

8 rows × 22 columns

Dropping unnecessary columns

In [13]:
```
df = df[['FL_NUM', 'MONTH', 'DAY_OF_MONTH', 'DAY_OF_WEEK', 'ORIGIN', 'DEST', 'DEP_I
df.head()
```

Out[13]:

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	DEP_DEL15	CRS_ARR_TIM
0	1399	1	1	5	ATL	SEA	0.0	214
1	1476	1	1	5	DTW	MSP	0.0	143
2	1597	1	1	5	ATL	SEA	0.0	121
3	1768	1	1	5	SEA	MSP	0.0	133
4	1823	1	1	5	SEA	DTW	0.0	60

Handling Missing Values

Checking for null values

In [14]:
```
df.isnull().any()
```

Out[14]:
```
FL_NUM          False
MONTH           False
DAY_OF_MONTH    False
DAY_OF_WEEK     False
ORIGIN          False
DEST            False
DEP_DEL15        True
CRS_ARR_TIME    False
ARR_DEL15        True
dtype: bool
```

Replacing null values

In [15]:
```
df.fillna(df['DEP_DEL15'].mode()[0], inplace=True)
df.fillna(df['ARR_DEL15'].mode()[0], inplace=True)
```

Checking if the replacement is made

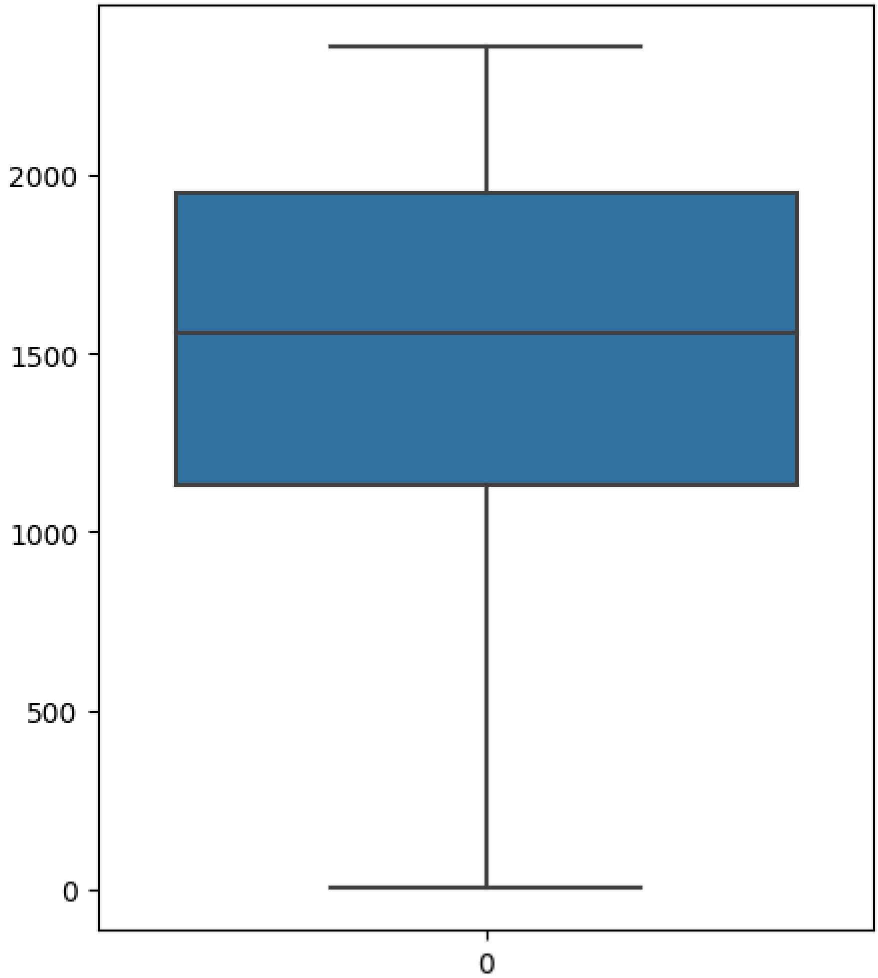
In [16]:
```
df.isnull().any()
```

Out[16]:
```
FL_NUM          False
MONTH           False
DAY_OF_MONTH    False
DAY_OF_WEEK     False
ORIGIN          False
DEST            False
DEP_DEL15       False
CRS_ARR_TIME    False
ARR_DEL15       False
dtype: bool
```

Handling Outliers

In [17]:
```
fig, ax = plt.subplots(figsize=(5, 6))
sb.boxplot(data=df['CRS_ARR_TIME'])
```

```
plt.show()
```



There are no outliers

Encoding

One Hot Encoding

In [18]:
```
df = pd.get_dummies(df, columns=['ORIGIN', 'DEST'])
df.head()
```

Out[18]:

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	DEP_DEL15	CRS_ARR_TIME	ARR_DEL15
0	1399	1	1	5	0.0	2143	0.0
1	1476	1	1	5	0.0	1435	0.0
2	1597	1	1	5	0.0	1215	0.0
3	1768	1	1	5	0.0	1335	0.0
4	1823	1	1	5	0.0	607	0.0

In [19]:
```
df.columns
```

Out[19]:
```
Index(['FL_NUM', 'MONTH', 'DAY_OF_MONTH', 'DAY_OF_WEEK', 'DEP_DEL15',
       'CRS_ARR_TIME', 'ARR_DEL15', 'ORIGIN_ATL', 'ORIGIN_DTW', 'ORIGIN_JFK',
       'ORIGIN_MSP', 'ORIGIN_SEA', 'DEST_ATL', 'DEST_DTW', 'DEST_JFK',
       'DEST_MSP', 'DEST_SEA'],
      dtype='object')
```

Splitting dataset into Independent and Dependent Variables

In [20]:
```
X = df.drop(columns=['ARR_DEL15'])
Y = df[['ARR_DEL15']]
```

Scaling the Independent Variables

In [21]:
```
scale_crs = StandardScaler()
X[['CRS_ARR_TIME']] = scale_crs.fit_transform(X[['CRS_ARR_TIME']])
scale_flnum = StandardScaler()
X[['FL_NUM']] = scale_flnum.fit_transform(X[['FL_NUM']])
X.head()
```

Out[21]:

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	DEP_DEL15	CRS_ARR_TIME	ORIGIN_ATL
0	0.079664	1	1	5	0.0	1.205371	1
1	0.174510	1	1	5	0.0	-0.203612	0
2	0.323555	1	1	5	0.0	-0.641431	1
3	0.534188	1	1	5	0.0	-0.402620	0
4	0.601935	1	1	5	0.0	-1.851405	0

In [22]:
```
X.FL_NUM.value_counts()
```

Out[22]:
```
-0.549771    98
-0.918071    96
 0.808873    96
-0.919302    95
-0.532526    94
             ..
 1.865732     1
 0.242258     1
 0.195451     1
 0.212695     1
 1.608292     1
Name: FL_NUM, Length: 690, dtype: int64
```

Converting the Independent and Dependent Variables to 1D Arrays

In [23]:
```
X = X.values
Y = Y.values
```

Splitting dataset into Train and Test datasets

In [24]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)
```

In [25]:
```
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

Out[25]:
```
((8984, 16), (2247, 16), (8984, 1), (2247, 1))
```

Building the Machine Learning Models

Logistic Regression

In [26]:
```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(max_iter=200)
log_reg.fit(X_train, Y_train.ravel())
```

Out[26]:
```
▾ LogisticRegression
LogisticRegression(max_iter=200)
```

Support Vector Machine (Classifier)

In [27]:
```
from sklearn.svm import SVC
svc = SVC()
svc.fit(X_train, Y_train.ravel())
```

Out[27]:
```
▾ SVC
SVC()
```

KNN Classifier

In [28]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train.ravel())
```

Out[28]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

Random Forest Classifier

In [29]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=15, max_depth=3)
rf.fit(X_train, Y_train.ravel())
```

Out[29]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=3, n_estimators=15)
```

Testing the Models

Logistic Regression

In [30]:
```
Y_pred_log_train = log_reg.predict(X_train)
Y_pred_log_test = log_reg.predict(X_test)
```

In [31]:
```
pd.DataFrame(Y_pred_log_train).value_counts()
```

Out[31]:
```
0.0    7733
1.0    1251
dtype: int64
```

In [32]:
```
pd.DataFrame(Y_pred_log_test).value_counts()
```

Out[32]:
```
0.0    1930
1.0     317
dtype: int64
```

Support Vector Machine (Classifier)

In [33]:
```
Y_pred_svc_train = svc.predict(X_train)
Y_pred_svc_test = svc.predict(X_test)
```

In [34]:
```
pd.DataFrame(Y_pred_svc_train).value_counts()
```

Out[34]:
```
0.0    7716
1.0    1268
dtype: int64
```

In [35]:
```
pd.DataFrame(Y_pred_svc_test).value_counts()
```

Out[35]:
```
0.0    1926
1.0     321
dtype: int64
```

KNN Classifier

In [36]:
```
Y_pred_knn_train = knn.predict(X_train)
Y_pred_knn_test = knn.predict(X_test)
```

In [37]:
```
pd.DataFrame(Y_pred_knn_train).value_counts()
```

Out[37]:
```
0.0    8599
1.0     385
dtype: int64
```

In [38]:
```
pd.DataFrame(Y_pred_knn_test).value_counts()
```

Out[38]:
```
0.0    2159
1.0      88
dtype: int64
```

Random Forest Classifier

In [39]:
```
Y_pred_rf_train = rf.predict(X_train)
Y_pred_rf_test = rf.predict(X_test)
```

In [40]:
```
pd.DataFrame(Y_pred_rf_train).value_counts()
```

Out[40]:
```
0.0    8517
1.0     467
dtype: int64
```

In [41]:
```
pd.DataFrame(Y_pred_rf_test).value_counts()
```

Out[41]:
```
0.0    2137
1.0     110
dtype: int64
```

Evaluating the ML Models using Metrics

Logistic Regression

Classification Report

In [42]:
```
print(classification_report(Y_test, Y_pred_log_test))
```

```
              precision    recall  f1-score   support

         0.0       0.97      0.94      0.95      1983
         1.0       0.64      0.77      0.70       264

    accuracy                           0.92      2247
   macro avg       0.80      0.85      0.83      2247
weighted avg       0.93      0.92      0.92      2247
```

Accuracy, Precision, Recall, F1 Score

In [43]:
```
acc_log = accuracy_score(Y_test, Y_pred_log_test)
prec_log, rec_log, f1_log, sup_log = precision_recall_fscore_support(Y_test, Y_pre
print('Accuracy Score =', acc_log)
print('Precision =', prec_log[0])
print('Recall =', rec_log[0])
print('F1 Score =', f1_log[0])
```

```
Accuracy Score = 0.9212283044058746
Precision = 0.9678756476683937
Recall = 0.9420070600100857
F1 Score = 0.9547661640684897
```

Checking for Overfitting and Underfitting

In [44]:
```
log_train_acc = accuracy_score(Y_train, Y_pred_log_train)
log_test_acc = accuracy_score(Y_test, Y_pred_log_test)
print('Training Accuracy =', log_train_acc)
print('Testing Accuracy =', log_test_acc)
```

```
Training Accuracy = 0.9200801424755121
Testing Accuracy = 0.9212283044058746
```

There is no big variation in the training and testing accuracy. Therefore, the Logistic Regression model is not overfit or underfit.

Confusion Matrix

In [45]:
```
pd.crosstab(Y_test.ravel(), Y_pred_log_test)
```

Out[45]:

col_0	0.0	1.0
row_0		
0.0	1868	115
1.0	62	202

Support Vector Machine (Classifier)

Classification Report

In [46]:
```
print(classification_report(Y_test, Y_pred_svc_test))
```

```
              precision    recall  f1-score   support

         0.0       0.97      0.94      0.95      1983
         1.0       0.63      0.77      0.69       264

    accuracy                           0.92      2247
   macro avg       0.80      0.85      0.82      2247
weighted avg       0.93      0.92      0.92      2247
```

Accuracy, Precision, Recall, F1 Score

In [47]:
```
acc_svc = accuracy_score(Y_test, Y_pred_svc_test)
prec_svc, rec_svc, f1_svc, sup_svc = precision_recall_fscore_support(Y_test, Y_pre
print('Accuracy Score =', acc_svc)
print('Precision =', prec_svc[0])
print('Recall =', rec_svc[0])
print('F1 Score =', f1_svc[0])
```

```
Accuracy Score = 0.9203382287494437
Precision = 0.9683281412253375
Recall = 0.940494200706001
F1 Score = 0.9542082374008697
```

Checking for Overfitting and Underfitting

In [48]:
```
svc_train_acc = accuracy_score(Y_train, Y_pred_svc_train)
svc_test_acc = accuracy_score(Y_test, Y_pred_svc_test)
print('Training Accuracy =', svc_train_acc)
print('Testing Accuracy =', svc_test_acc)
```

```
Training Accuracy = 0.9204140694568121
Testing Accuracy = 0.9203382287494437
```

There is no big variation in the training and testing accuracy. Therefore, the Support Vector Classifier model is not overfit or underfit.

Confusion Matrix

In [49]:
```
pd.crosstab(Y_test.ravel(), Y_pred_svc_test)
```

Out[49]:

col_0	0.0	1.0
row_0		
0.0	1865	118
1.0	61	203

KNN Classifier

Classification Report

In [50]:
```
print(classification_report(Y_test, Y_pred_knn_test))
```

```
              precision    recall  f1-score   support

         0.0       0.90      0.98      0.94      1983
         1.0       0.57      0.19      0.28       264

    accuracy                           0.89      2247
   macro avg       0.73      0.59      0.61      2247
weighted avg       0.86      0.89      0.86      2247
```

Accuracy, Precision, Recall, F1 Score

In [51]:
```
acc_knn = accuracy_score(Y_test, Y_pred_knn_test)
prec_knn, rec_knn, f1_knn, sup_knn = precision_recall_fscore_support(Y_test, Y_pre
print('Accuracy Score =', acc_knn)
print('Precision =', prec_knn[0])
print('Recall =', rec_knn[0])
print('F1 Score =', f1_knn[0])
```

```
Accuracy Score = 0.8878504672897196
Precision = 0.9008800370541917
Recall = 0.9808371154815936
F1 Score = 0.939159826170932
```

Checking for Overfitting and Underfitting

In [52]:
```
knn_train_acc = accuracy_score(Y_train, Y_pred_knn_train)
knn_test_acc = accuracy_score(Y_test, Y_pred_knn_test)
print('Training Accuracy =', knn_train_acc)
print('Testing Accuracy =', knn_test_acc)
```

```
Training Accuracy = 0.9065004452359751
Testing Accuracy = 0.8878504672897196
```

There is no big variation in the training and testing accuracy. Therefore, the KNN Classifier model is not overfit or underfit.

Confusion Matrix

In [53]:
```
pd.crosstab(Y_test.ravel(), Y_pred_knn_test)
```

Out[53]:

col_0	0.0	1.0
row_0		
0.0	1945	38
1.0	214	50

Random Forest Classifier

Classification Report

In [54]:
```
print(classification_report(Y_test, Y_pred_rf_test))
```

```
              precision    recall  f1-score   support

         0.0       0.91      0.98      0.94      1983
         1.0       0.66      0.28      0.39       264

    accuracy                           0.90      2247
   macro avg       0.79      0.63      0.67      2247
weighted avg       0.88      0.90      0.88      2247
```

Accuracy, Precision, Recall, F1 Score

In [55]:
```
acc_rf = accuracy_score(Y_test, Y_pred_rf_test)
prec_rf, rec_rf, f1_rf, sup_rf = precision_recall_fscore_support(Y_test, Y_pred_rf_
print('Accuracy Score =', acc_rf)
print('Precision =', prec_rf[0])
print('Recall =', rec_rf[0])
print('F1 Score =', f1_rf[0])
```

```
Accuracy Score = 0.8985313751668892
Precision = 0.9106223678053346
Recall = 0.9813414019162885
F1 Score = 0.9446601941747573
```

Checking for Overfitting and Underfitting

In [56]:
```
rf_train_acc = accuracy_score(Y_train, Y_pred_rf_train)
rf_test_acc = accuracy_score(Y_test, Y_pred_rf_test)
print('Training Accuracy =', rf_train_acc)
print('Testing Accuracy =', rf_test_acc)
```

```
Training Accuracy = 0.9002671415850401
Testing Accuracy = 0.8985313751668892
```

There is no big variation in the training and testing accuracy. Therefore, the Random Forest Classifier model is not overfit or underfit.

Confusion Matrix

In [57]:
```
pd.crosstab(Y_test.ravel(), Y_pred_rf_test)
```

Out[57]:

col_0	0.0	1.0
row_0		
0.0	1946	37
1.0	191	73

On comparing the four models built, based on the performance metrics it is clear that Support Vector Classifier gives the highest performance. Hence, that model is chosen for deployment

Dumping the Chosen Model into pkl file

In [58]:
```
joblib.dump(svc, 'model.pkl')
```

Out[58]:
```
['model.pkl']
```

Dumping the Scaling Modules into pkl file

In [59]:
```
pickle.dump(scale_crs, open('crs_scale.pkl', 'wb'))
pickle.dump(scale_flnum, open('flnum_scale.pkl', 'wb'))
```